INFORMATIK

augmented VISION

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Image processing with OpenCV and Python

Kripasindhu Sarkar
kripasindhu.sarkar@dfki.de

Kaiserslautern University,
DFKI – Deutsches Forschungszentrum für Künstliche Intelligenz
http://av.dfki.de

Some of the contents are taken from
Slides from Didier Stricker, SS16
Slides from Rahul Sukthankar, CMU
Images from OpenCV website
Example from Stanford CS231n

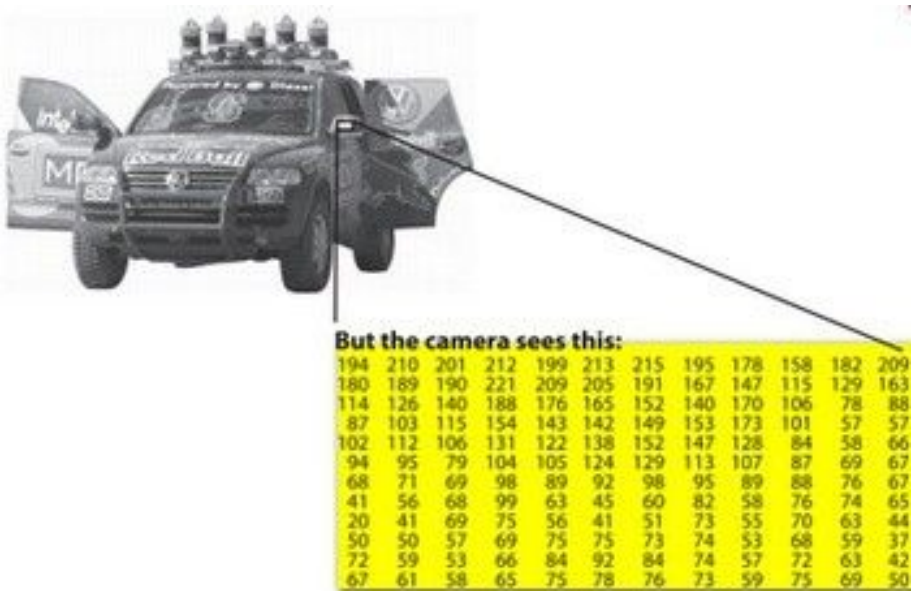
Outline

- Motivation - What is an Image?
- Tools required for Image processing
- Introduction to OpenCV
  - Intro
  - Installation + usage
  - Modules
  - …
- Introduction to Python
  - Numpy
  - Broadcasting
  - Simple maths
- Example applications
  - Edge detector
  - Thresholding, histogram normalization, etc
  - Filters
- Filters/Convolution
  - Background
  - maths

What is an image?

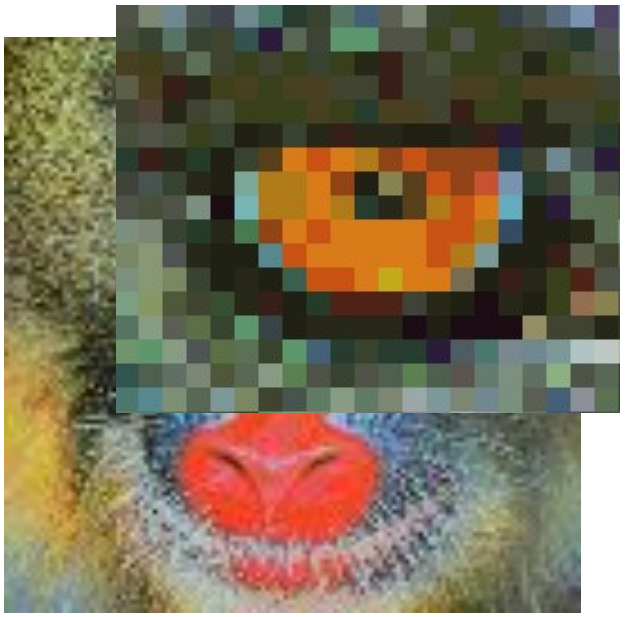
Image from Rahul Sukthankar, CMU

What is an image?



- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

What is an image?



- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

Slide from Rahul Sukthankar, CMU

What is an image?

- 2D array of pixels
- Binary image (bitmap)
  - Pixels are bits
- Grayscale image
  - Pixels are scalars
  - Typically 8 bits (0..255)
- Color images
  - Pixels are vectors
  - Order can vary: RGB, BGR
  - Sometimes includes Alpha

Slide from Rahul Sukthankar, CMU

Tools required for Image processing

- Good data structure for representing Grids/Matrix
- Efficient Operations on Matrix
  - Matrix multiplications
  - Broadcasting
  - Inverse… etc
  - Eigen analysis
  - SVD
- IO
  - Reading/writing of images and videos
- GUI
- Machine learning support
  - Optimization algorithms
  - Gradient descent etc
- Image processing algorithms
  - Filters/Convolutions
  - Transforms
  - Histogram Equalization
  - Specific CV algorithms - Canny Edge detectors, Flood filling, Scale space, Contours, Features, etc.

Introduction to OpenCV

- OpenCV stands for the Open Source Computer Vision Library.
- Founded at Intel in 1999
- OpenCV is free for commercial and research use.
- It has a BSD license. The library runs across many platforms and actively supports Linux, Windows and Mac OS.
- OpenCV was founded to advance the field of computer vision.
- It gives everyone a reliable, real time infrastructure to build on. It collects the most useful algorithms.

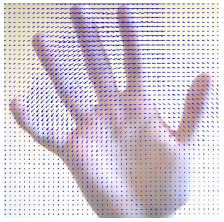
OpenCV Algorithm Modules Overview

HighGUI:
I/O, Interface

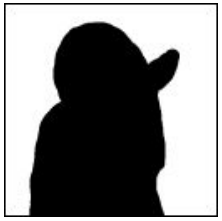
Image Processing

Transforms

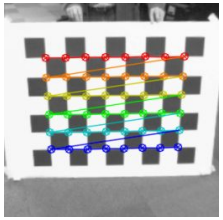
Fitting



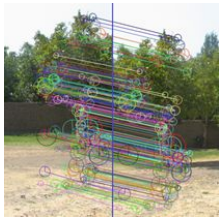
Optical Flow
Tracking



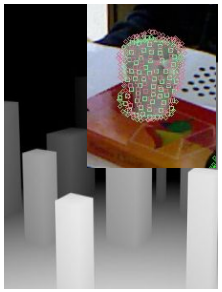
Segmentation



Calibration

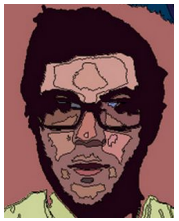


Features
VSLAM



Depth, Pose
Normals, Planes,
3D Features

Object recognition
Machine learning



Computational
Photography

CORE:
Data structures, Matrix math, Exceptions etc

Slide from G. Bradsky

OpenCV Overview:

opencv.willowgarage.com

> 500 functions

augmented VISION

Robot support

General Image Processing Functions

Image Pyramids

Geometric Descriptors

Segmentation

Camera Calibration, Stereo, 3D

Features

Transforms

Utilities and Data Structures

Tracking

Machine Learning:
- Detection,
- Recognition

Fitting

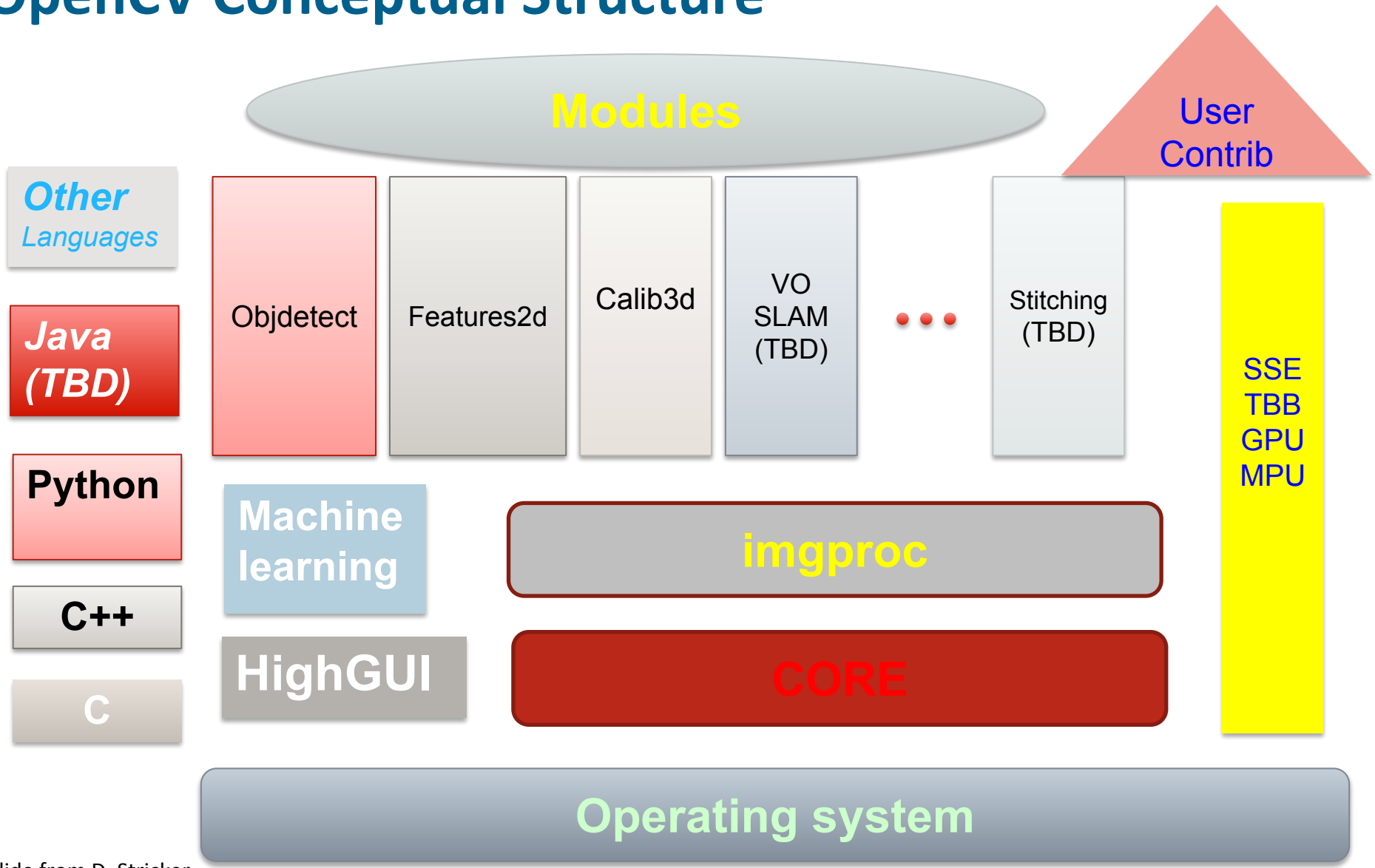
Matrix Math

Slide Courtesy *OpenCV Tutorial Gary Bradski*

augmented VISION

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

OpenCV Conceptual Structure



Slide from D. Stricker

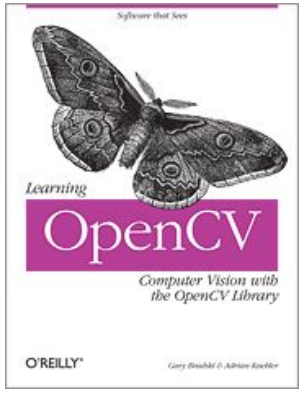
OpenCV – Getting Started

- Download OpenCV

  http://opencv.org
- Online Reference:

  http://docs.opencv.org

- Books?



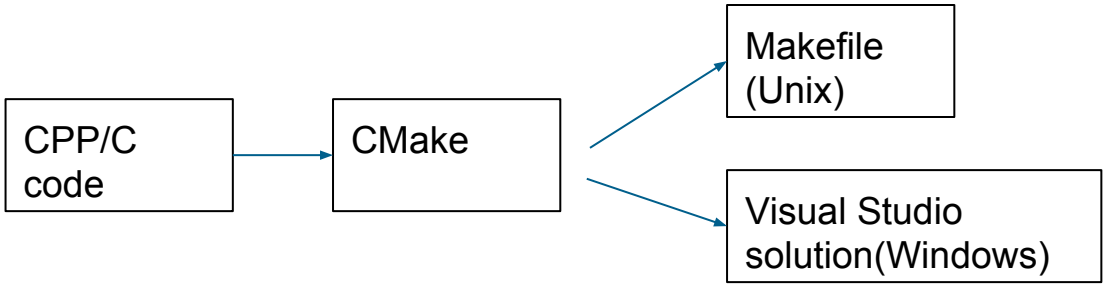
OpenCV – Installation

- Installation instruction
  - https://docs.opencv.org/3.0.0/d7/d9f/tutorial_linux_install.html

- Build from Source
  - Install the dependencies
  - Download the code (OpenCV)
  - Configure and Compile the code
  - Install

CMake Introduction

- "CMake is an open-source, **cross-platform** family of tools designed to **build**, test and package software"
  - Cross-platform Project generator



- Resources -
- Contents in the official website:
- https://cmake.org/cmake/help/v3.11/manual/cmake-buildsystem.7.html

OpenCV – Installation

- Installation instruction
  - https://docs.opencv.org/3.0.0/d7/d9f/tutorial_linux_install.html

- Build from Source
  - Install the dependencies
  - Download the code (OpenCV)
  - Configure and Compile the code
  - Install

OpenCV – Installation

- Installation instruction
  - https://docs.opencv.org/3.0.0/d7/d9f/tutorial_linux_install.html

- Build from Source
  - Install the dependencies
  - Download the code (OpenCV)
  - **Configure and Compile the code**
  - Install
    (make sure installation files
    libs and exes are in your default paths)
  - (make sure `cv2.so` is in default PYTHONPATH)

```
cd ~/opencv
mkdir build
cd build
```

```
cmake -D CMAKE_BUILD_TYPE=Release -D CMAKE_INSTALL_PREFIX=/usr/local ..
```

```
make -j7 # runs 7 jobs in parallel
```

OpenCV – Usage

DisplayImage.cpp:

```
#include <stdio.h>
#include <opencv2/opencv.hpp>

using namespace cv;

int main(int argc, char** argv )
{
    if ( argc != 2 )
    {
        printf("usage: DisplayImage.out <Image_Path>\n");
        return -1;
    }

    Mat image;
    image = imread( argv[1], 1 );

    if ( !image.data )
    {
        printf("No image data \n");
        return -1;
    }
    namedWindow("Display Image", WINDOW_AUTOSIZE );
    imshow("Display Image", image);

    waitKey(0);

    return 0;
}
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8)
project( DisplayImage )
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable( DisplayImage DisplayImage.cpp )
target_link_libraries( DisplayImage ${OpenCV_LIBS} )
```

OpenCV – Core

- All the OpenCV classes and functions are placed into the cv namespace
- **core** - compact module defining basic data structures and basic functions used by all other modules
- Basic image class cv::Mat
  - (https://docs.opencv.org/3.1.0/d3/d63/classcv_1_1Mat.html)

```
Mat A, C;                          // creates just the header parts
A = imread(argv[1], IMREAD_COLOR); // here we'll know the method used (allocate matrix)
Mat B(A);                                // Use the copy constructor
C = A;


//Initializers….
Mat             E               =              Mat::eye(4,              4,              CV_64F);
Mat             O               =              Mat::ones(2,             2,              CV_32F);
Mat                Z                =                 Mat::zeros(3,3,                 CV_8UC1);
```

The `Mat` class

- **Important things to know:**
  - Shallow copy: `Mat A = B;` does not copy data.
  - Deep copy: `clone()` and/or `B.copyTo(A);` (for ROIs, etc).
  - Most OpenCV functions can resize matrices if needed
- **Lots of convenient functionality (Matrix expressions):**
  - `s` is a `cv::Scalar`, α scalar (double)
  - Addition, scaling, …: `A±B, A±s, s±A, αA`
  - Per-element multiplication, division…: `A.mul(B), A/B, α/A`
  - Matrix multiplication, dot, cross product: `A*B, A.dot(B),`
  - `A.cross(B)`
  - Transposition, inversion: `A.t(), A.inv([method])`
  - And a few more.

HighGUI

HighGUI
Image I/O, rendering
Processing keyboard and other events, timeouts
Trackbars
Mouse callbacks
Video I/O

HighGUI

Example functions

- **void cv::namedWindow(const string& winname, int flags=WINDOW_AUTOSIZE);**
  - **Creates window accessed by its name. Window handles repaint, resize events. Its position is remembered in registry.**
- **void cv::destroyWindow(const string& winname);**
- **void cv::imshow(const string& winname, cv::Mat& mat);**
  - **Copies the image to window buffer, then repaints it when necessary. {8u|16s|32s|32f}{C1|3|4} are supported.**

HighGUI

- **Mat imread(const string& filename, int flags=1);**
  - loads image from file, converts to color or grayscale, if need, and returns it (or returns empty cv::Mat()).
  - image format is determined by the file contents.
- **bool imwrite(const string& filename, Mat& image);**
  - saves image to file, image format is determined from extension.
- **Example: convert JPEG to PNG**
  - cv::Mat img = cv::imread(“picture.jpeg”);
  - if(!img.empty()) cv::imwrite(“picture.png”, img);

HighGUI: Creating Interfaces I

- **Start off by creating a program that will constantly input images from a camera**

```
#include <opencv2/opencv.hpp>
int main( int argc, char* argv[] ) {
    cv::VideoCapture capture("filename.avi");
    if (!capture.isOpened()) return 1;
    cv::Mat frame;
    while (true) {
        capture >> frame; if(!frame.data) break;
        //process the frame here
    }
    capture.release();
    return 0;
}
```

Python and Numpy

- Python is a high-level, dynamically typed multiparadigm programming language.
- Python code is often said to be almost like pseudocode, since it allows you to express very powerful ideas in very few lines of code while being very readable.

Example:

```
def quicksort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quicksort(left) + middle + quicksort(right)

print(quicksort([3,6,8,10,1,2,1]))
# Prints "[1, 1, 2, 3, 6, 8, 10]"
```

Python examples in this section are taken from Stanford CS231n

Python basic types and containers

- Basic types - integers, floats, booleans, and strings...

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
```

- Containers - lists, dictionaries, sets, and tuples.

```
xs = [3, 1, 2]    # Create a list
print(xs, xs[2])  # Prints "[3, 1, 2] 2"
print(xs[-1])     # Negative indices count from the end of the list; prints "2"
```

List comprehension

```
nums = [0, 1, 2, 3, 4]
squares = [x ** 2 for x in nums]
print(squares)   # Prints [0, 1, 4, 9, 16]
```

Python basic types and containers

- Dictionaries

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))
```

- Tuples
  - ordered list of values

Python - Function

- Functions

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)  # Prints "HELLO, FRED!"
```

Python - Numpy

- Arrays
  - A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Python - Numpy

- Arrays - Slicing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

Python - Numpy

- Boolean array indexing

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

Python - Numpy

- Array operations

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))
# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

- Matrix multiplication - dot

```
x = np.array([[1,2],[3,4]])
v = np.array([9,10])

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
```

Python - Numpy

- Broadcasting

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)  # Prints "[[ 2  2  4]
          #         [ 5  5  7]
          #         [ 8  8 10]
          #         [11 11 13]]"
```

- Rules
  - If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.
  - The two arrays are said to be *compatible* in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.
  - The arrays can be broadcast together if they are compatible in all dimensions.
  - After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.
  - In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension

Python - Image operations

- Scipy library

```
from scipy.misc import imread, imsave, imresize

# Read an JPEG image into a numpy array
img = imread('assets/cat.jpg')
print(img.dtype, img.shape)  # Prints "uint8 (400, 248, 3)"

# We can tint the image by scaling each of the color channels
# by a different scalar constant. The image has shape (400, 248, 3);
# we multiply it by the array [1, 0.95, 0.9] of shape (3,);
# numpy broadcasting means that this leaves the red channel unchanged,
# and multiplies the green and blue channels by 0.95 and 0.9
# respectively.
img_tinted = img * [1, 0.95, 0.9]

# Resize the tinted image to be 300 by 300 pixels.
img_tinted = imresize(img_tinted, (300, 300))

# Write the tinted image back to disk
imsave('assets/cat_tinted.jpg', img_tinted)
```

Python - LA

- PCA/Eigen analysis

```
cov = np.cov((X - X.mean(axis=0)).transpose())
eigenvalues,eigenvectors = np.linalg.eig(cov)
```

Image processing Examples

- Image filtering/convolution operations
- Edge detection algorithms
- Object detection
- Segmentation
- …

Filtering - Theory

- ***Smoothing (blurring)*** **is a simple and frequently used operation**
  - **There are many reasons for smoothing, e.g. noise suppression**
  - **To perform a smoothing operation we will apply a *filter* to our image. The most common type of filters are *linear*, in which an output pixel's value (i.e. $g(i,j)$) is determined as a weighted sum of input pixel values (i.e. $f(i+k,j+l)$) :**

$$g(i,j) = \sum_{k,l} f(i+k,j+l)h(k,l)$$

  - $h(k,l)$ **is called the *kernel*, which is nothing more than the coefficients of the filter.**
  - **It helps to visualize a *filter* as a window of coefficients sliding across the image.**

Slide from D. Stricker

Normalized Box Filter

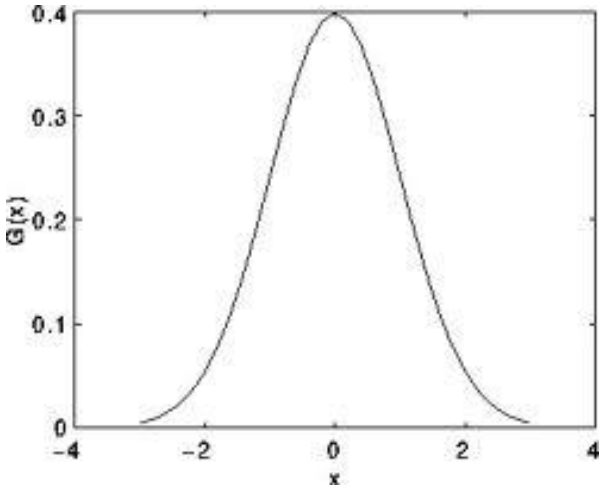
- This filter is the simplest of all! Each output pixel is the *mean* of its kernel neighbors (all of them contribute with equal weights)
- The kernel is below:

$$K = \frac{1}{K_{width} * K_{height}} \begin{bmatrix} 1 & \dots & 1 \\ \dots & \dots & \dots \\ 1 & \dots & 1 \end{bmatrix}$$

Slide from D. Stricker

Gaussian Filter I

- **Probably the most useful filter (although not the fastest). Gaussian filtering is done by convolving each point in the input array with a *Gaussian kernel*.**
- **1D *Gaussian kernel***



Slide from D. Stricker

Gaussian Filter II

- Pixel located in the middle has the biggest weight.
- The weight of its neighbors decreases as the spatial distance between them and the center pixel increases.
- 2D *Gaussian kernel*

$$G_0(x, y) = Ae^{\frac{-(x-\mu_x)^2}{2\sigma_x^2} + \frac{-(y-\mu_y)^2}{2\sigma_y^2}}$$

- where μ is the mean (the peak) and σ represents the variance (per each of the variables x and y)

Slide from D. Stricker

Median filter

- The median filter run through each element of the signal (in this case the image) and replace each pixel with the median of its neighboring pixels (located in a square neighborhood around the evaluated pixel).
- The median of a finite list of numbers can be found by arranging all the observations from lowest value to highest value and picking the middle one.

Slide from D. Stricker

Usage examples

- **Box filter**

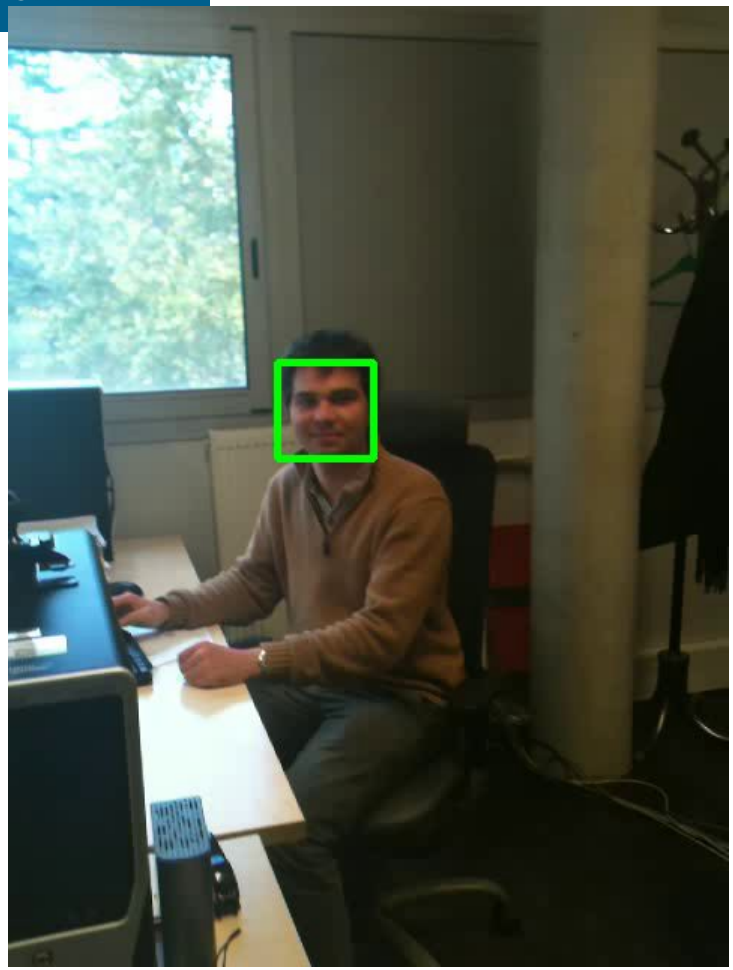
```
blur(src, dst, Size( filt_size_x, filt_size_y), Point(-1,-1));
```

  - *src*: Source image
  - *dst*: Destination image
  - *Size(w,h)*: Defines the size of the kernel to be used (of width *w* pixels and height *h* pixels)
  - *Point(-1, -1)*: Indicates where the anchor point (the pixel evaluated) is located with respect to the neighborhood. If there is a negative value, then the center of the kernel is considered the anchor point.

- **Gaussian blur**

```
GaussianBlur( src, dst, Size(filt_size_x, filt_size_y ), 0, 0 );
```

  - *Size(w, h)*: The size of the kernel to be used (the neighbors to be considered). and have to be odd and positive numbers otherwise the size will be calculated using the and arguments.
  - *sigma_x:* The standard deviation in x. Writing 0 implies that is calculated using kernel size.
  - *sigma_y:* The standard deviation in y. Writing 0 implies that is calculated using kernel size.

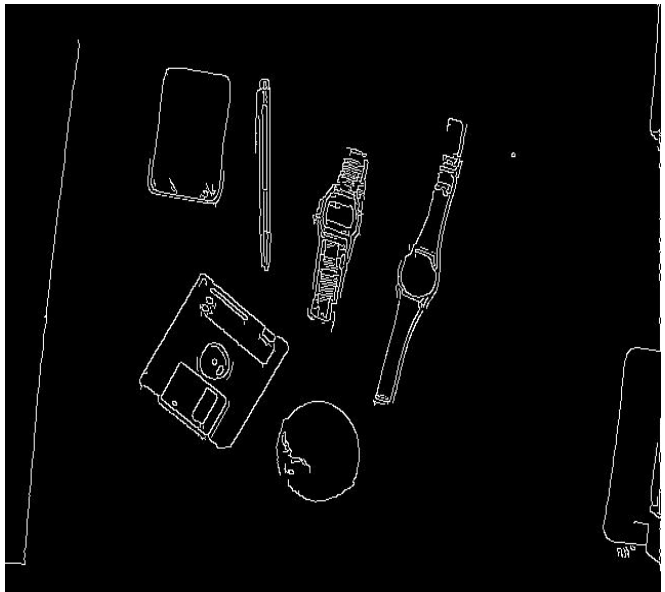


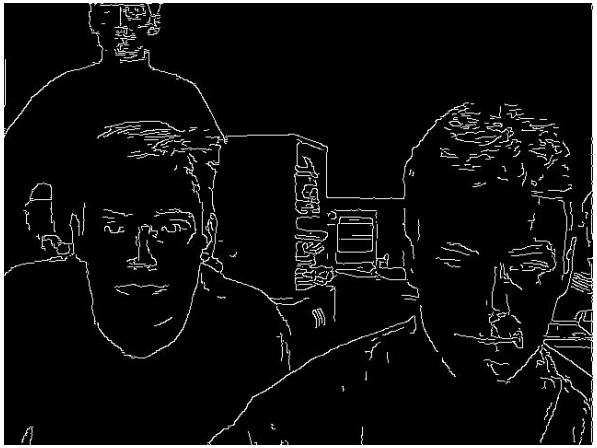
Th

Image processing Examples

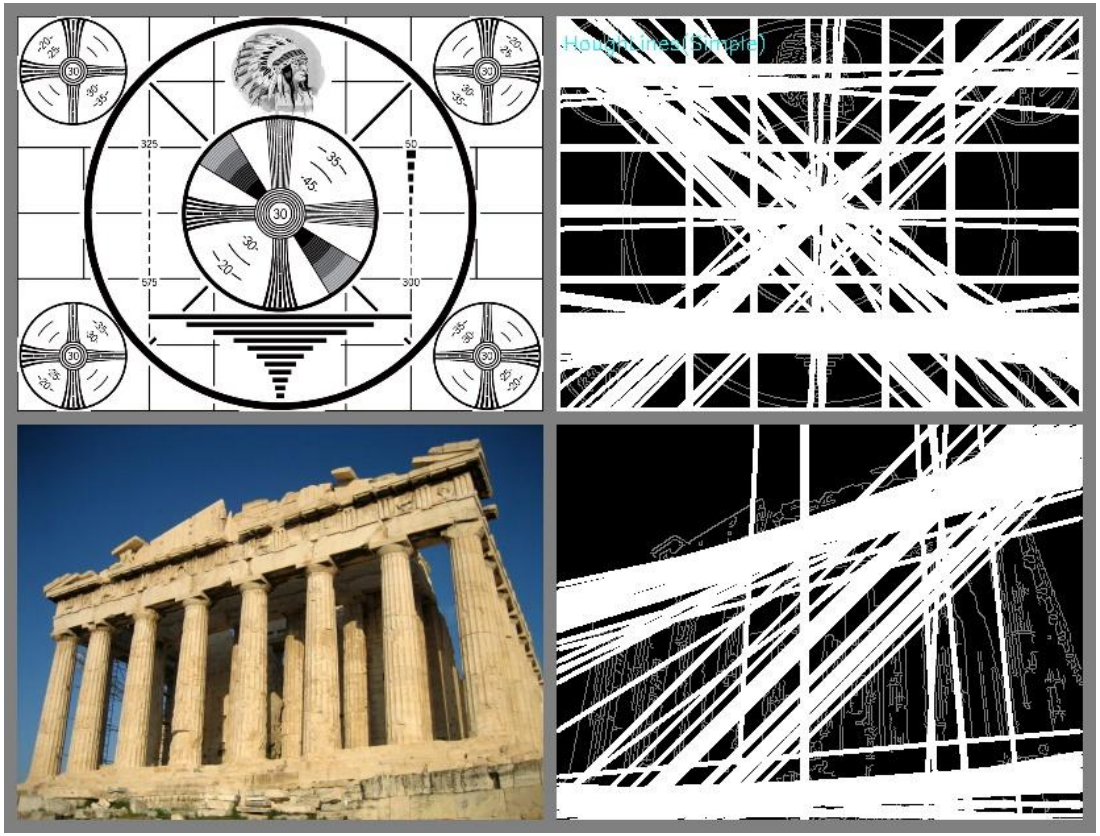
- Image filtering/convolution operations
- **Edge detection algorithms**
- **Object detection**
- **Segmentation**
- **…**

Canny Edge Detector





Hough Transform



Gary Bradski, Adrian Kahler
2008

Scale Space

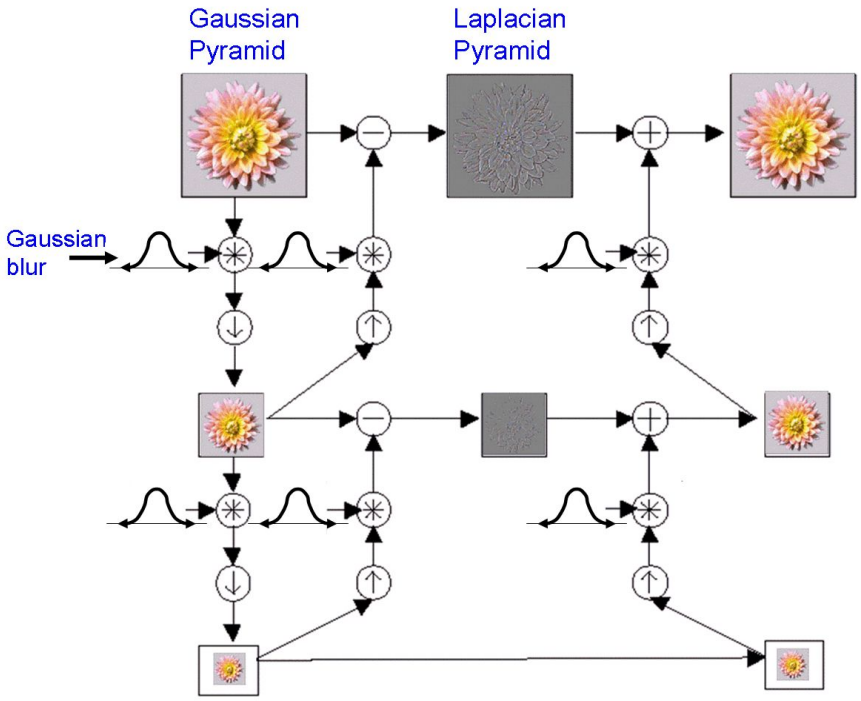
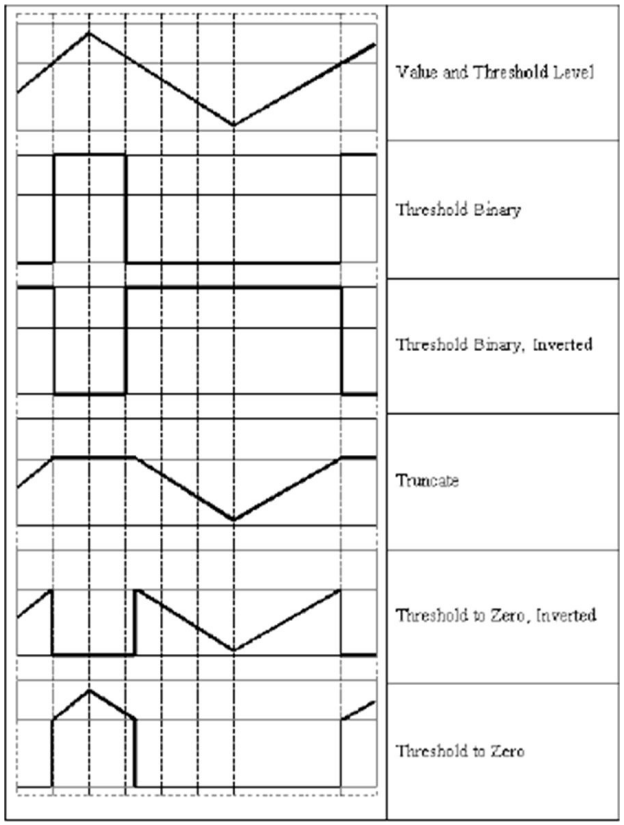


Chart by Gary Bradski, 2005

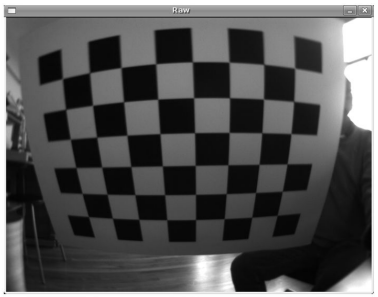
```
void cvPyrDown(
        IplImage* src,
        IplImage* dst,
        IplFilter    filter =
IPL_GAUSSIAN_5x5);
```

```
void cvPyrUp(
        IplImage* src,
        IplImage* dst,
        IplFilter    filter =
IPL_GAUSSIAN_5x5);
```

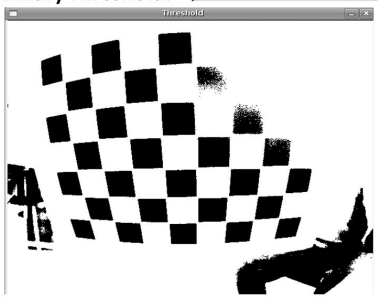
Thresholds



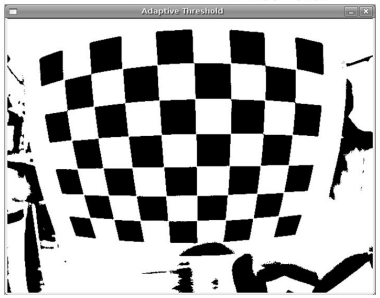
Source Image:



Binary Threshold:

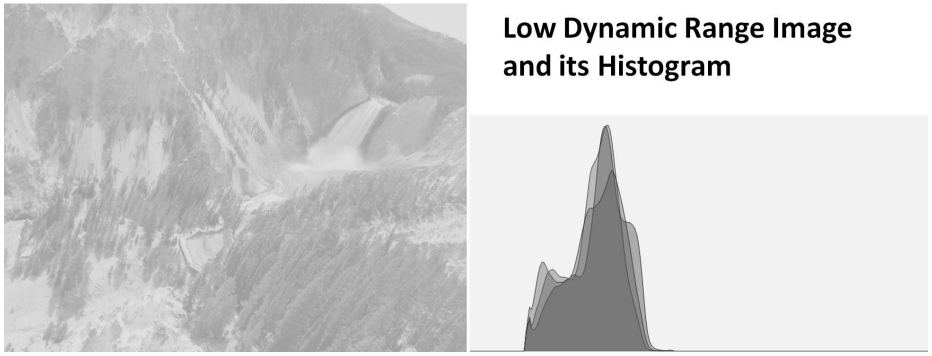


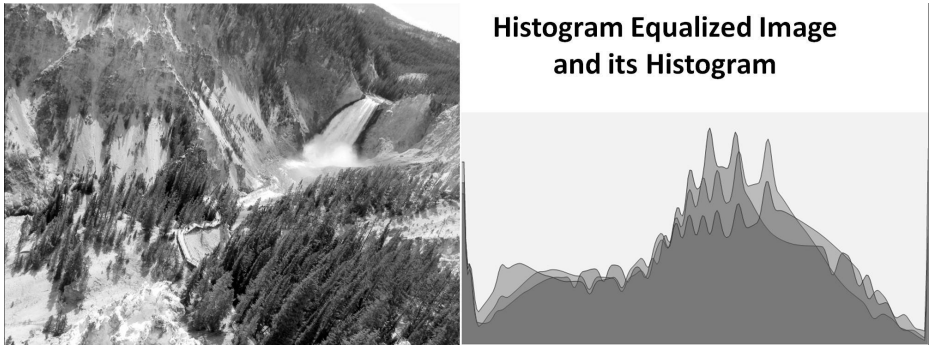
Adaptive Binary Threshold:



Screen shots by Gary Bradski, 2005

Histogram Equalization





Screen shots by Gary Bradski, 2005

Contours

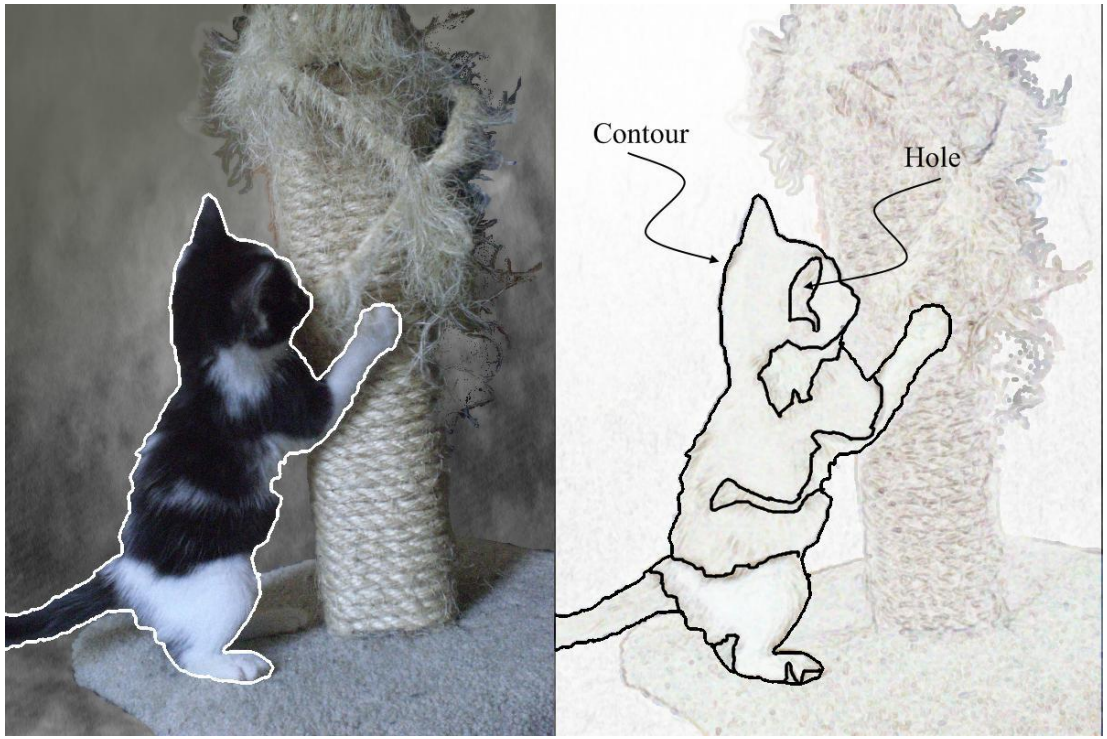
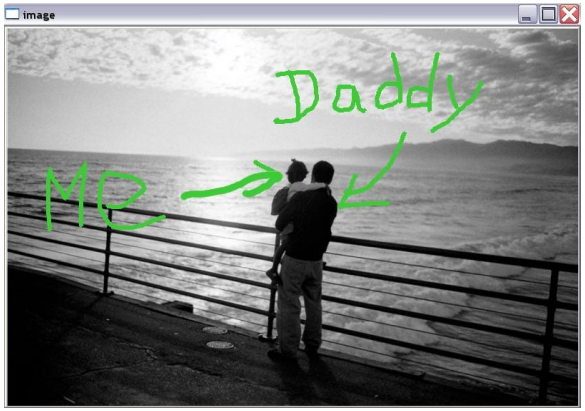
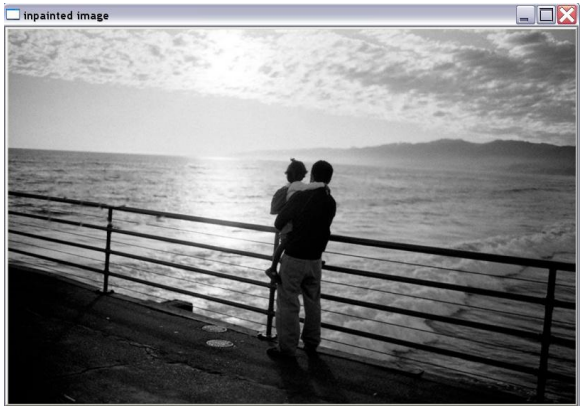


Image textures

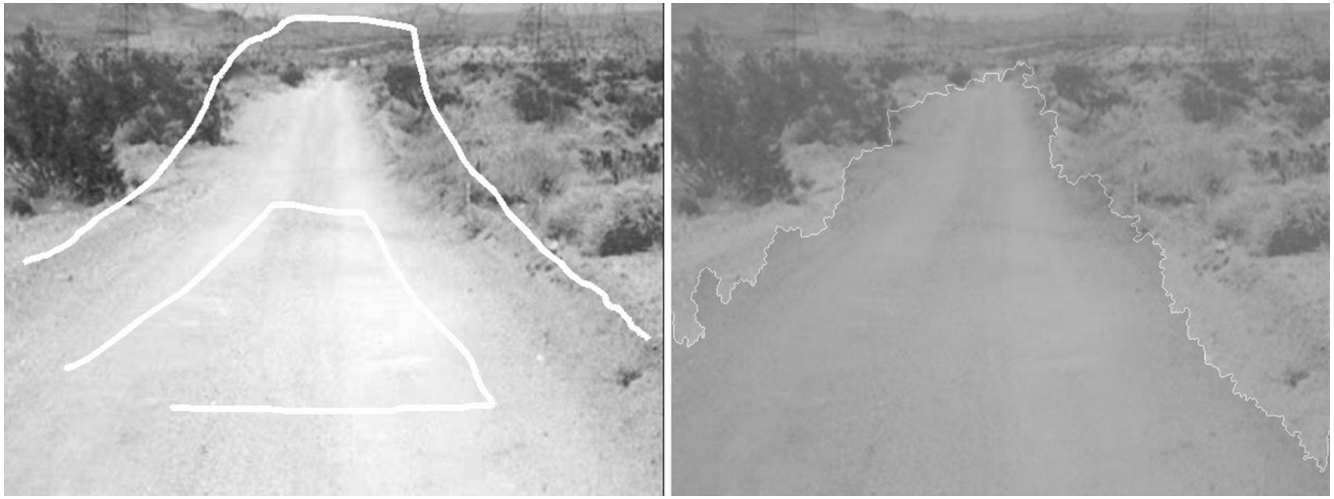
- **Inpainting:**
- **Removes damage to images, in this case, it removes the text.**





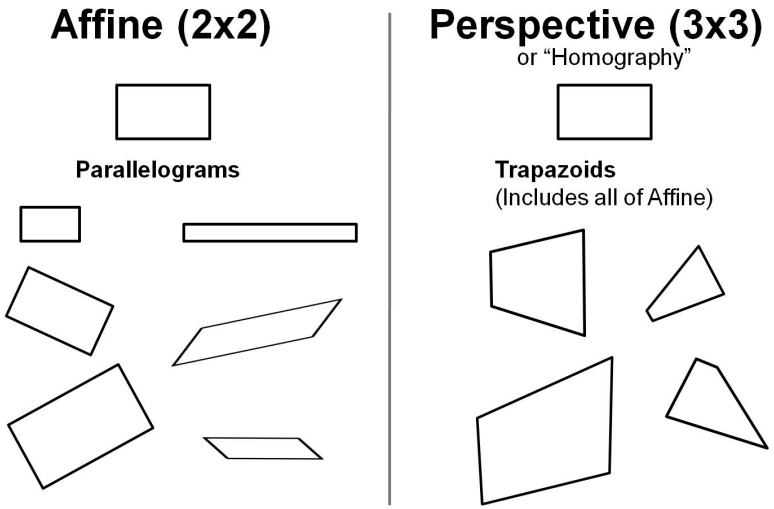
Segmentation

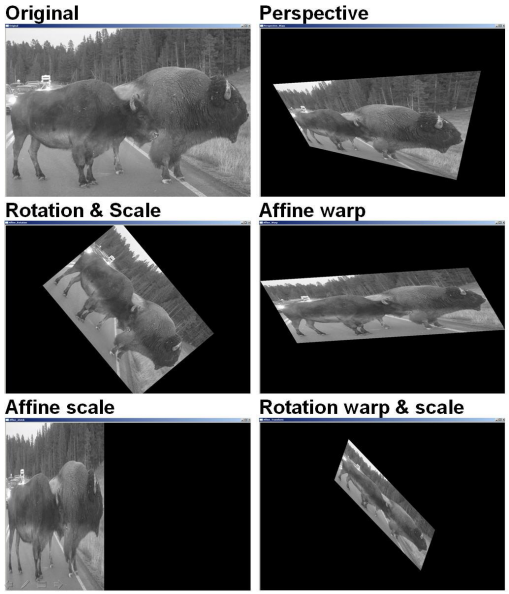
- Pyramid, mean-shift, graph-cut
- Here: Watershed



Screen shots by Gary Bradski, 2005

Projections





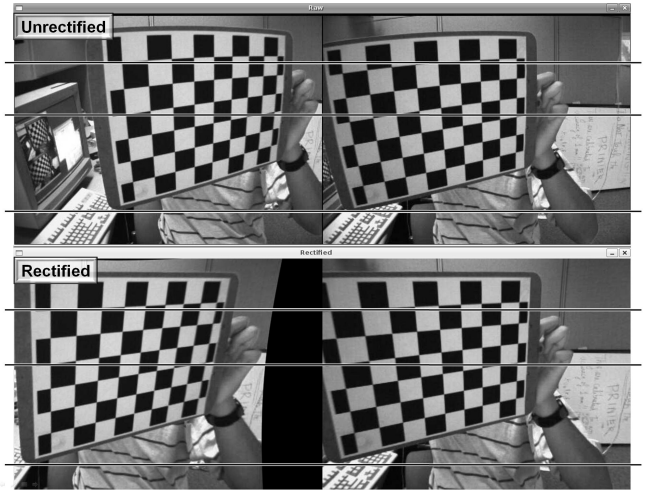
Screen shots by Gary Bradski, 2005

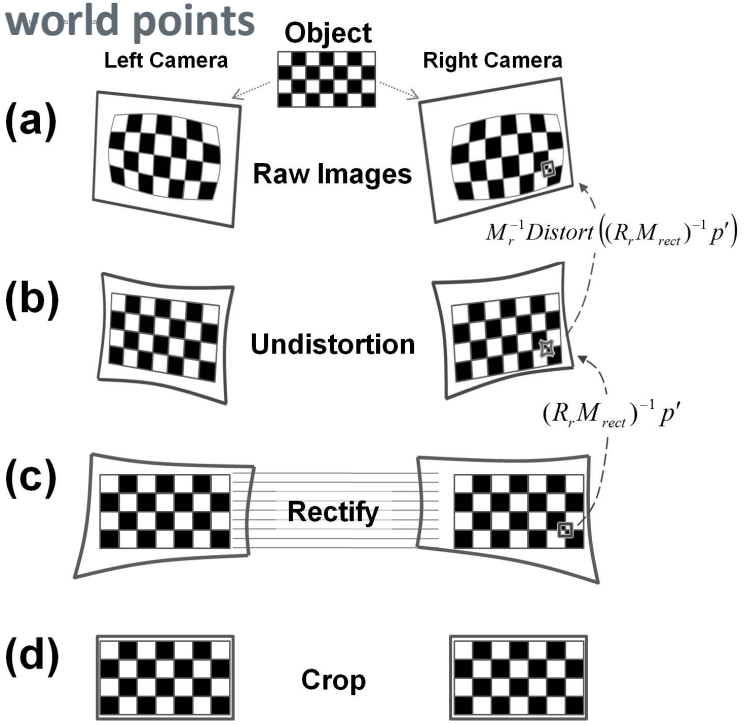
augmented VISION

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Stereo Rectification

- **Algorithm steps are shown at right:**
- **Goal:**
  - Each row of the image contains the same world points
  - "Epipolar constraint"



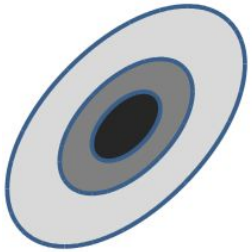


augmented VISION

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Features2d contents

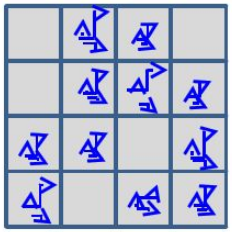
Detection



Detectors available

- SIFT
- SURF
- FAST
- STAR
- MSER
- HARRIS
- GFTT (Good Features To Track)

Description



Descriptors available

- SIFT
- SURF
- Calonder
- Ferns
- One way

Matching

Matchers available

- BruteForce
- FlannBased
- BOW

Matches filters

(under construction)

- Cross check
- Ratio check

Slide from D. Stricker

OpenCV summary

- **Following slides group together important methods/classes in OpenCV and summarises them.**
  - **Use them as a rough outline - the main source of information still should be from the official documentation page.**

Matrix Manipulation

`src.copyTo(dst)`	Copy matrix to another one
`src.convertTo(dst,type,scale,shift)`	Scale and convert to another datatype
`m.clone()`	Make deep copy of a matrix
`m.reshape(nch,nrows)`	Change matrix dimensions and/or number of channels without copying data
`m.row(i), m.col(i)`	Take a matrix row/column
`m.rowRange(Range(i1,i2))` `m.colRange(Range(j1,j2))`	Take a matrix row/column span
`m.diag(i)`	Take a matrix diagonal
`m(Range(i1,i2),Range(j1,j2))` `m(roi)`	Take a submatrix
`m.repeat(ny,nx)`	Make a bigger matrix from a smaller one
`flip(src,dst,dir)`	Reverse the order of matrix rows and/or columns
`split(...)`	Split multi-channel matrix into separate channels
`merge(...)`	Make a multi-channel matrix out of the separate channels
`mixChannels(...)`	Generalized form of split() and merge()
`randShuffle(...)`	Randomly shuffle matrix elements

Example 1. Smooth image ROI in-place

```
Mat imgroi = image(Rect(10, 20, 100, 100));
GaussianBlur(imgroi, imgroi, Size(5, 5), 1.2, 1.2);
```

Example 2. Somewhere in a linear algebra algorithm

```
m.row(i) += m.row(j)*alpha;
```

Example 3. Copy image ROI to another image with conversion

```
Rect r(1, 1, 10, 20);
Mat dstroi = dst(Rect(0,10,r.width,r.height));
src(r).convertTo(dstroi, dstroi.type(), 1, 0);
```

Slide from D. Stricker

Simple Matrix Operations

- `add()`, `subtract()`, `multiply()`, `divide()`, `absdiff()`, `bitwise_and()`, `bitwise_or()`, `bitwise_xor()`, `max()`, `min()`, `compare()`

  – correspondingly, addition, subtraction, element-wise multiplication ... comparison of two matrices or a matrix and a scalar.

Example. Alpha compositing function:

```
void alphaCompose(const Mat& rgba1,
    const Mat& rgba2, Mat& rgba_dest)
{
    Mat a1(rgba1.size(), rgba1.type()), ra1;
    Mat a2(rgba2.size(), rgba2.type());
    int mixch[]={3, 0, 3, 1, 3, 2, 3, 3};
    mixChannels(&rgba1, 1, &a1, 1, mixch, 4);
    mixChannels(&rgba2, 1, &a2, 1, mixch, 4);
    subtract(Scalar::all(255), a1, ra1);
    bitwise_or(a1, Scalar(0,0,0,255), a1);
    bitwise_or(a2, Scalar(0,0,0,255), a2);
    multiply(a2, ra1, a2, 1./255);
    multiply(a1, rgba1, a1, 1./255);
    multiply(a2, rgba2, a2, 1./255);
    add(a1, a2, rgba_dest);
}
```

- `sum()`, `mean()`, `meanStdDev()`, `norm()`, `countNonZero()`, `minMaxLoc()`,

  – various statistics of matrix elements.

- `exp()`, `log()`, `pow()`, `sqrt()`, `cartToPolar()`, `polarToCart()`

  – the classical math functions.

- `scaleAdd()`, `transpose()`, `gemm()`, `invert()`, `solve()`, `determinant()`, `trace()` `eigen()`, `SVD`,

  – the algebraic functions + SVD class.

- `dft()`, `idft()`, `dct()`, `idct()`,

  – discrete Fourier and cosine transformations

For some operations a more convenient algebraic notation can be used, for example:

```
Mat delta = (J.t()*J + lambda*
   Mat::eye(J.cols, J.cols, J.type()))
   .inv(CV_SVD)*(J.t()*err);
```

implements the core of Levenberg-Marquardt optimization algorithm.

Slide from D. Stricker

Simple Image Processing

`filter2D()`	Non-separable linear filter
`sepFilter2D()`	Separable linear filter
`boxFilter()`, `GaussianBlur()`, `medianBlur()`, `bilateralFilter()`	Smooth the image with one of the linear or non-linear filters
`Sobel()`, `Scharr()`	Compute the spatial image derivatives
`Laplacian()`	compute Laplacian: $\Delta I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$
`erode()`, `dilate()`	Erode or dilate the image

Example. Filter image in-place with a 3x3 high-pass kernel
(preserve negative responses by shifting the result by 128):

```
filter2D(image, image, image.depth(), (Mat_<float>(3,3)<<
    -1, -1, -1, -1, 9, -1, -1, -1, -1), Point(1,1), 128);
```

Slide from D. Stricker

Image Conversions

`resize()`	Resize image
`getRectSubPix()`	Extract an image patch
`warpAffine()`	Warp image affinely
`warpPerspective()`	Warp image perspectively
`remap()`	Generic image warping
`convertMaps()`	Optimize maps for a faster remap() execution

Example. Decimate image by factor of $\sqrt{2}$:

```
Mat dst; resize(src, dst, Size(), 1./sqrt(2), 1./sqrt(2))
```

`cvtColor()`	Convert image from one color space to another
`threshold()`, `adaptivethreshold()`	Convert grayscale image to binary image using a fixed or a variable threshold
`floodFill()`	Find a connected component using region growing algorithm
`integral()`	Compute integral image
`distanceTransform()`	build distance map or discrete Voronoi diagram for a binary image.
`watershed()`, `grabCut()`	marker-based image segmentation algorithms. See the samples `watershed.cpp` and `grabcut.cpp`.

Slide from D. Stricker

Histogram

Histograms

`calcHist()`	Compute image(s) histogram
`calcBackProject()`	Back-project the histogram
`equalizeHist()`	Normalize image brightness and contrast
`compareHist()`	Compare two histograms

Example. Compute Hue-Saturation histogram of an image:

```
Mat hsv, H; MatND tempH;
cvtColor(image, hsv, CV_BGR2HSV);
int planes[]={0, 1}, hsize[] = {32, 32};
calcHist(&hsv, 1, planes, Mat(), tempH, 2, hsize, 0);
H = tempH;
```

Slide from D. Stricker

Input/Output

Writing and reading raster images

```
imwriteimwrite("myimage.jpg", image);
Mat image_color_copy = imread("myimage.jpg", 1);
Mat image_grayscale_copy = imread("myimage.jpg", 0);
```

The functions can read/write images in the following formats: ***BMP (.bmp), JPEG (.jpg, .jpeg), TIFF (.tif, .tiff), PNG (.png), PBM/PGM/PPM (.p?m), Sun Raster (.sr), JPEG 2000 (.jp2).*** *Every format supports 8-bit, 1- or 3-channel images. Some formats (PNG, JPEG 2000) support 16 bits per channel.*

Reading video from a file or from a camera

```
VideoCapture cap;
if(argc > 1) cap.open(string(argv[1])); else cap.open(0);
Mat frame; namedWindow("video", 1);
for(;;) {
   cap >> frame; if(!frame.data) break;
   imshow("video", frame); if(waitKey(30) >= 0) break;
}
```

Slide from D. Stricker

Serialization I/O

Data I/O

XML/YAML storages are collections (possibly nested) of scalar values, structures and heterogeneous lists.

Writing data to YAML (or XML)

```
// Type of the file is determined from the extension
```

```
FileStorage fs("test.yml", FileStorage::WRITE);
fs << "i" << 5 << "r" << 3.1 << "str" << "ABCDEFGH";
fs << "mtx" << Mat::eye(3,3,CV_32F);
fs << "mylist" << "[" << CV_PI << "1+1" <<
   "{:" << "month" << 12 << "day" << 31 << "year"
   << 1969 << "}" << "]";
fs << "mystruct" << "{" << "x" << 1 << "y" << 2 <<
   "width" << 100 << "height" << 200 << "lbp" << "[:";
const uchar arr[] = {0, 1, 1, 0, 1, 1, 0, 1};
fs.writeRaw("u", arr, (int)(sizeof(arr)/sizeof(arr[0])));
fs << "]" << "}";
```

Scalars (integers, floating-point numbers, text strings), matrices, STL vectors of scalars and some other types can be written to the file storages using << operator

Slide from D. Stricker

Serialization I/O

Reading the data back

```
;// Type of the file is determined from the content
FileStorage fs("test.yml", FileStorage::READ);
int i1 = (int)fs["i"]; double r1 = (double)fs["r"];
string str1 = (string)fs["str"];
Mat M; fs["mtx"] >> M;
FileNode tl = fs["mylist"];
CV_Assert(tl.type() == FileNode::SEQ && tl.size() == 3);
double tl0 = (double)tl[0]; string tl1 = (string)tl[1];
int m = (int)tl[2]["month"], d = (int)tl[2]["day"];
int year = (int)tl[2]["year"];
FileNode tm = fs["mystruct"];
Rect r; r.x = (int)tm["x"], r.y = (int)tm["y"];
r.width = (int)tm["width"], r.height = (int)tm["height"];
int lbp_val = 0;
FileNodeIterator it = tm["lbp"].begin();
for(int k = 0; k < 8; k++, ++it)
   lbp_val |= ((int)*it) << k;
```

Scalars are read using the corresponding FileNode's cast operators. Matrices and some other types are read using >> operator. Lists can be read using FileNodeIterator's.

Slide from D. Stricker

GUI ("HighGUI")

`namedWindow(winname,flags)`	Create named highgui window
`destroyWindow(winname)`	Destroy the specified window
`imshow(winname, mtx)`	Show image in the window
`waitKey(delay)`	Wait for a key press during the specified time interval (or forever). Process events while waiting. *Do not forget to call this function several times a second in your code.*
`createTrackbar(...)`	Add trackbar (slider) to the specified window
`setMouseCallback(...)`	Set the callback on mouse clicks and movements in the specified window

See `camshiftdemo.c` and other OpenCV samples on how to use the GUI functions.

Slide from D. Stricker

Camera Calibration, Pose, Stereo

`calibrateCamera()`	Calibrate camera from several views of a calibration pattern.
`findChessboardCorners()`	Find feature points on the checkerboard calibration pattern.
`solvePnP()`	Find the object pose from the known projections of its feature points.
`stereoCalibrate()`	Calibrate stereo camera.
`stereoRectify()`	Compute the rectification transforms for a calibrated stereo camera.
`initUndistortRectifyMap()`	Compute rectification map (for `remap()`) for each stereo camera head.
`StereoBM`, `StereoSGBM`	The stereo correspondence engines to be run on rectified stereo pairs.
`reprojectImageTo3D()`	Convert disparity map to 3D point cloud.
`findHomography()`	Find best-fit perspective transformation between two 2D point sets.

To calibrate a camera, you can use `calibration.cpp` or `stereo_calib.cpp` samples. To get the disparity maps and the point clouds, use `stereo_match.cpp` sample.

Slide from D. Stricker

Object Recognition

`matchTemplate`	Compute proximity map for given template.
`CascadeClassifier`	Viola's Cascade of Boosted classifiers using Haar or LBP features. Suits for detecting faces, facial features and some other objects without diverse textures. See `facedetect.cpp`
`HOGDescriptor`	N. Dalal's object detector using Histogram-of-Oriented-Gradients (HOG) features. Suits for detecting people, cars and other objects with well-defined silhouettes. See `peopledetect.cpp`

Slide from D. Stricker

Thank you!