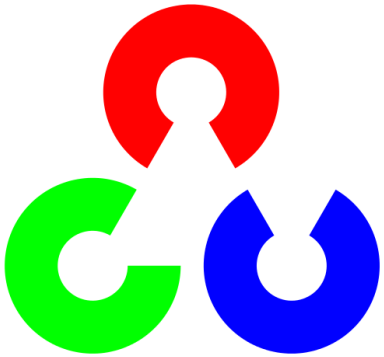
TELECOM SudParis

Institut Mines-Télécom

High Tech Imaging

IMA 4509 | Visual Content Analysis

OpenCV Tutorial



Yassine LEHIANI - Nicolas ROUGON

ARTEMIS Department

Yassine.Lehiani@telecom-sudparis.eu

Overview

1. Introduction
2. Modules
3. Basic structures
4. Basic routines
5. Advanced routines

Introduction

What is OpenCV?

- **OpenCV: Open Source Computer Vision library of programming functions**
  - Created in 1999 by Intel
  - Supported from 2008 by WillowGarage (a SME dedicated to hardware & open source software for personal robotics applications)

    www.willowgarage.com/pages/software/overview

    - WillowGarage also supports the Point Cloud Library (PCL)
- **Cross-platform**

  Windows | Linux | Android | Mac OS | iOS …
- **Free under BSD License**

  Commercial & non-commercial applications

What is OpenCV?

- **Written in C / C++**
  - Developers Wiki — **code.opencv.org**
  - Source code — **github.com/Itseez/opencv**
- **APIs available for a variety of programming languages**

  Python | Java | Matlab / Octave | CUDA | OpenCL
- **Online documentation** — **opencv.org/documentation.html**

  Reference | Tutorials | Books | Publications | QuickStart | Wiki
- **Current stable version is 3.1**

Installing OpenCV

- **Download page** **opencv.org/downloads.html**
  - Precompiled distributions available for standard OS
- **Installing & Building OpenCV on Windows**
  1. Download & Install CMake **cmake.org**
  2. Download & Unpack OpenCV archives
  3. Configure & Generate solution for a target IDE
     MS Visual | Eclipse | …
  4. Build static & dynamic libraries
  5. Add include & library paths to the project

Modules

Main packages

- **core** ▶ Base data structures & core routines
- **imgproc** ▶ Image processing routines
  - Linear / nonlinear image filtering
  - Geometric image transforms
  - Shape descriptors
  - Basic image operators
  - Histograms
  - Basic feature detection
- **video** ▶ Video analysis routines
  - Motion estimation
  - Motion segmentation
  - Background subtraction
  - Object tracking

Main packages

- **calib3D** ▶ Basic multiple-view geometry algorithms
  - Single / stereo camera calibration
  - Object pose estimation
  - Stereo correspondence
  - 3D reconstruction
- **features2D** ▶ 2D image features routines
  - Feature detectors
  - Descriptor extractors
  - Descriptor matchers
  - Object categorization
- **objdetect** ▶ object detection routines
  - Detection of objects and instances of predefined classes
    e.g. faces | eyes | mugs | people | cars | …

Main packages

- **highgui** ▶ High-level GUI & Media I/O
  - Video capturing
    incl. OpenNI-compatible depth sensors (Kinect | XtionPRO | ...)
  - Image & video codecs
  - Simple GUI capabilities
- **gpu** ▶ GPU-accelerated computer vision
- **ocl** ▶ OpenCL-accelerated computer vision

Main packages

- **Other helper modules**
  - **FLANN** ▶ Fast Library for Approximate Nearest Neighbors
    Clustering & search in multidimensional spaces
  - **ml** ▶ Machine learning
  - **cv2** ▶ OpenCV-Python bindings
  - **photo** ▶ Computational photography
  - **stitching** ▶ Image stitching
  - **superres** ▶ Image super resolution
  - **viz** ▶ 3D visualizer
  - …

Basic structures

Header files

- Programming applications with OpenCV requires to include header files (suffixed .hpp) depending on function calls

- Example

```
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/video/video.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/ml/ml.hpp"
#include "opencv2/highgui/highgui.hpp"
```

Namespace

- **All OpenCV classes and functions are in the cv namespace**
  - **'using namespace cv'** must be added after including header files

Example

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"

using namespace cv;

int main()
{
    Mat frame = imread( "landscape.jpg", 1 );
    imshow( "image", frame );
    waitKey( -1 );

return 0;
}
```

Namespace

- **All OpenCV classes and functions are in the cv namespace**
  - Alternatively, append the **'cv::'** specifier to every OpenCV classes, functions and data structures

Example

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"

int main()
{
    cv::Mat frame = cv::imread( "landscape.jpg", 1 );
    cv::imshow( "image", frame );
    cv::waitKey( -1 );

return 0;
}
```

class Mat

- **nD dense numerical single / multi-channel array class**

  An array object has a specified data type which defines

  - the # of allocated bits per element (pixel for an image)
  - the data format used for representing an element value

- **The primitive identifier has the following syntax**

  CV_<bit_depth><data_type>C<nb_channels>

  - bit_depth 8 | 16 | 32 | 6
  - data_type U | S | F
    unsigned char | signed short integer | float
  - nb_channels 1 | 2 | 3 | … | 512

class Mat

► Examples

- cv::Mat M1 (2, 3, CV_8UC1)
  (2x3) single channel array with 8-bit unsigned char data

- cv::Mat A (10, 10, CV_16SC3)
  (10x10) 3-channel array with 16-bit signed short integer data

- cv::Mat P (67, 53, CV_64FC(15))
  (67x53) 15-channel array with 64-bit float data

class Point_ , Point3_

- **Template classes for 2D/3D point specified by its Cartesian coordinates with various representation formats**

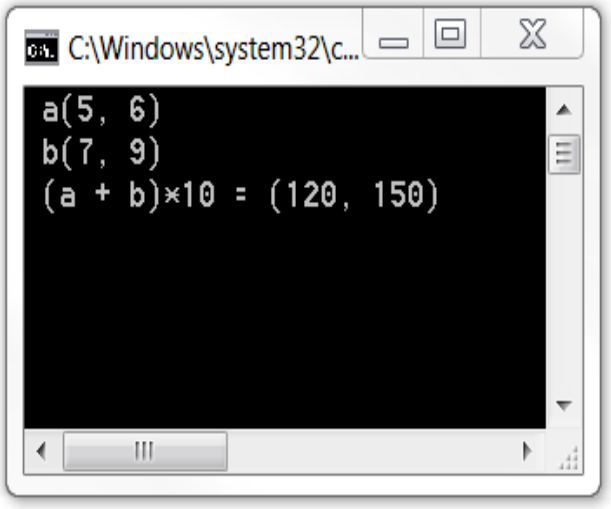
- **2D point**
  - **Point_<type>**
    type int | float | double
  - **Point2i** integer coordinates
  - **Point2f** float coordinates
  - **Point2d** double coordinates

- **3D point**
  - **Point3_<type>**
  - **Point3i** integer coordinates
  - **Point3f** float coordinates
  - **Point3d** double coordinates

class Point_ , Point_3

▶ Example

```
Point2f a(5, 6), b(7, 9);
cout << " a("<< a.x << ", " << a.y << ")" << endl;
cout << " b("<< b.x << ", " << b.y << ")" << endl;

Point pt = (a + b) * 10;
cout << " (a + b)*10 = ("<< pt.x << ", " << pt.y << ")" << endl;
```



class Vec

- **Template class for short numerical vector specified by its Cartesian coordinates with various representation formats**
  - **Vec<type, n>** type uchar | short | int | float | double
    n const int
  - **Vec2b | Vec3b | Vec4b** unsigned char 2/3/4D coordinates
  - **Vec2s | Vec3s | Vec4s** signed 2/3/4D coordinates
  - **Vec2i | Vec3i | Vec4i** integer 2/3/4D coordinates
  - **Vec2f | Vec3f | Vec4f | Vec5f | Vec6f**
    float 2/3/4/5/6D coordinates
  - **Vec2d | Vec3d | Vec4d | Vec5d | Vec6d**
    double 2/3/4/5/6D coordinates

Data type: class Vec

▶ Example

```
Vec2i v1( 1, 2 ), v2( 4, 5 );
cout << " v1("<< v1(0) << ", " << v1(1) << ")" << endl;
cout << " v2("<< v2(0) << ", " << v2(1) << ")" << endl;

Vec2i v = v2 - v1;
cout << " v2 - v1 = ("<< v(0) << ", " << v(1) << ")" << endl;
```

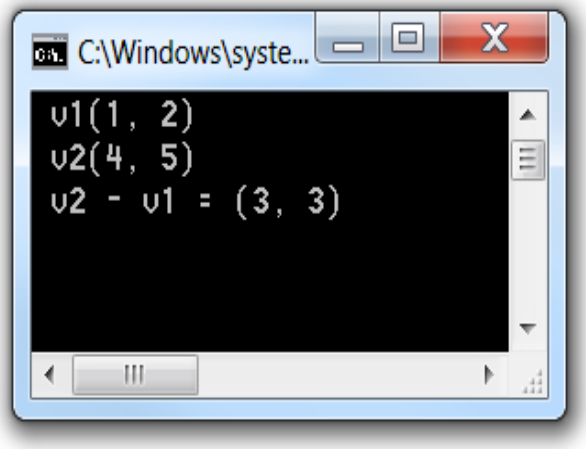


Image IOs

- **Loading an image:** **imread()**

  > Mat imread(const string& filename, int flags)

  - Returns a single (grayscale) / multiple (color) channel array
    Supported formats: BMP, JPEG, JPEG-2000, PBM/PGM/PPM, PNG, TIFF

    – filename — image file name

    – flags — CV_LOAD_IMAGE_ANYDEPTH
    returns 16/32-bit image when the input has corresponding depth, otherwise convert it to 8-bit

    CV_LOAD_IMAGE_COLOR
    convert image to color

    CV_LOAD_IMAGE_GRAYSCALE
    convert image to grayscale

Image IOs

- **Writing an image:** **imwrite()**

  > bool imwrite(const string& filename, InputArray image, const vector<int>& parameters)

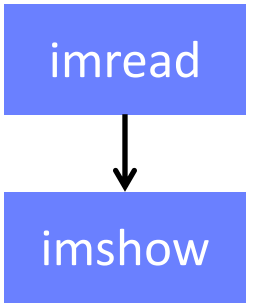
  - filename — image file name
  - image — image array to be saved
    InputArray is a proxy class for passing read-only arrays
    It can be constructed from Mat
  - parameters — format-specific parameters, encoded as pairs *parameterID_n, parameterValue_n*

Image display

- **Displaying an image:** **imshow()**
  - > void imshow(const string& windowName, InputArray image)
    - windowName target window name
    - image image array to be displayed

Basic visualization pipeline

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main()
{
    const char *windname = "image";
    // create a window
    namedWindow( windname, CV_WINDOW_AUTOSIZE );

    // load the image
    Mat frame = imread( "landscape.jpg", IMREAD_UNCHANGED );

    // check if the image is valid
    if( frame.empty() )
        {
            cout << "Image cannot be loaded..." << endl;
            return -1;
        }

    // display the image
    imshow( windname, frame );

    // wait for pressed key
    waitKey( 0 );

    // destroy the window
    destroyWindow( windname );

return 0;
```



Video IOs

- **Video from file / device** — **Class VideoCapture**
  - Open file / device — **open()**
  > bool open(string &fileName)
  > bool open(int device)
  - Check for initialization — **isOpened()**
  > bool isOpened()
  - Get / Set device property — **get() / set()**
  > double get(int propertyID)
  > bool set(int propertyID, double value)
  - Close file / device — **release()**
  > void release()

Video IOs

- **Video from file / device** — **Class VideoCapture**
  - Grab next frame — **grab()**

  > bool grab()

  - Decode grabbed frame — **retrieve()**

  > bool retrieve(Mat &image, int channel = 0)

  - Grab & decode next frame — **read()** | *operator* **>>**

  > bool read(Mat &image)
  >
  > *operator* >> (Mat &image)

Video IOs

- **Save video** — **Class VideoWriter**
  - (Re-)initialize video writer — **open()**

> bool open(string &fileName, int fourcc, double fps,
> Size framesize, bool isColor = true)

    - fourcc — codec 4-character code > see **fourcc.org**
    - fps — frame rate

  - Check for initialization — **isOpened()**

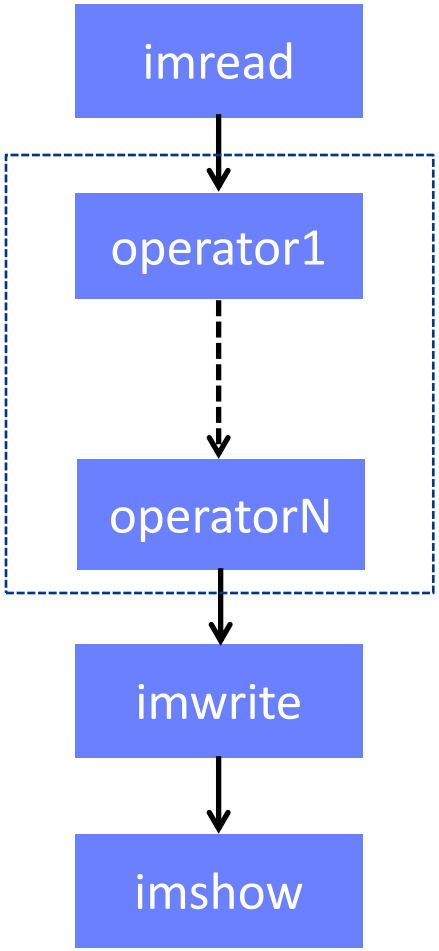
> bool isOpened()

  - Write next frame — **write()** | *operator* **<<**

> void write(Mat &image)
> *operator* << (Mat &image)

Basic routines

Basic processing pipeline



- IO
- Processing
- IO
- Visualization

Image processing

Image linear filtering

- 2D image convolution: **filter2D()**

> void filter2D(InputArray src, OutputArray dst, *parameters*)

Image processing

Image denoising

- Average filtering: **blur () / boxFilter()**

> void blur(InputArray src, OutputArray dst, *parameters*)

> void boxFilter(InputArray src, OutputArray dst, *parameters*)

- Gaussian filtering: **gaussianBlur()**

> void gaussianBlur(InputArray src, OutputArray dst, *parameters*)

- Bilateral filtering: **bilateral Filter()**

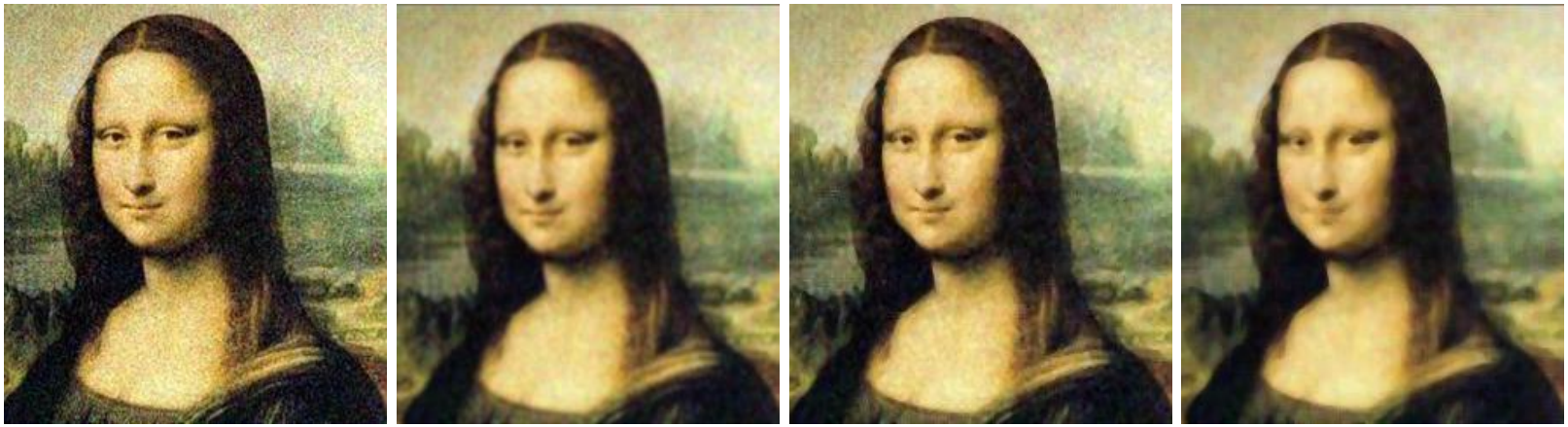
> void bilateralFilter(InputArray src, OutputArray dst, *parameters*)

- Median filtering: **medianBlur()**

> void medianBlur(InputArray src, OutputArray dst, *parameters*)

Image processing

- **Image denoising**
  - > void gaussianBlur(InputArray src, OutputArray out, *parameters*)
  - > void bilateralFilter(InputArray src, OutputArray dst, *parameters*)
  - > void medianBlur(InputArray src, OutputArray dst, *parameters*)



original Gaussian bilateral median

Image processing

Mathematical morphology

- Erosion / Dilation **erode () / dilate()**

> void erode(InputArray src, OutputArray dst, *parameters*)
> void dilate(InputArray in, OutputArray out, *parameters*)

- Higher-order operators **morphologyEx()**

> void morphologyEx(InputArray src, OutputArray dst, int op, *parameters*)

– op	MORPH_OPEN	opening
	MORPH_CLOSE	closing
	MORPH_GRADIENT	morphological gradient
	MORPH_TOPHAT	top hat
	MORPH_BLACKHAT	negative top hat

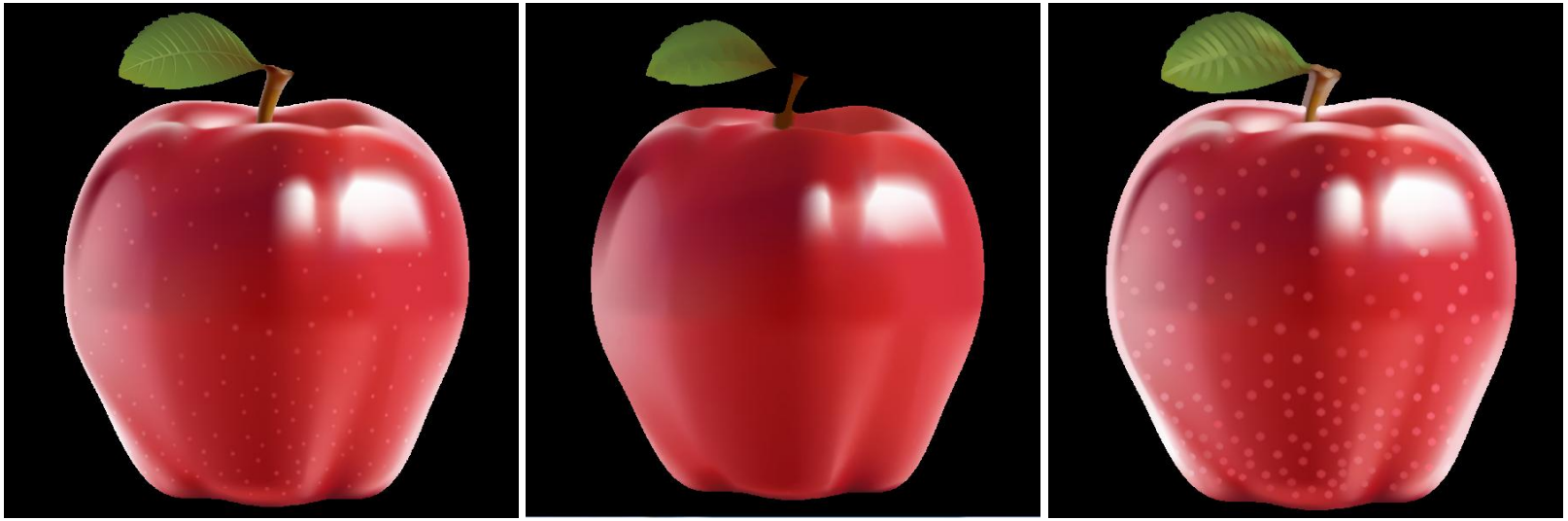
Image processing

Mathematical morphology

- Erosion / Dilation **erode () / dilate()**

> void erode(InputArray src, OutputArray dst, *parameters*)

> void dilate(InputArray in, OutputArray out, *parameters*)



original erosion dilation

Image processing

Edge detection

- Sobel 2D gradient filter: **sobel()**

> void sobel(InputArray src, OutputArray dst, *parameters*)

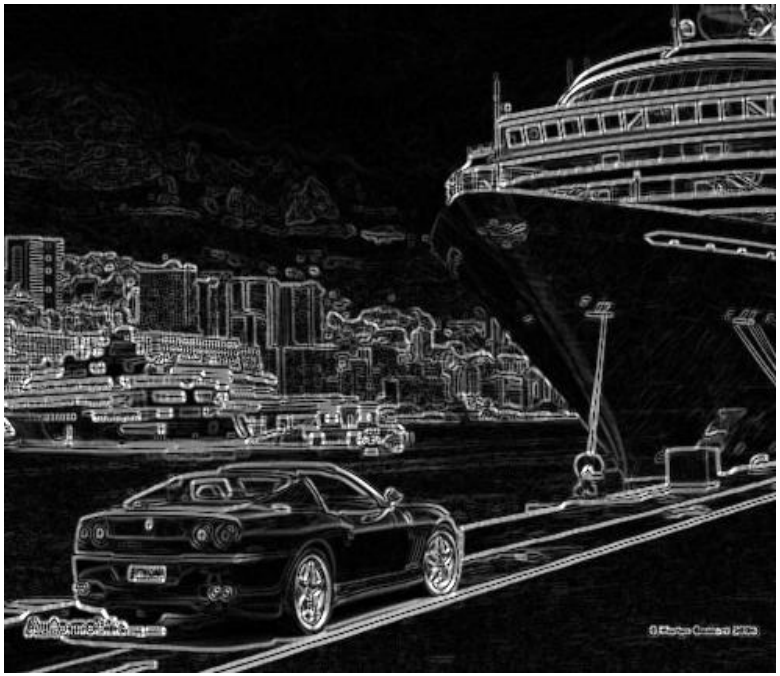


Image processing

Edge detection

- 2D Laplacian filter: **laplacian()**

> void laplacian(InputArray src, OutputArray dst, *parameters*)

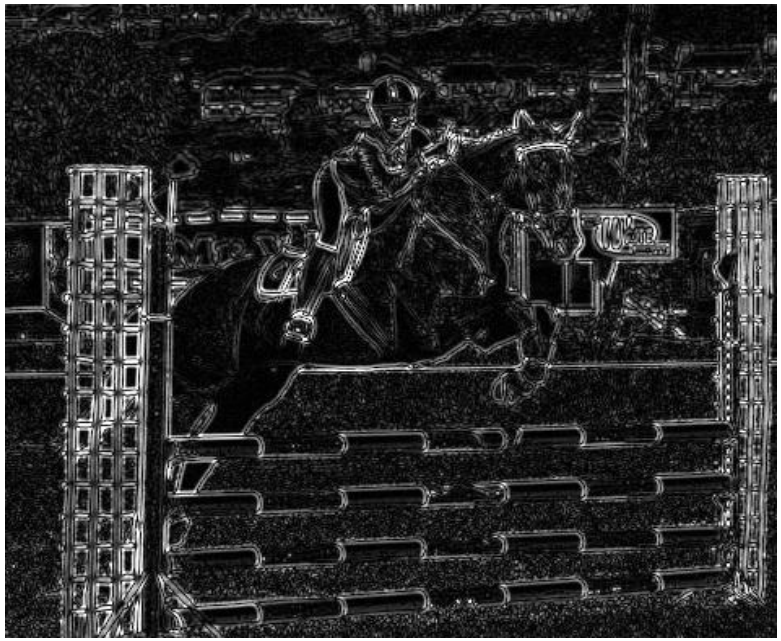


Image processing

Edge detection

- Canny 2D edge detector: **Canny()**
  Canny gradient filter + hysteresis threshold

> void Canny(InputArray src, OutputArray dst, *parameters*)

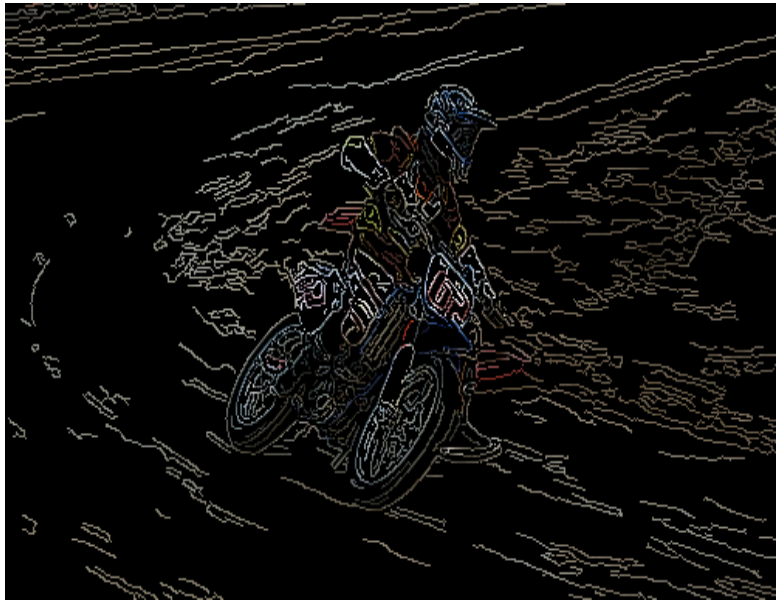


Image processing

Edge detection

- Chain & organize edge points into contour hierarchies: **findContours()**

> void findContours(InputOutputArray src, OutputArrayOfArrays dst, *parameters*)

Canny → findContours → drawContours

Image processing

Thresholding

- Fixed-level threshold: **threshold()**

> double threshold(InputArray src, OutputArray dst,
> double thresh, *parameters*)

- Adaptive threshold: **adaptiveThreshold()**

> void adaptiveThreshold(InputArray src, Output Array dst, …,
> int method, *parameters*)

– method ADAPTIVE_THRESH_MEAN_C local mean
ADAPTIVE_THRESH_GAUSSIAN_C local Gaussian mean

Image processing

Histograms

- Histogram calculation: **calcHist()**

> void calcHist(Mat *images, …, OutputArray H, *parameters*)

- Histogram similarity: **compareHist()**

> void compareHist(InputArray H1, InputArray H2, int method)

– method

CV_COMP_CORREL	correlation
CV_COMP_CHISQR	Chi-square
CV_COMP_INTERSECT	intersection
CV_COMP_BHATTACHARYYA	Bhattacharyya distance

Image processing

Histograms

- Histogram equalization: **equalizeHist()**

> void equalizeHist(InputArray src, OutputArray dst)

Image processing

Object detection

- Template matching: **matchTemplate()**

> void matchTemplate(InputArray src, InputArray template, OutputArray similarity_map, int method)

– method

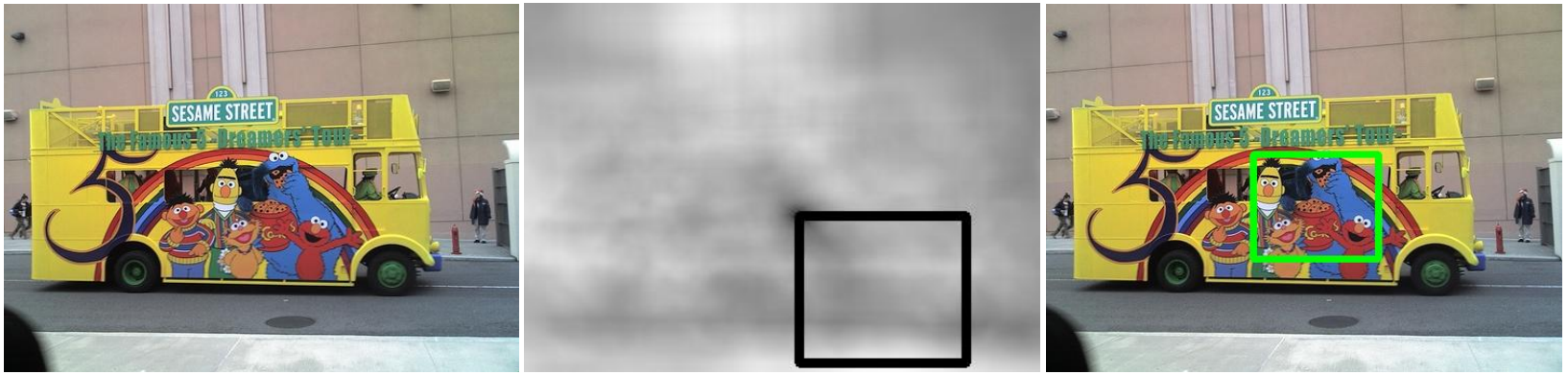
CV_TM_SQDIFF	SSD
CV_TM_SQDIFF_NORMED	normalized SSD
CV_TM_CCORR	correlation
CV_TM_CCORR_NORMED	normalized correlation
CV_TM_CCOEFF	covariance
CV_TM_CCOEFF_NORMED	correlation coefficient

Image processing

Object detection

- Template matching: **matchTemplate()**

> void matchTemplate(InputArray src, InputArray template,
OutputArray similarity_map, int method)



template

matchTemplate → minMaxLoc

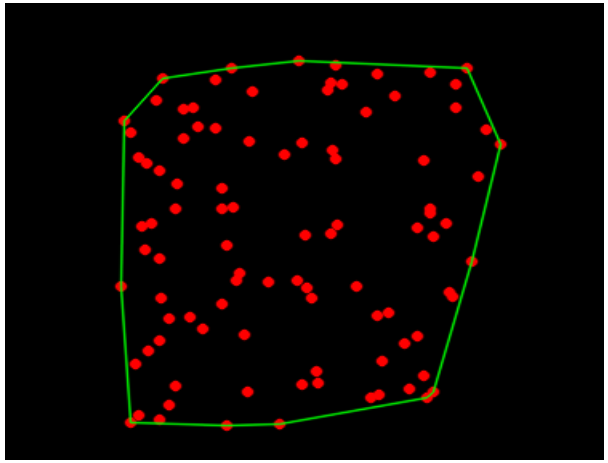
Image processing

- **Structural analysis**
  - Bounding rectangle: **boundingRect()**

  > Rect boundingRect(InputArray pts)

  - Convex hull: **convexHull()**

  > void convexHull(InputArray pts, OutputArray dst, *parameters*)



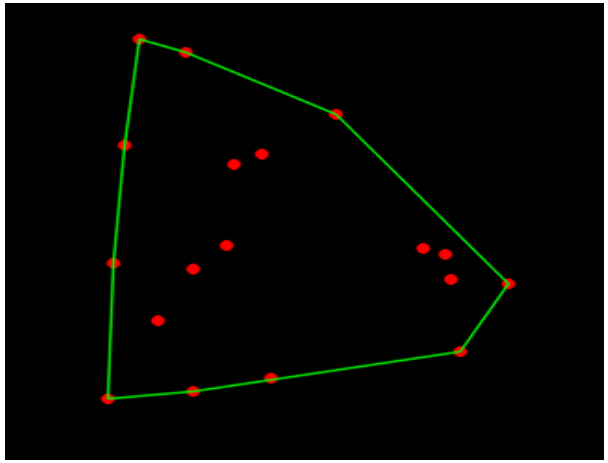


Image processing

Corner detection

- Harris detector: **cornerHarris()**

> void cornerHarris(InputArray src, OutputArray dst, *parameters*)

- Shi-Tomasi detector: **goodFeaturesToTrack()**

> void goodFeaturesToTrack(InputArray src, OutputArray dst, *parameters*)

Image processing

- **Corner detection**
  - > void cornerHarris(InputArray src, OutputArray dst, *parameters*)
  - > void goodFeaturesToTrack(InputArray src, OutputArray dst, *parameters*)

Harris

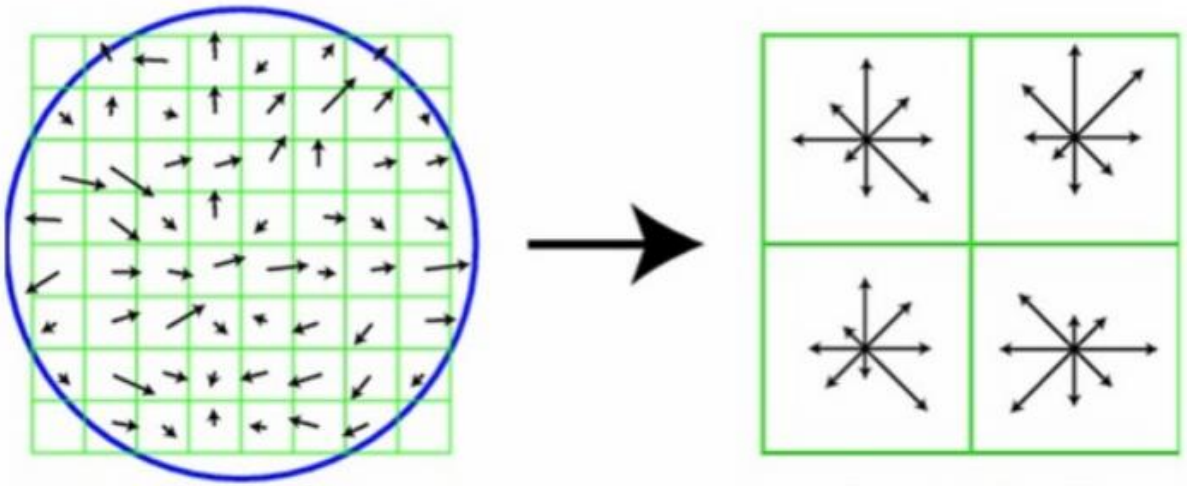
Shi-Tomasi

Feature2D

- **Feature point descriptors**

  Compute a set of parameters characterizing salient image points based their neighborhood, with various (*e.g.* scale, rotation, contrast) invariance properties

  Classes: ORB | BRISK | FREAK | FAST | SURF | SIFT

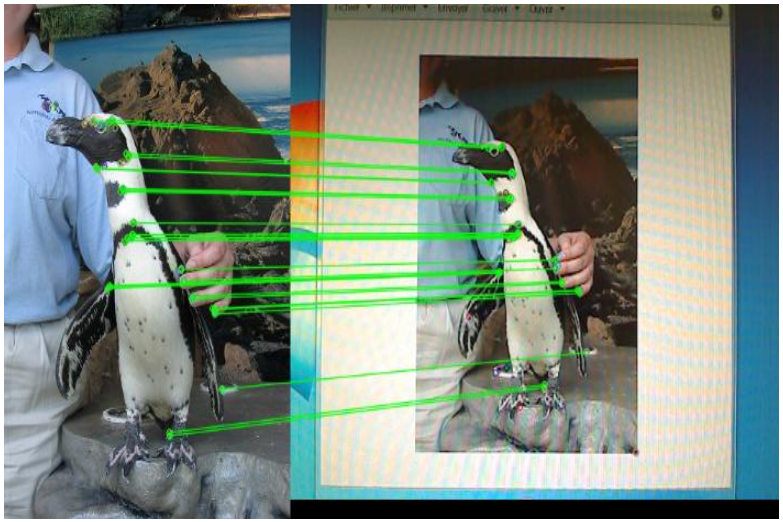


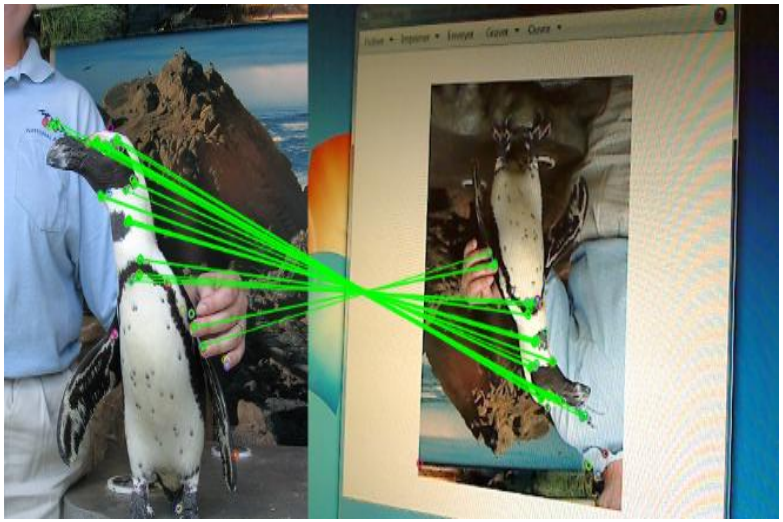
Feature2D

- **Matching descriptors**

  Match keypoint descriptor sets. Matched descriptors are represented as nD vectors comprising pairwise descriptors and best match indices

  Classes **DescriptorMatcher::(match | knnMatch | radiusMatch)**
  BFMatcher | FlannBasedMatcher





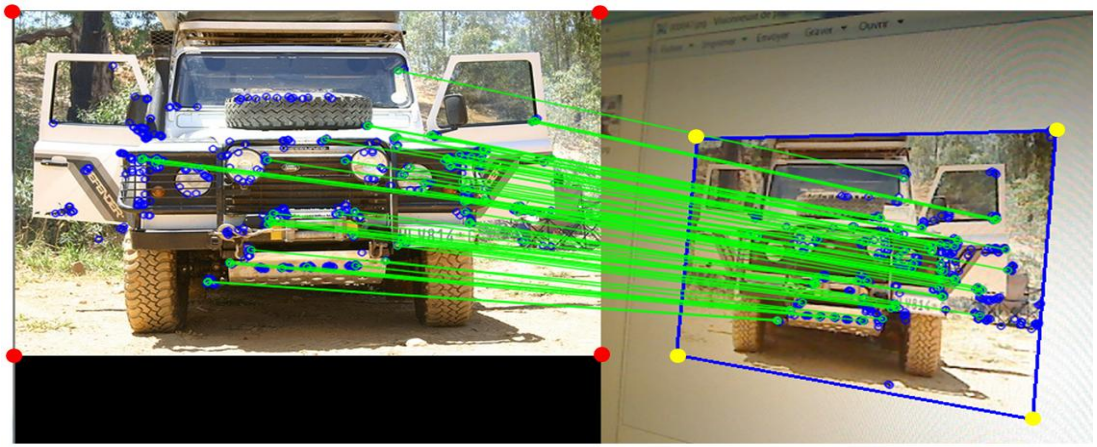
Calibration & 3D reconstruction

Matching 2D point sets

- Perspective transform estimation: **findHomography()**

> void findHomography(InputArray srcPts, InputArray dstPts, int method, *parameters*)

– method CV_RANSAC RANSAC
CV_LMEDS Least-Median



findHomography

↓

warpPerpective

Advanced routines

Video analysis

- **Dense optical flow estimation**
  - Gunnar-Farneback estimator: **calcOpticalFlowFarneback()**

> void calcOpticalFlowFarneback(InputArray prev,
> InputArray next,
> InputOutputArray flow,
> *parameters*)

  - SimpleFlow estimator: **calcOpticalFlow SF()**

> void calcOpticalFlowSF(Mat &prev, Mat &next,
> Mat &flow, *parameters*)

Video analysis

- **Sparse optical flow estimation**
  - Lucas-Kanade tracker: **calcOpticalFlowPyrLK()**

> void calcOpticalFlowPyrLK(InputArray prev,
> InputArray next,
> InputArray prevPts,
> InputOutpuArray nextPts,
> *parameters*)

Video analysis

Kalman filtering — Class KalmanFilter

Kalman filters estimate system parameters from measurements. Predicted states are updated to yield refined states

- KF initialization: **init()**

> void init(*parameters*)

- KF state prediction: **predict()**

> Mat& predict(Mat &control)

- KF state update: **correct()**

> Mat& correct(Mat &measurement)

Video analysis

- **Kalman filtering** **Class KalmanFilter**

  Kalman filters estimate system parameters from measurements. Predicted states are updated to yield refined states

Video analysis

- **Object tracking via histogram back-projection**
  - Histogram back-projection: **CalcBackProject()**

  > void CalcBackProject(Mat *images, …, InputArray H, OutputArray bp, *parameters*)

  - MeanShift tracker: **meanShift()**

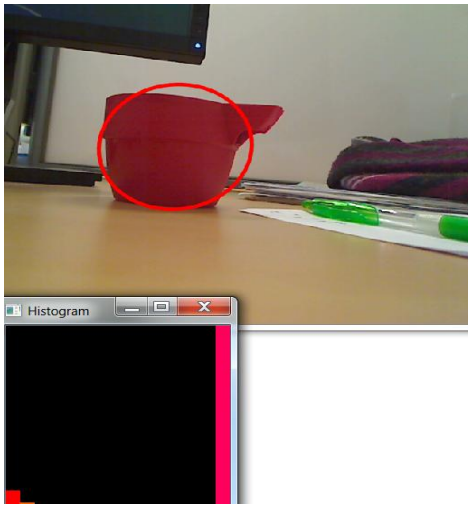
  > int meanShift(InputArray bp, Rect& win, *parameters*)

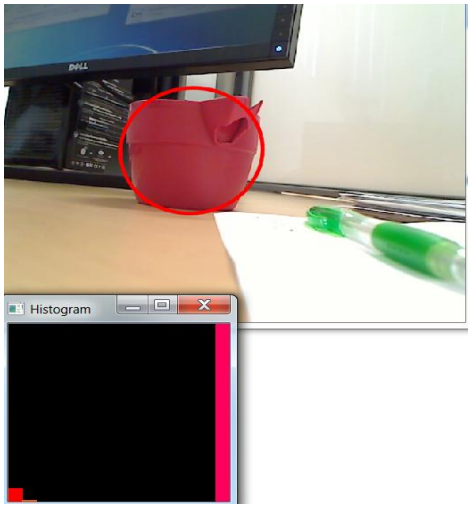
  - CAMSHIFT tracker: **CamShift()**

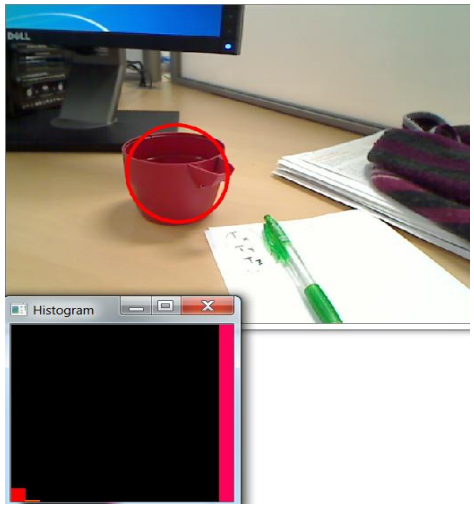
  > RotatedRect CamShift(InputArray bp, Rect& win, *parameters*)

Video analysis

- **Object tracking via histogram back-projection**
  - > void CalcBackProject(Mat *images, …, InputArray H, OutputArray bp, *parameters*)
  - > int meanShift(InputArray bp, Rect& win, *parameters*)
  - > RotatedRect CamShift(InputArray bp, Rect& win, *parameters*)







Calibration & 3D reconstruction

- **Camera calibration** **calibrateCamera()**

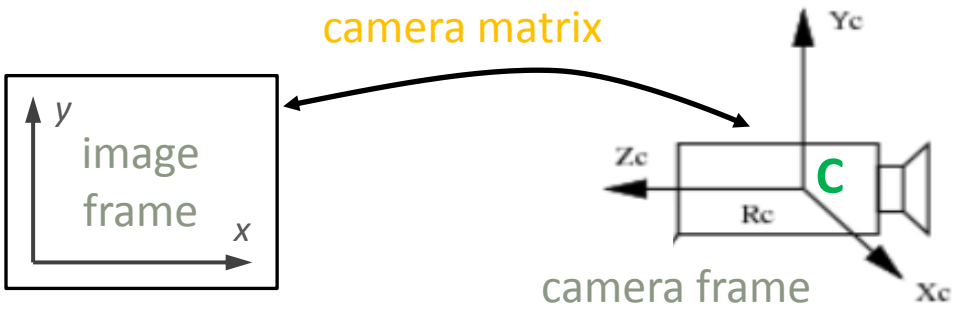
  Estimate camera intrinsic & extrinsic parameters from several views of a calibration pattern (*e.g.* chessboard)

  > double calibrateCamera(InputArrayOfArrays objectPts,
  > InputArrayOfArrays imagePts, …,
  > InputOutputArray cameraMatrix,…)

scaling C

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

camera matrix



findChessboardCorners

↓

calibrateCamera

Calibration & 3D reconstruction

- **Camera calibration** **calibrateCamera()**

  Estimate camera intrinsic & extrinsic parameters from several views of a calibration pattern (*e.g.* chessboard)

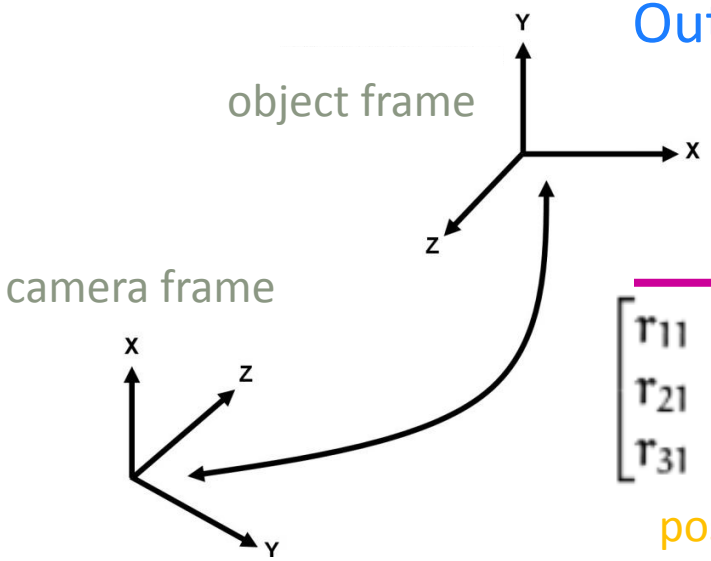
  > double calibrateCamera(InputArrayOfArrays objectPts,
  > InputArrayOfArrays imagePts, …,
  > InputOutputArray cameraMatrix,
  > InputOutputArray distorsionCoefficients,
  > OutputArrayOfArrays R,
  > OutputArrayOfArrays T,
  > *parameters*)

Calibration & 3D reconstruction

- **3D pose estimation** **solvePnP()**

  Estimate object pose from 3D-2D point correspondences

  > bool solvePnP(InputArray objectPts, InputArray imagePts,
  > InputArray cameraMatrix,
  > InputArray distorsionCoefficients,
  > OutputArray **R**, OutputArray **T**, *parameters*)



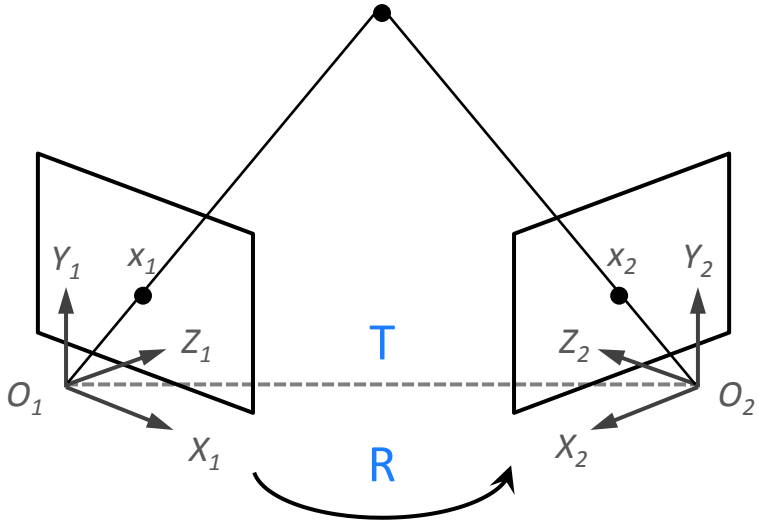
$$\underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{R} \underbrace{\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}}_{T}$$

pose transform

Calibration & 3D reconstruction

- **Stereo camera calibration** **stereoCalibrate()**

> double stereoCalibrate(InputArrayOfArrays objectPts,
InputArrayOfArrays imagePts1,
InputArrayOfArrays imagePts2,
InputOuputArray cameraMatrix1,
InputOutputArray distorsionCoeffs1,
InputOuputArray cameraMatrix2,
InputOutputArray distorsionCoeffs2,
OutputArray R, OutputArray T,
OutputArray **E**, OutputArray **F**,
parameters)



Fundamental matrix

$x_1^T \mathbf{F} x_2 = 0$

Essential matrix

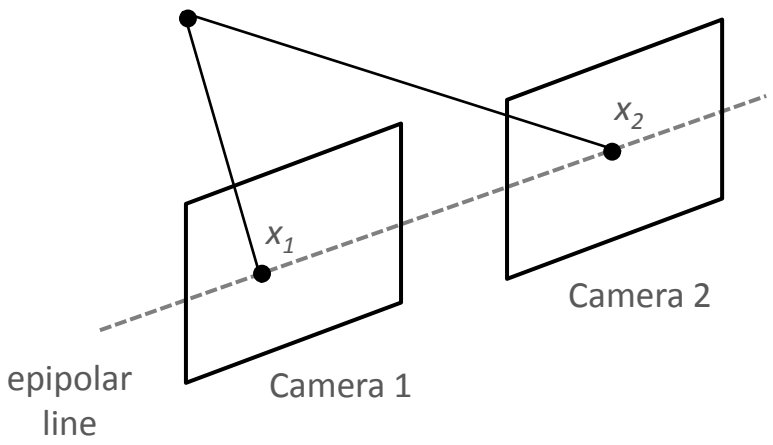
$\mathbf{E} = R\,[T]_x$

Calibration & 3D reconstruction

- **Stereo reconstruction** **triangulatePoints()**

  3D reconstruction from corresponding points in a stereo pair using camera projection matrices

  > void triangulatePoints(InputArray projectionMatrix1,
  > InputArray projectionMatrix2,
  > InputArray imagePts1,
  > InputArray imagePts2,
  > OutputArray Points3D)



Machine learning

- **Support Vector Machines** **Class CvSVM**

  SVM classifiers are supervised learning models used to separate linear and nonlinear data

  - SVM training: **train()**

  > bool train(Mat &trainData, Mat &responses, *parameters*)

  - Optimal SVM training: **train_auto()**

  > bool train_auto(Mat &trainData, Mat &responses, *parameters*)

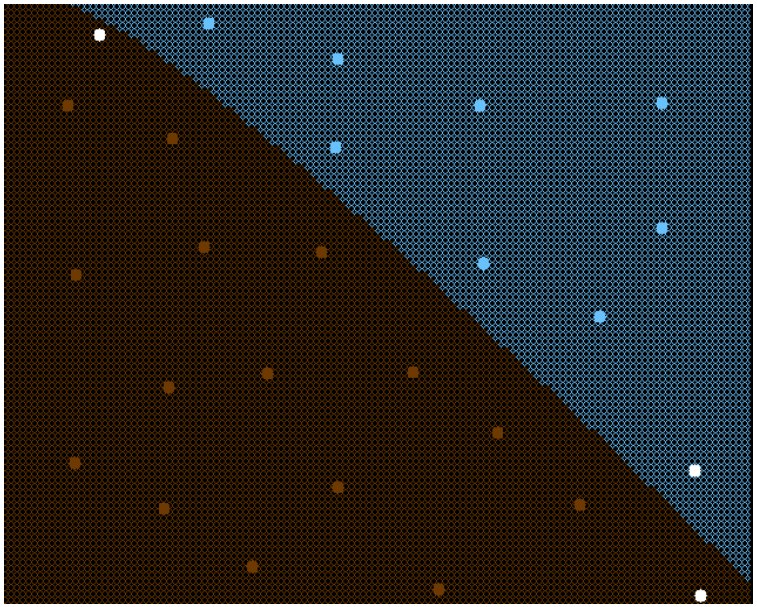
  - SVM-based prediction: **predict()**

  > float predict(const Mat &sample, *parameters*)

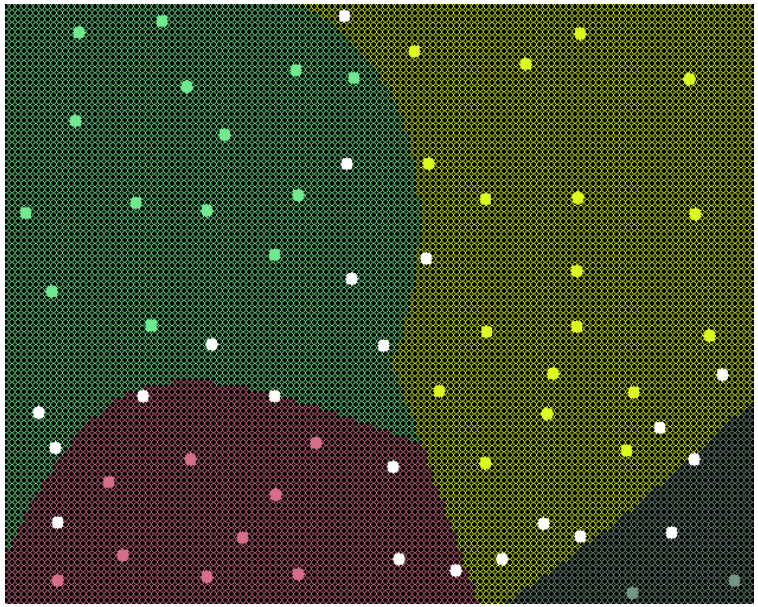
Machine learning

- **Support Vector Machines** **Class CvSVM**

  SVM classifiers are supervised learning models used to separate linear and nonlinear data



linear classification



nonlinear classification

Machine learning

- **Multi-Layer Perceptrons** **Class CvANN_MLP**

  MLP consist of connected input/hidden/output neuron layers. Neuron outputs are function of linear combinations of inputs with weights assigned by training

  - Create MLP: **create()**

  > void create(Mat &layerSizes, *parameters*)

  - MLP training / update: **train()**

  > int train(Mat &input, Mat &output, *parameters*)

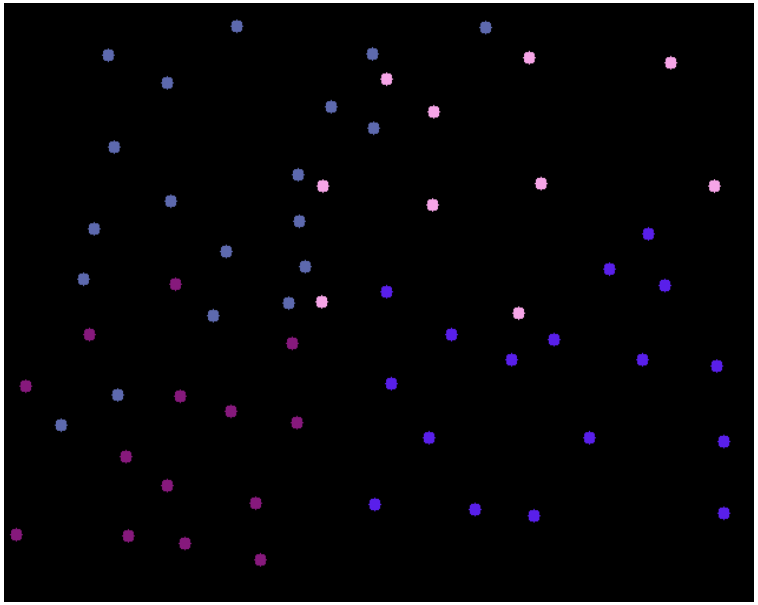
  - MLP-based prediction: **predict()**

  > float predict(const Mat &inputs, const Mat &outputs)

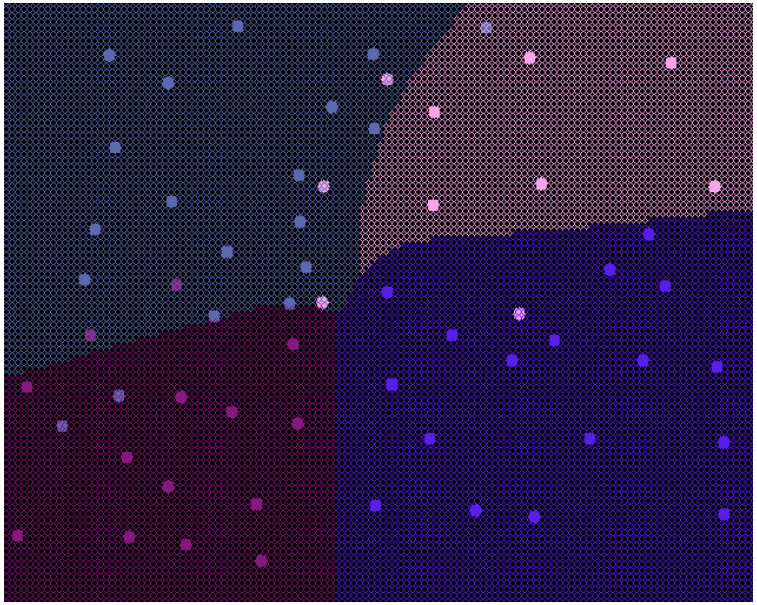
Machine learning

- **Multi-Layer Perceptrons** **Class CvANN_MLP**

  MLP consist of connected input/hidden/output neuron layers. Neuron outputs are function of linear combinations of inputs with weights assigned by training



data



MLP-based classification

Machine learning

- **K-Nearest Neighbors** **Class CvKNearest**

  kNN classification / regression predicts new samples from the *k*-nearest neighbors samples of an indexed training set

  - KNN training: **train()**

  > int train(Mat &trainData, Mat &responses, *parameters*)

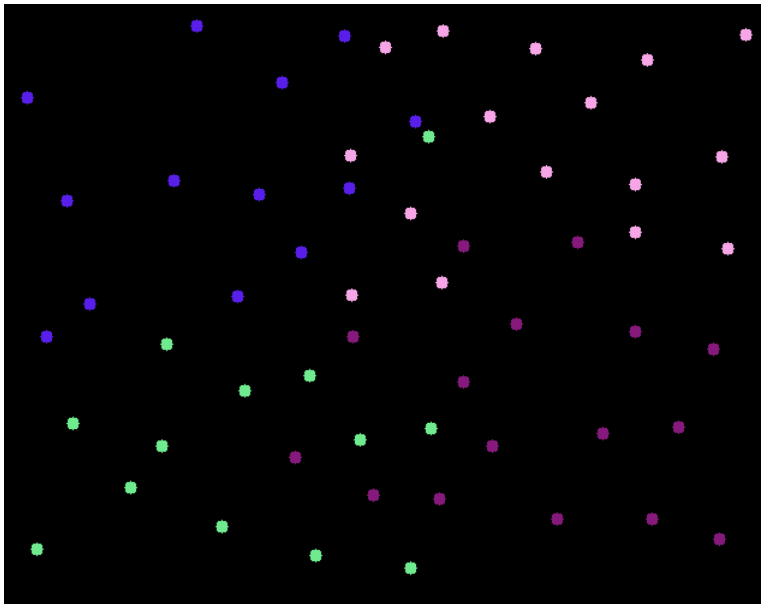
  - KNN-based prediction: **find_nearest()**

  > float find_nearest(Mat &inputs, int k, Mat &results, *parameters*)

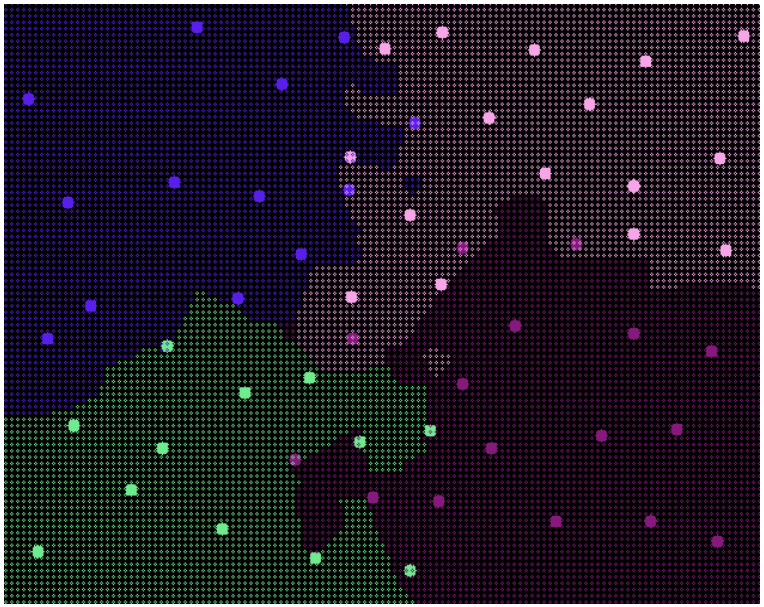
Machine learning

- **K-Nearest Neighbors** **Class CvKNearest**

  kNN classification / regression predicts new samples from the *k*-nearest neighbors samples of an indexed training set



data



kNN-based classification

Machine learning

- **Bayesian classification using Gaussian Mixture Models**

  Class CvNormalBayesClassifier

  - GMM Bayesian estimation: **train()**

  > bool train(Mat &trainData, Mat &responses, *parameters*)

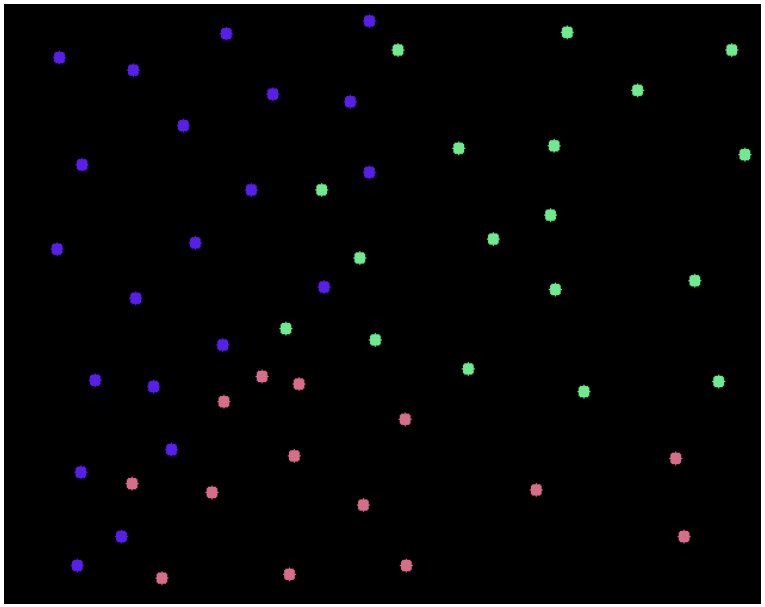
  - GMM-based prediction: **predict()**

  > float predict(Mat &inputs, Mat &results)

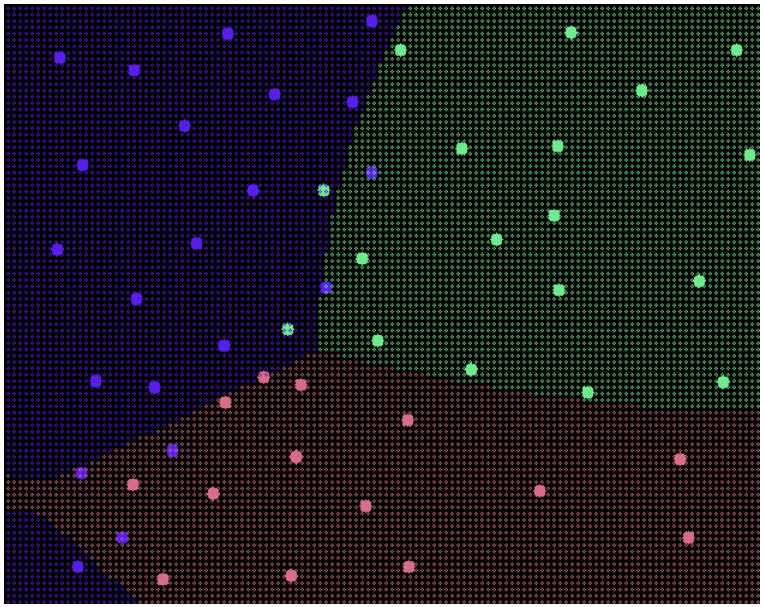
Machine learning

- **Bayesian classification using Gaussian Mixture Models**

  Class CvNormalBayesClassifier



data



GMM-based Bayesian classification

Graphic User Interface

Window routines

- Creation: **namedWindow()**

> void namedWindow(string &winName, int flags)

  - flags
    - WINDOW_NORMAL — free resize by user
    - WINDOW_AUTORESIZE — fit to content [default]
    - WINDOW_OPENGL — OpenGL support

- Destruction: **destroyWindow()** **destroyAllWindows()**

> void destroyWindow(string &winName)

> void destroyAllWindows()

Graphic User Interface

- **Window routines**
  - Move: **moveWindow ()**

> void moveWindow(string &winName, int x, int y)

  - Resize: **resizeWindow()**

> void resizeWindow(string &winName, int width, int height)

  - Update content: **updateWindow()**

> void updateWindow(string &winName)

Graphic User Interface

Window event handling

- Set mouse events handler: **setMouseCallback()**

> void setMouseCallback(string &winName,
>
> MouseCallback onMouse,
>
> void *userdata)

  - onMouse mouse events handling routine with prototype

    void onMouse(int event, int x, int y, int flags, void *userdata)

- Wait for & get pressed key: **waitKey ()**

> int waitKey (int delay = 0)

Graphic User Interface

Slider routines

- Create a slider widget: **createTrackbar()**

```
> void createTrackbar(string &trackbarName,
                      string &parentWindowName,
                      int *value, int maxValue,
                      TrackbarCallback onChange,
                      void *userdata)
```

  - onChange slider change handling routine with prototype
    void onChange(int value, void *userdata)

Slider minimum value is always 0

Graphic User Interface

Slider routines

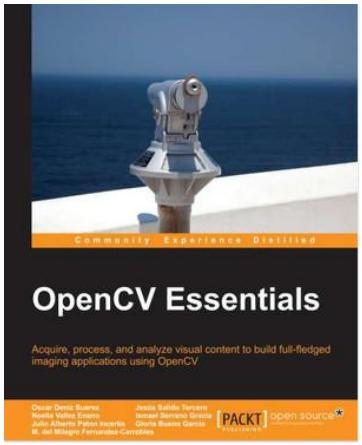
- Get slider value: **getTrackbarPos()**

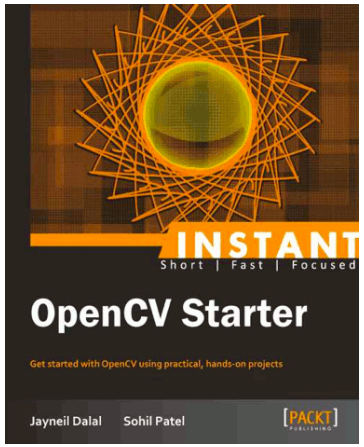
> int getTrackbarPos(string &trackbarName, string &winName)

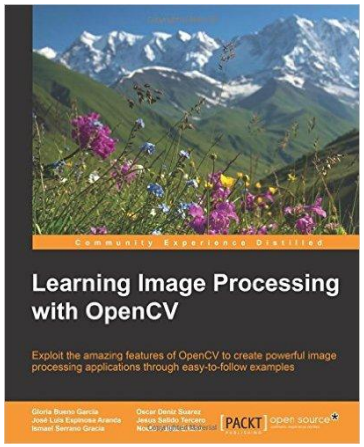
- Set slider value: **setTrackbarPos()**

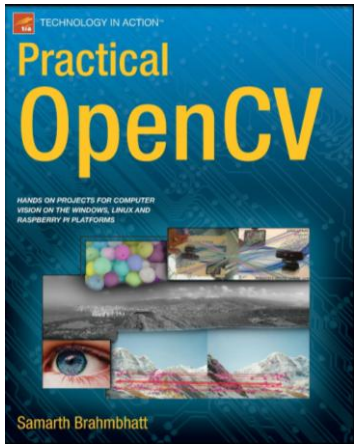
> void setTrackbarPos(string &trackbarName, string &winName, int value)

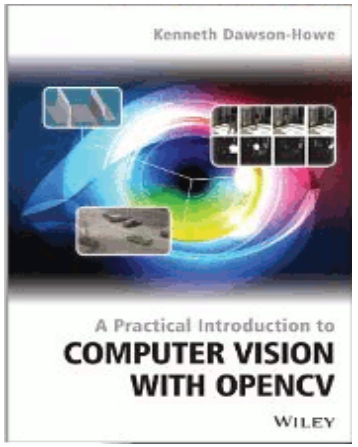
Bibliography

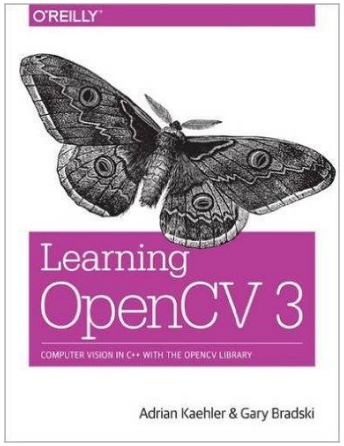


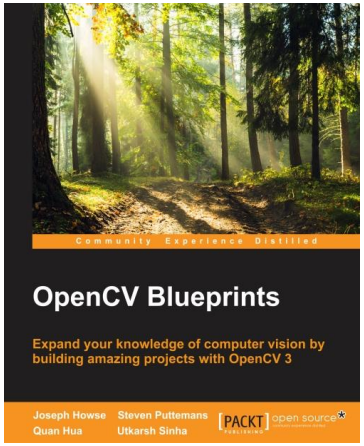






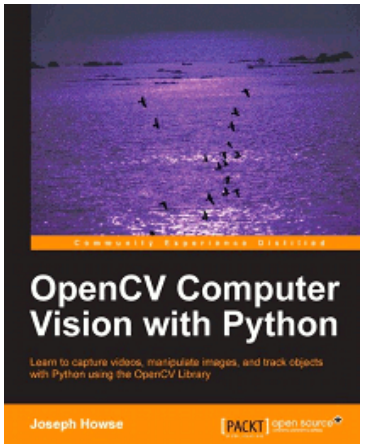


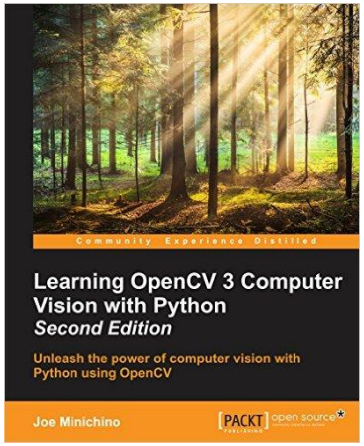


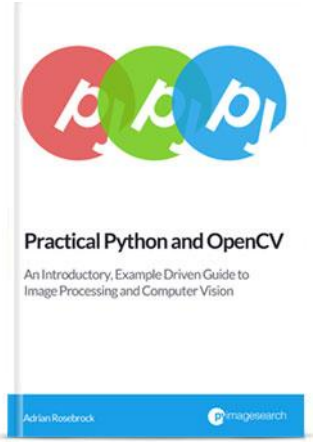


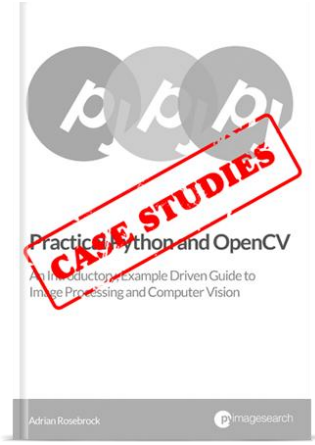
Bibliography

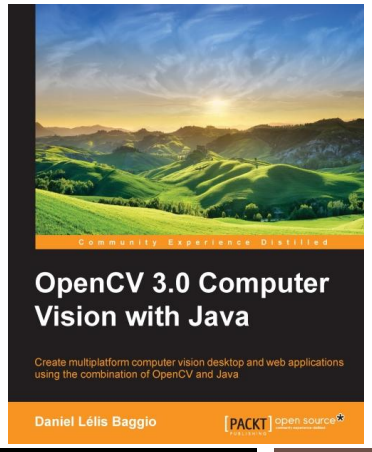
- **Python | Java | .NET APIs**

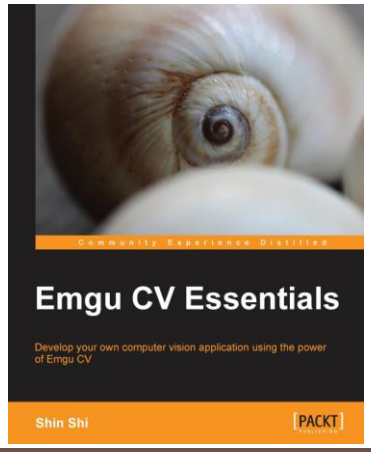








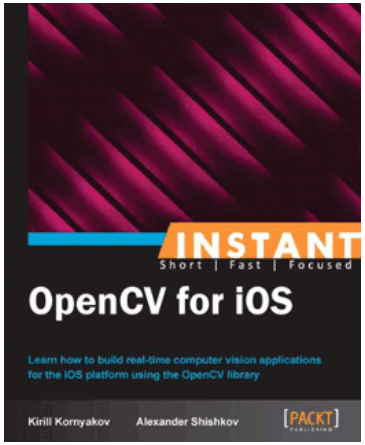


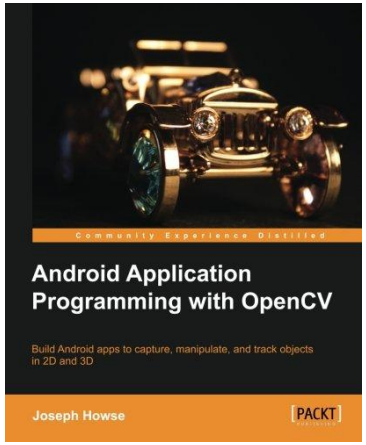


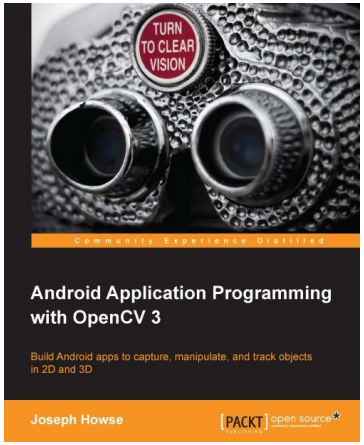
Bibliography

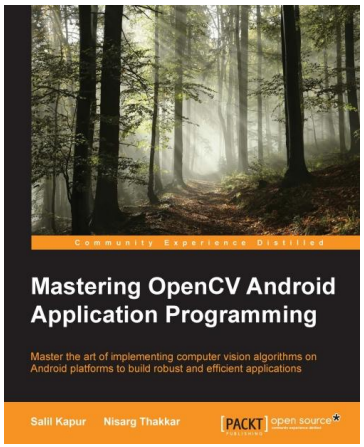
- **Mobile & Game APIs**

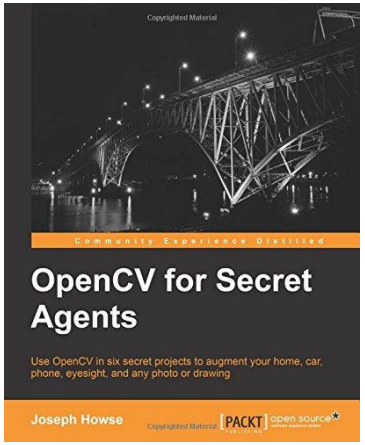
  Android | iOS | Raspberry Pi | Unity











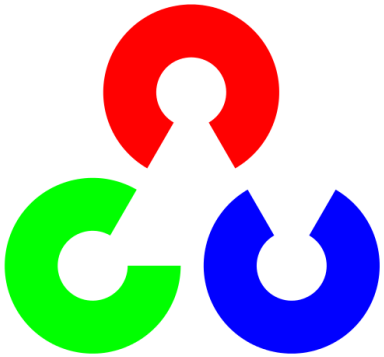
TELECOM SudParis

Institut Mines-Télécom

High Tech Imaging

IMA 4509 | Visual Content Analysis

OpenCV Tutorial



Yassine LEHIANI - Nicolas ROUGON

ARTEMIS Department

Yassine.Lehiani@telecom-sudparis.eu