Introduction to OpenCV

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

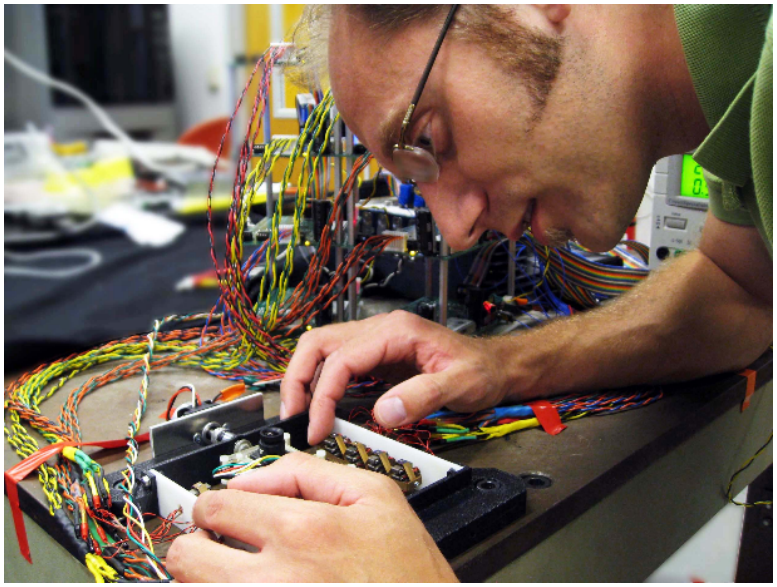
SAPIENZA
UNIVERSITÀ DI ROMA

Roberto Capobianco

Thanks to Alberto Pretto, Domenico Bloisi, Gary Bradski, James Hays and Stefano Soatto for some of the slides!

The sense of vision

- Main sense in humans
- Plays a fundamental role in most living organisms
- **Generates a representation of the surrounding world**



The human visual system measures the **intensity of light incident the retinas** and processes the measurements in the brain to produce an estimate of the 3D layout of the scene, to recognize objects, etc.

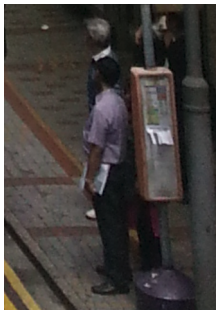
[Pictures: www.getholistichealth.com and gatech.edu]

Computer vision

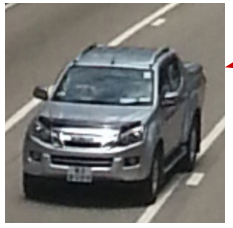
Enable a computer to understand an image or a video

- Place recognition
- 3D reconstruction
- Navigation
- Localization
- People detection
- Object recognition
- Action recognition
- ...

Two people, they are waiting the bus, ...

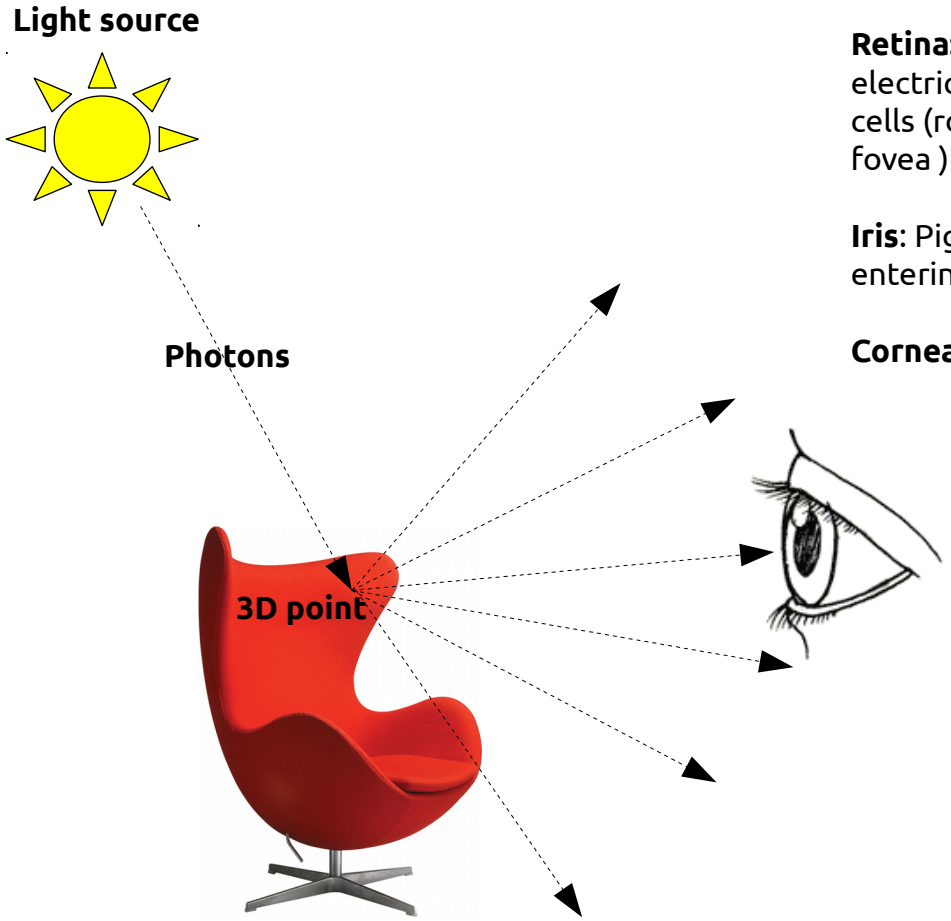


Car, model XXX, it is moving, ...



Car plate number YYY

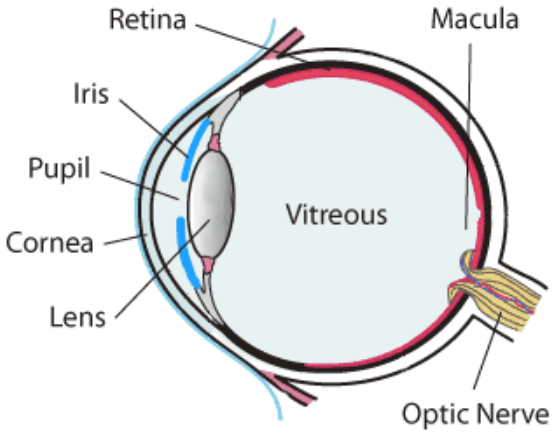
Capturing Light - human eye



Retina: Part of the eye that converts images into electrical impulses, i.e. it includes the photoreceptor cells (rods and cones, the latter densely packed in the fovea)

Iris: Pigmented tissue that controls amount of light entering eye by varying size of the **pupil**

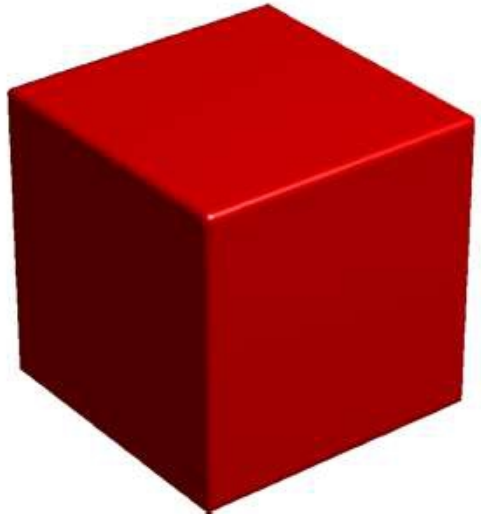
Cornea + **Lens**: natural lens of eye



[Source: heightseyecare.com]

Surface reflectance

Lambertian reflectance: Computer vision algorithms often assume that the color and brightness intensity of a point on a surface does not change with the vantage point

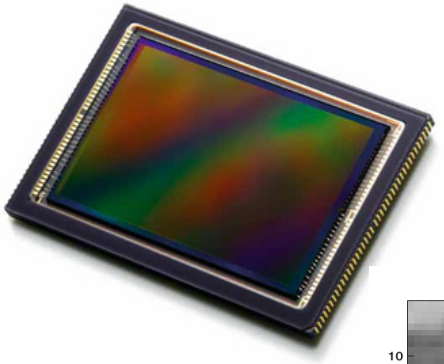


This is **not** true in general! But it is very hard to deal with non-lambertian reflectance...

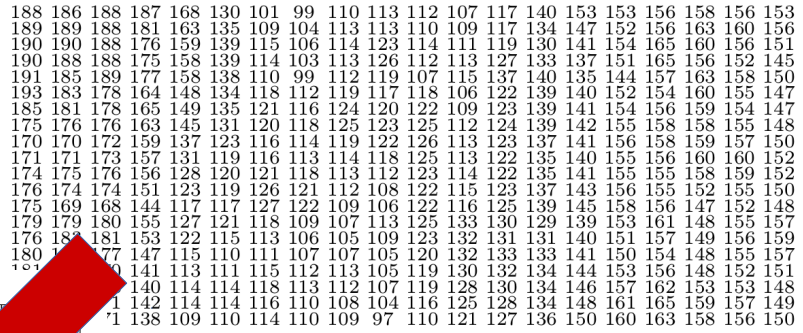
Capturing Light – digital cameras

A continuous scene is framed by a discrete array

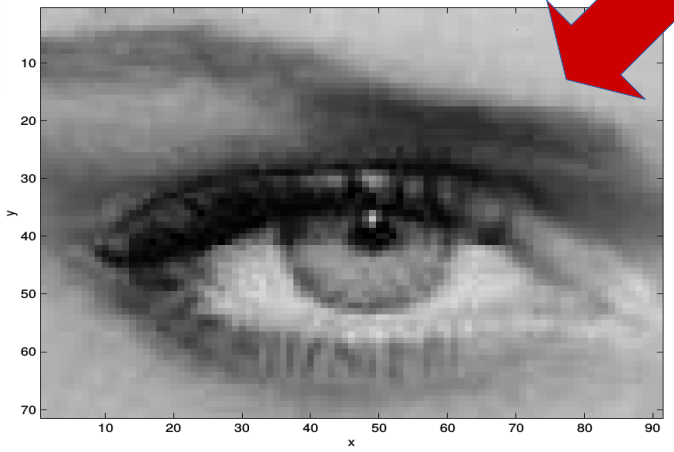
A CMOS sensor: array of photoreceptors, each sensor has its own amplifier



Provide a a two-dimensional brightness array: the **image**

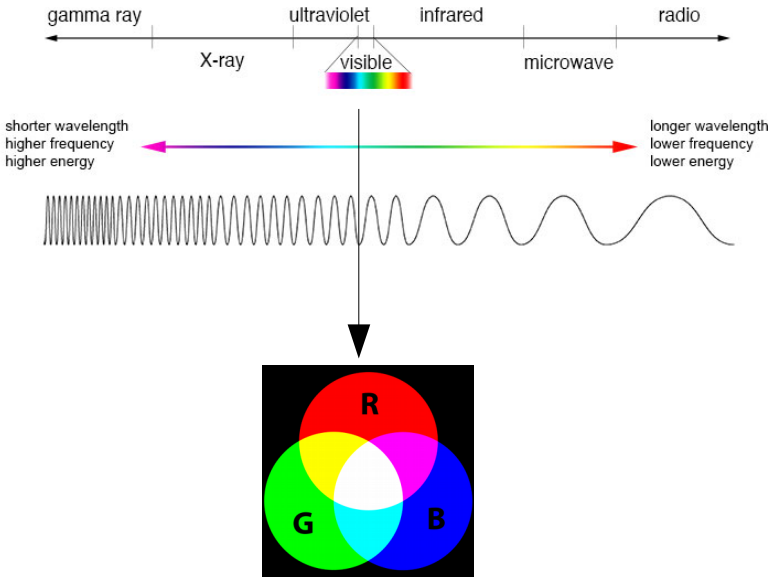


```
188 186 188 187 168 130 101  99 110 113 112 107 117 140 153 153 156 158 156 153
189 189 188 181 163 135 109 104 113 113 110 109 117 134 147 152 156 163 160 156
190 190 188 176 159 139 115 106 114 123 114 111 119 130 141 154 165 160 156 151
190 188 188 175 158 139 114 103 113 126 112 113 127 133 137 151 165 156 152 145
191 185 189 177 158 138 110  99 112 119 107 115 137 140 135 144 157 163 158 150
193 183 178 164 148 134 118 112 119 117 118 106 122 139 140 152 154 160 155 147
185 181 178 165 149 135 121 116 124 120 122 109 123 139 141 154 156 159 154 147
175 176 176 163 145 131 120 118 125 123 125 112 124 139 142 155 158 158 155 148
170 170 172 159 137 123 116 114 119 122 126 113 123 137 141 156 158 159 157 150
171 171 173 157 131 119 116 113 114 118 125 113 122 135 140 155 156 160 160 152
174 175 176 156 128 120 121 118 113 112 123 114 122 135 141 155 155 158 159 152
176 174 174 151 123 119 126 121 112 108 122 115 123 137 143 156 155 152 155 150
175 169 168 144 117 117 127 122 109 106 122 116 125 139 145 158 156 147 152 148
179 179 180 155 127 121 118 109 107 113 125 133 130 129 139 153 161 148 155 157
176 [illegible] 181 153 122 115 113 106 105 109 123 132 131 131 140 151 157 149 156 159
180 [illegible] 177 147 115 110 111 107 107 105 120 132 133 133 141 150 154 148 155 157
[illegible] 0 141 113 111 115 112 113 105 119 130 132 134 144 153 156 148 152 151
[illegible] 140 114 114 118 113 112 107 119 128 130 134 146 157 162 153 153 148
[illegible] 1 142 114 114 116 110 108 104 116 125 128 134 148 161 165 159 157 149
[illegible] 1 138 109 110 114 110 109  97 110 121 127 136 150 160 163 158 156 150
```



A "**picture**" of the image: a picture is different representation of the image, i.e. a scene that produces on the imaging sensor the same image as the original scene.

Perceive colors: the RGB model

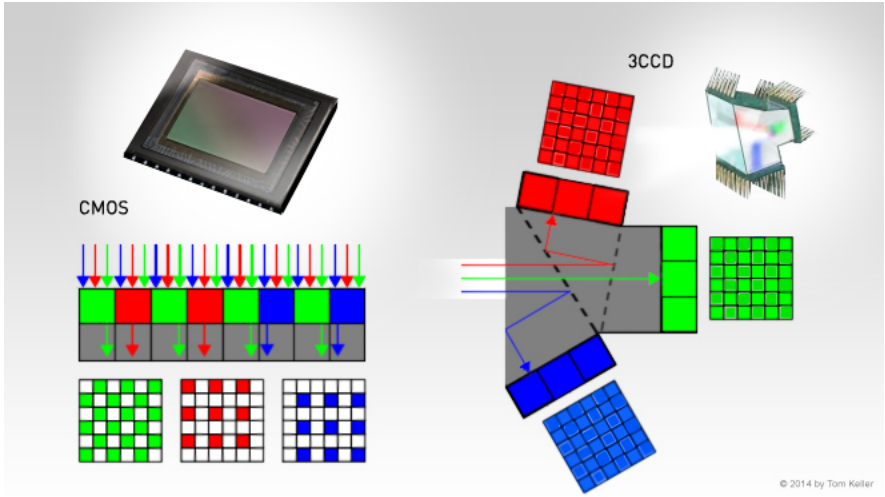


Additive color system: the **RGB model**

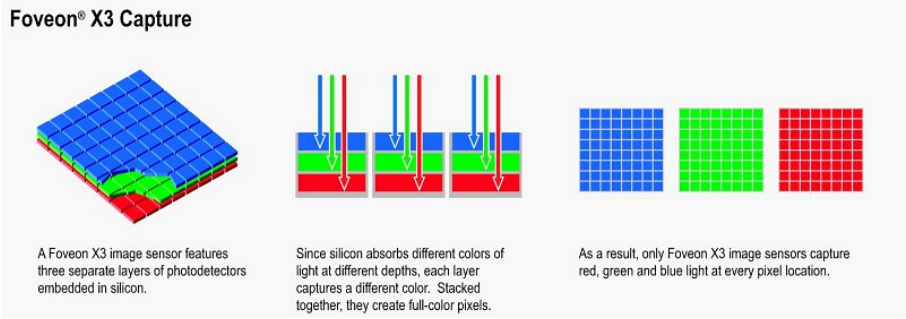
Currently, 3 approaches:

Bayer pattern

3 sensors



Three layer sensor



[Source: nasa.gov]

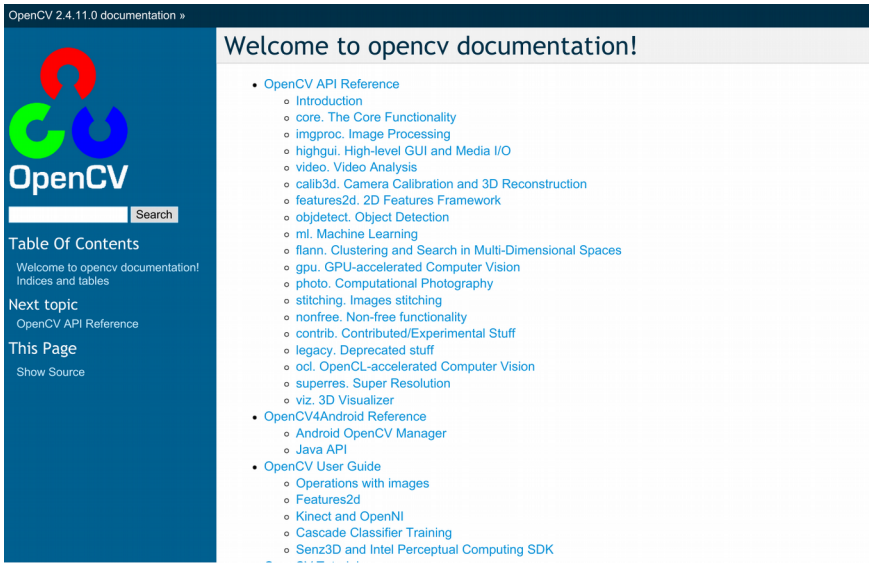
The OpenCV libraries

http://opencv.org/

- OpenCV (Open Source Computer Vision) is a library of programming functions for real-time computer vision.
- BSD Licensed - free for commercial use
- C++, C, Python and Java (Android) interfaces
- Supports Windows, Linux, Android, iOS and Mac OS
- More than 2500 optimized algorithms
- Installation tutorial:
https://www.learnopencv.com/install-opencv3-on-ubuntu/

OpenCV documentation

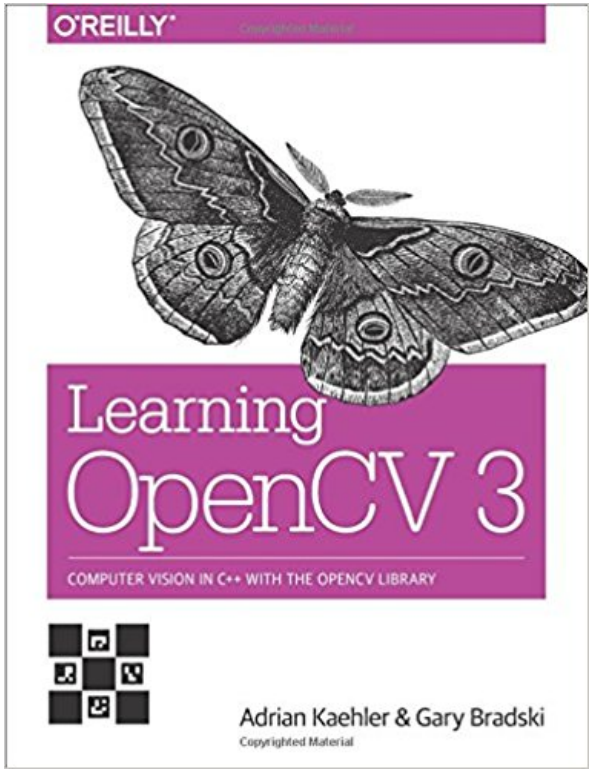
Online documentation:
http://docs.opencv.org



Adrian Kaehler, Gary Bradski

Learning OpenCV 3

O'Reilly Media



OpenCV Modules

OpenCV has a modular structure:

- Core
- Imgproc
- Video
- Calib3d
- Features2d
- Objdetect
- Highgui
- Gpu

To include single modules:

- #include <opencv2/core/core.hpp>
- #include <opencv2/highgui/highgui.hpp>

Image Processing modules (1/2)

core - a compact module defining basic data structures, including the dense multi-dimensional array Mat and basic functions used by all other modules.

imgproc - an image processing module that includes linear and non-linear image filtering, geometrical image transformations (resize, affine and perspective warping, generic table-based remapping), color space conversion, histograms, and so on.

OpenCV

Image Processing modules (2/2)

features2d - salient feature detectors, descriptors, and descriptor matchers.

highgui - an easy-to-use interface to video capturing, image and video codecs, as well as simple UI capabilities.

OpenCV

Data types

Set of primitive data types the library can operate on:

- `uchar`: 8-bit unsigned integer
- `schar`: 8-bit signed integer
- `ushort`: 16-bit unsigned integer
- `short`: 16-bit signed integer
- `int`: 32-bit signed integer
- `float`: 32-bit floating-point number
- `double`: 64-bit floating-point number

OpenCV

Images in OpenCV: cv::Mat

```
// Creates two images (i.e., just the header parts, no data)
cv::Mat A, C;
// Read an image from the disk
A = cv::imread(“picture.jpg”, cv::IMREAD_COLOR);
// Use the copy constructor (copy by reference)
cv::Mat B(A);
// Assignment operator (copy by reference)
C = A;
// Creates a new matrix D with data copied from A
cv::Mat D = A.clone();
// Creates the header for E with no data
cv::Mat E;
// Sets the data for E (copied from A)
A.copyTo(E);
```

OpenCV

Creating a Mat from scratch

```
cv::Mat M(Size(2,2), cv::DataType<Vec3b>::type,
cv::Scalar(0,0,255));
cout << "M = " << endl << " " << M << endl << endl;
```

```
M =
 [0, 0, 255, 0, 0, 255;
  0, 0, 255, 0, 0, 255]
```

```
cv::Mat M;
M.create(Size(4,4), cv::DataType<Vec2b>::type);
M = cv::Scalar(205,205);
cout << "M = "<< endl << " "  << M << endl << endl;
```

```
M =
 [205, 205, 205, 205, 205, 205, 205, 205;
  205, 205, 205, 205, 205, 205, 205, 205;
  205, 205, 205, 205, 205, 205, 205, 205;
  205, 205, 205, 205, 205, 205, 205, 205]
```

OpenCV

MATLAB style initializer

```
cv::Mat E = cv::Mat::eye(4, 4, CV_64F);
cout << "E = " << endl << " " << E << endl << endl;

cv::Mat Z = cv::Mat::zeros(3,3, CV_8UC1);
cout << "Z = " << endl << " " << Z << endl << endl;

cv::Mat O = cv::Mat::ones(2, 2, CV_32F);
cout << "O = " << endl << " " << O << endl << endl;
```

OpenCV

Cv::Mat in memory

	Column 0	Column 1	Column ...	Column m
Row 0	0,0	0,1	...	0, m
Row 1	1,0	1,1	...	1, m
Row ...	...,0	...,1	...	..., m
Row n	n,0	n,1	n,...	n, m

gray scale image

	Column 0			Column 1			Column ...			Column m		
Row 0	0,0	0,0	0,0	0,1	0,1	0,1	...	...	...	0, m	0, m	0, m
Row 1	1,0	1,0	1,0	1,1	1,1	1,1	...	...	...	1, m	1, m	1, m
Row ...	...,0	...,0	...,0	...,1	...,1	...,1	...	...	...	..., m	..., m	..., m
Row n	n,0	n,0	n,0	n,1	n,1	n,1	n,...	n,...	n,...	n, m	n, m	n, m

BGR image

Note that:

-The data is order row-major order
-Color channels in OpenCV is BGR instead of RGB.

OpenCV

How to scan gray scale images(1/2)

```
// Easy way
cv::Mat I = ...
...
for( int r = 0; r < I.rows; ++r)
{
  for( int c = 0; c < I.cols; ++c)
  {
    uchar g = I.at<uchar>(r,c);
    // ...
  }
}
```

OpenCV

How to scan gray scale images(2/2)

```
// Faster way
cv::Mat I = ...
...
for( int r = 0; r < I.rows; ++r)
{
  uchar *r_p = I.ptr<uchar>(r);
  for( int c = 0; c < I.cols; ++c, r_p++)
  {
    uchar g = *r_p;
    // ...
  }
}
```

OpenCV

How to scan RGB images

```
cv::Mat I = ...
…
for( int r = 0; r < I.rows; ++r)
{
  for( int c = 0; c < I.cols; ++c)
  {
    uchar blue = I.at<cv::Vec3b>(r,c)[0];
    uchar green = I.at<cv::Vec3b>(r,c)[1];
    uchar red = I.at<cv::Vec3b>(r,c)[2];
    …
  }
}
// Faster way as exercise :)
```

OpenCV

OpenCV and ROS (1/2)

Use the vision_opencv (meta) package

In your CmakeLists.txt:

```
find_package(OpenCV)
include_directories(${OpenCV_INCLUDE_DIRS})
target_link_libraries(my_awesome_library ${OpenCV_LIBRARIES})
```

From ROS image messages to cv::Mat:

```
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
...
sensor_msgs::ImageConstPtr& msg;
...
cv_bridge::CvImagePtr cv_ptr =
             bridge::toCvCopy(msg, sensor_msgs::image_encodings::BGR8);
// Or, if is possible, toCvShare()
cv::imshow(OPENCV_WINDOW, cv_ptr->image);
cv::waitKey(10);
```

OpenCV

OpenCV and ROS (2/2)

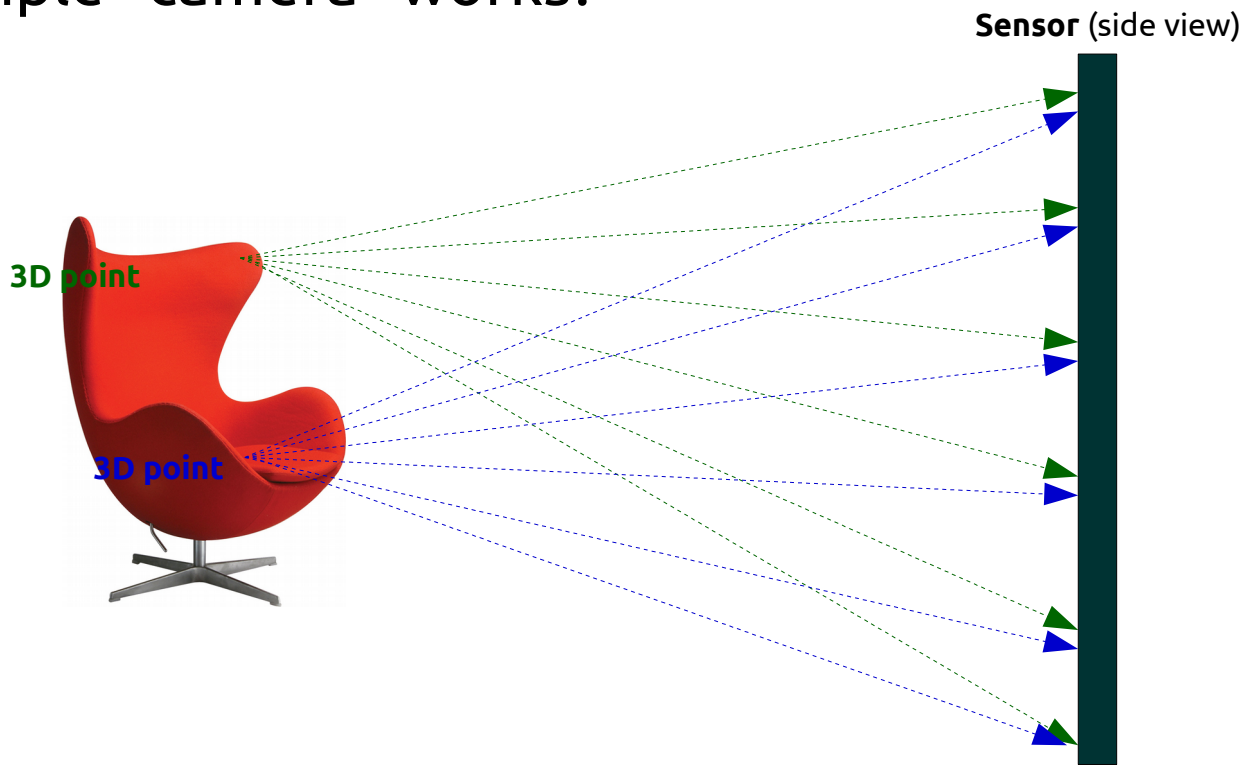
From cv::Mat to ROS image messages:

```
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>

...

cv::Mat my_img = …;
CvImage ros_img;
ros_img->image = my_img;
ros_img->encodings = bgr8;

sensor_msgs::ImagePtr msg = ros_img->toImageMsg()
```

OpenCV

Camera model

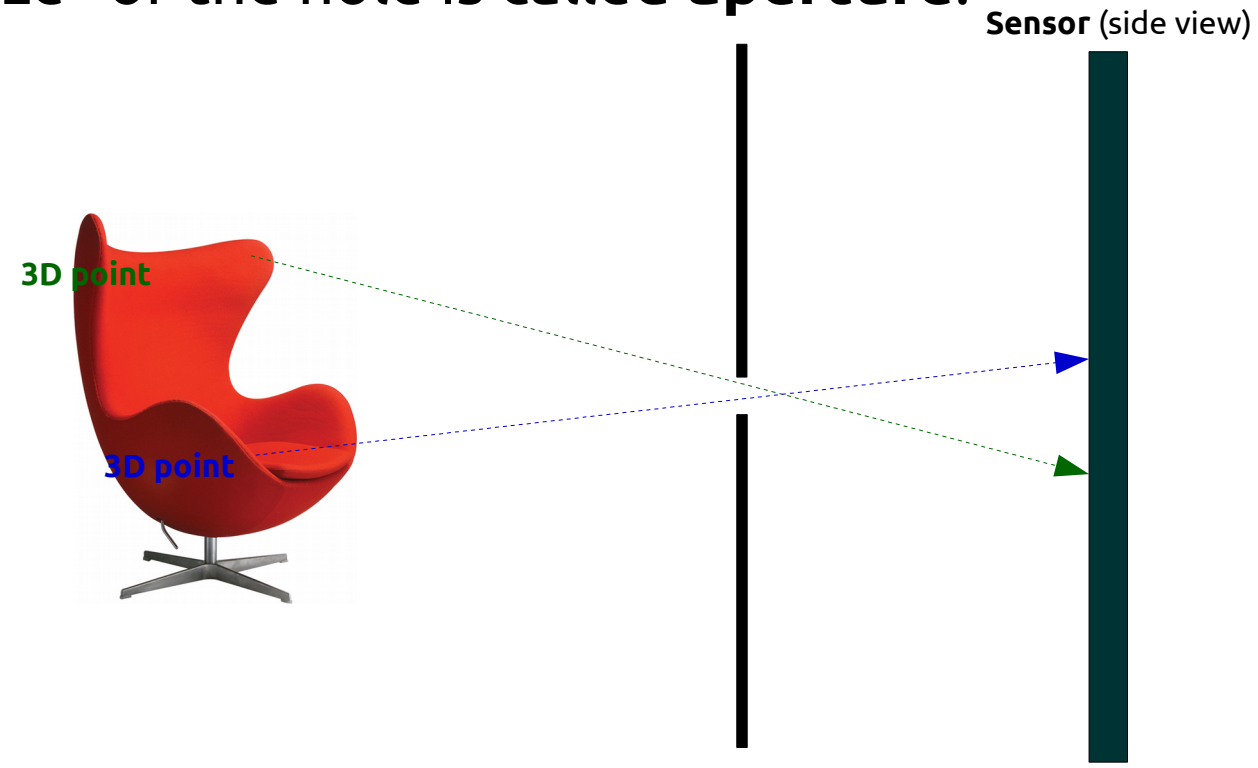
Put the sensor (e.g., the CMOS) in front of an object. Does this simple “camera” works?



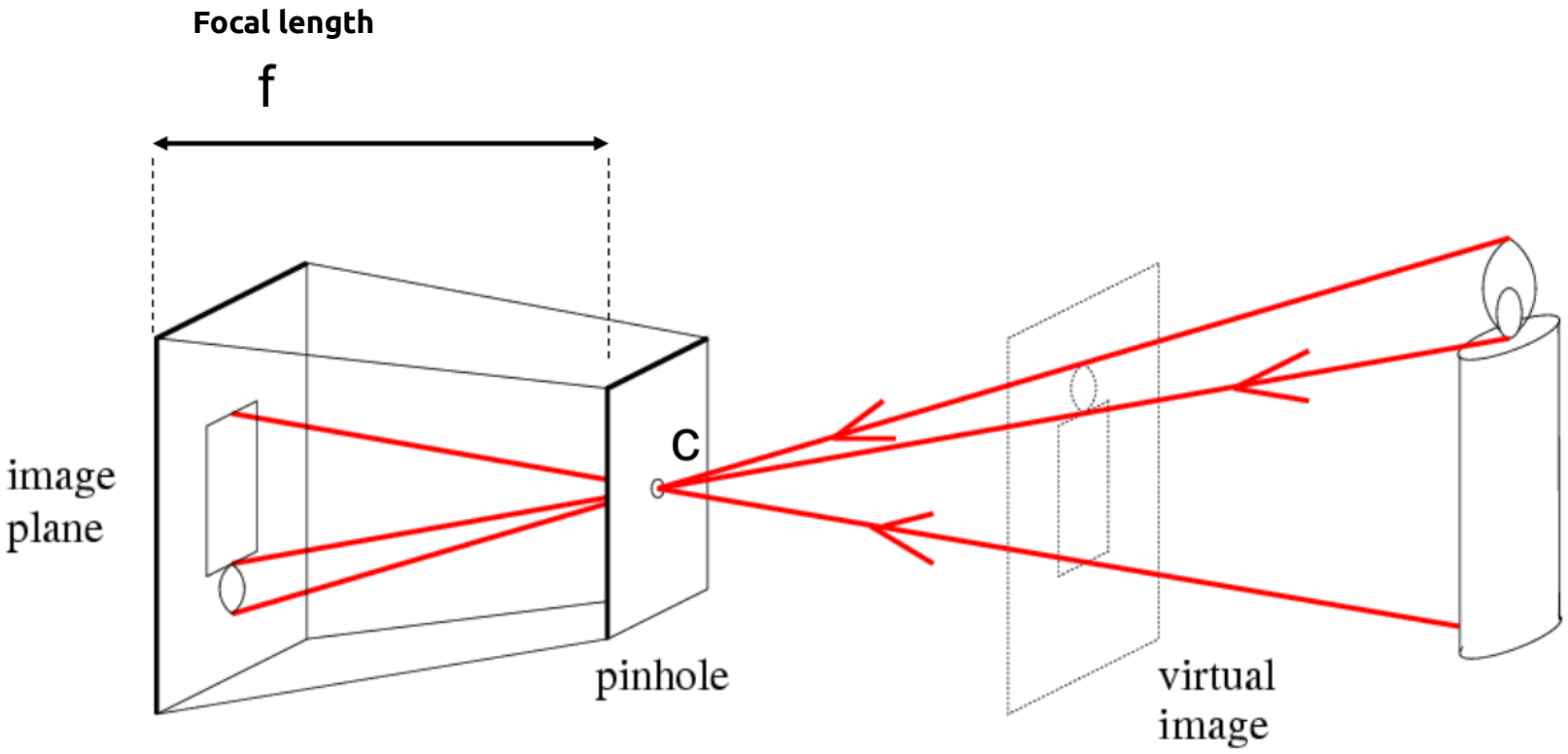
Pinhole camera model (1/2)

Idea: add a barrier to block off most of the rays!

The “size” of the hole is called **aperture**.



Pinhole camera model (2/2)

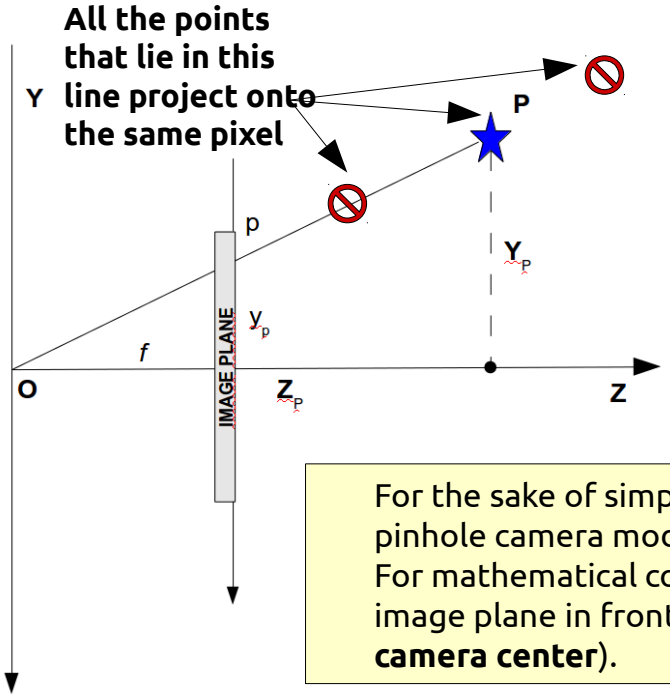


From 3D to 2D (1/2)

A 3D scene is “projected” on the 2D the image plane

Dimensionality reduction → **the depth (i.e., the z coordinate) of each point is lost!**

The standard coordinate system of the pinhole camera system seen from the X axis.

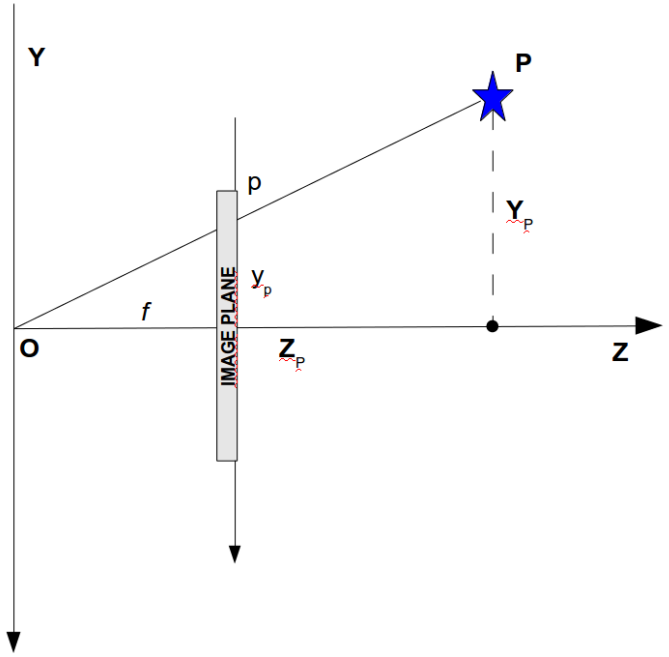


$$x_p = \frac{X_P f}{Z_P} \quad , \quad y_p = \frac{Y_P f}{Z_P}$$

For the sake of simplicity we will use the pinhole camera model.
For mathematical convenience, we put the image plane in front of the focal point (i.e., the **camera center**).

From 3D to 2D (2/2)

Use homogeneous coordinates!



$$x_p = \frac{X_P f}{Z_P} \quad , \quad y_p = \frac{Y_P f}{Z_P}$$

Converting to *homogeneous* coordinates

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w) \qquad \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_P \\ Y_P \\ Z_P \\ 1 \end{bmatrix}$$

Other real camera parameters

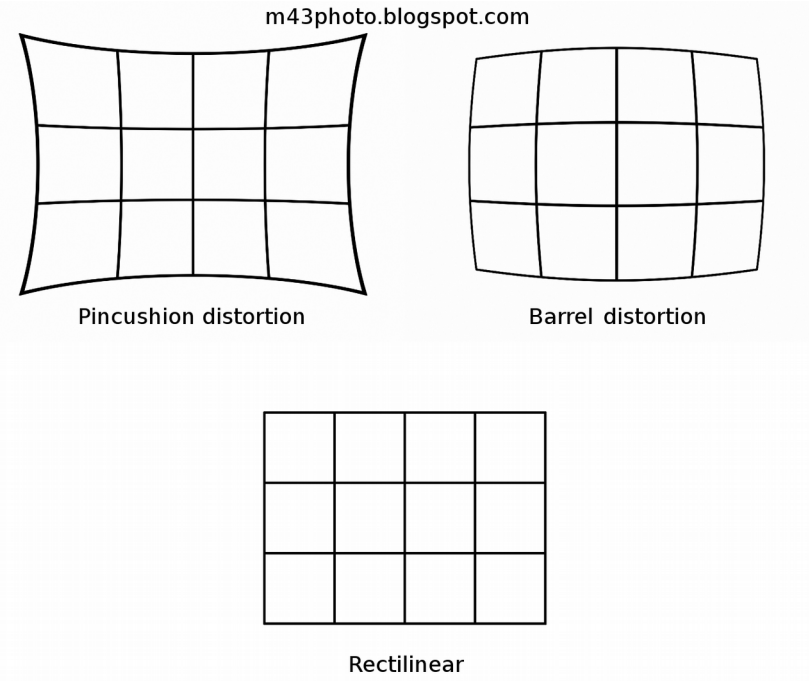
Pixel size

Image center

Distortion

Hints on radial distortion

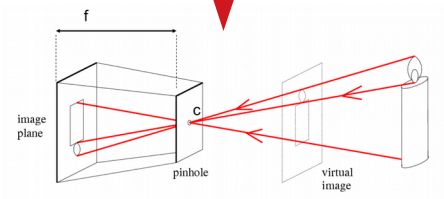
In many cases, like in the wide-angles camera applications, the lens distortion should be taken into account in the perspective projection: distortion is modeled by the nonlinear intrinsic parameters.



Camera Calibration

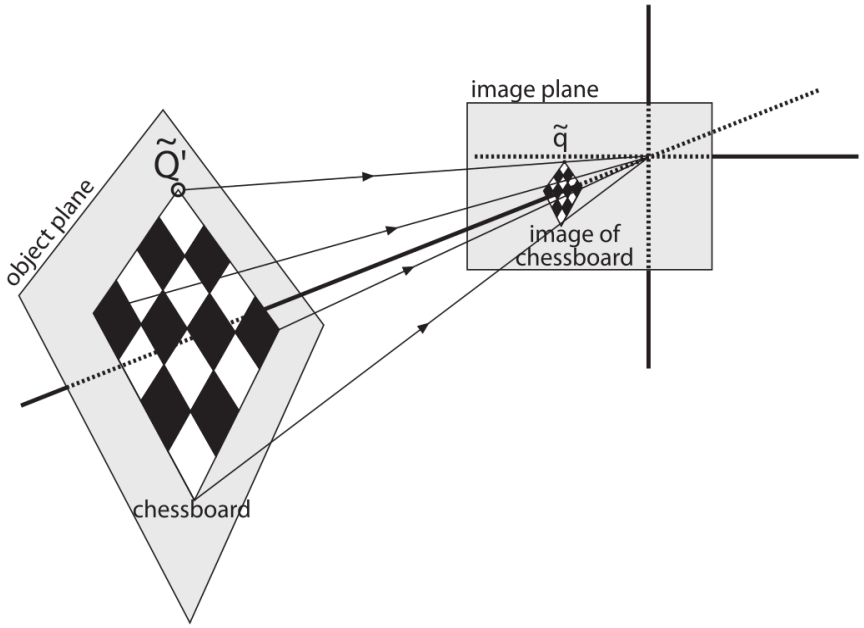
Calibration: the process of estimating the parameters (focal length, pixel size, camera center, distortion parameters) of a general camera

Once calibrated, a specific camera can easily led back to a general, pinhole camera → the **canonical camera**



Hints camera calibration

The camera calibration process aims to estimate the camera intrinsic parameters and the radial distortion parameters.



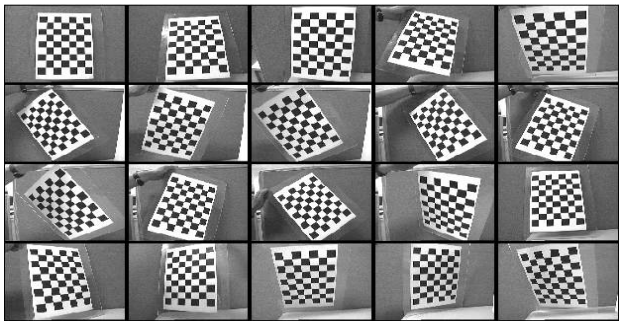
Basic idea of the most popular calibration technique [Zhang00], [Bouguet]:

-Use a known planar pattern (e.g., a chessboard)

-Collect a sequence of images of the pattern in several positions, and extract all corners

-Find the parameters set that minimize the squared distances in the image space, using conventional least square methods (e.g., Levenberg Marquardt)

Calibration in OpenCV (1/2)



```
// We assume to use a 5x7 checker board (square size 30 mm) and to have
// sequence of 1280x1024 images of the checkerboard stored in the disk
// (the image names are stored in the array img_name[x])
cv::Size board_size(7,5), image_size(1280,1024);
float square_size = 0.03;
vector<vector<cv::Point2f> > image_points;
// Extract the corners from each image
for(int i = 0; i< num_images;++i)
{
  vector<cv::Point2f> point_buf;
  // Read an image from the disk, and convert to gray levels
  cv::Mat img = cv::imread(img_name[i], cv::IMREAD_GRAYSCALE), img_gray;
  // Find the corners
  bool found = cv::findChessboardCorners( img, board_size, point_buf,
               CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);
  // Refine the corners pose
  if ( found )
  {
     cornerSubPix( img, point_buf, cv:Size(11,11),
                   Size(-1,-1), TermCriteria( CV_TERMCRIT_EPS+CV_TERMCRIT_ITER, 30, 0.1 ));
    // Collect the new corners points
    image_points.push_back(point_buf);
  }
}
```

OpenCV

Calibration in OpenCV (2/2)

```
// Prepare the 3D points (i.e., the checkerboard corners that
// will be projected onto the image)
vector< vector<cv::Point3f> > object_points;
vector<cv::Point3f> corners;
for( int i = 0; i < board_size.height; ++i )
  for( int j = 0; j < board_size.width; ++j )
    corners.push_back(cv::Point3f(float( j*square_size ),
float( i*square_size ), 0));
for(int i = 0; i< num_images;++i)
  object_points.push_back( corners );
// Prepare the object where the parameters will be stored
cv::Mat K = cv::Mat::eye(3, 3, CV_64F);
cv::Mat dist_coeffs = Mat::zeros(8, 1, CV_64F);
vector<cv::Mat> rvecs, vector<cv::Mat> tvecs;
// And run the calibration
cv::calibrateCamera(object_points, image_points, image_size, K,
                    dist_coeffs, rvecs, tvecs,
                    cv::CALIB_FIX_K4|cv::CALIB_FIX_K5);
```

See: http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html

Project points in OpenCV

```
// We use here the calibration parameters to project some 3D points onto the image plane
// Fill the world → camera rotation matrix and the translation vector
cv::Mat R(3,3,cv::DataType<double>::type);
cv::Mat t(3,1,cv::DataType<double>::type), r_vec;
R.at<double>(0,0) = 0.3; R.at<double>(0,1) = 0.001; ...
…
// Prepare some 3D points
vector<cv::Point3d> pts_3d;
pts_3d.push_back(cv::Point3d(3.2, 4.3, 2.3));
...
// Convert the rotation matrix in a rotation vector using the Rodrigues formluas (axis–angle representation)
cv::Rodrigues(R,r_vec);
// Project the points
vector<cv::Point2d> projected_pts;
cv::projectPoints(pts_3d, r_vec, t, K, dist_coeffs, projected_pts);
```

OpenCV

Image filtering

Compute function of local neighborhood at each position

Really important:

- Enhance images (de-noise, resize, increase contrast, etc...)
- Extract information from images (texture, edges, distinctive points, etc...)
- Detect patterns

Image filters in spatial domain: modify the pixels in an image based on some function of a local neighborhood of the pixels

- Filter is a mathematical operation of a grid of numbers
- Smoothing, sharpening, measuring texture

Linear filtering

Linear case is simplest and most useful

- Replace each pixel with a linear combination of its neighbors.

The prescription for the linear combination is sometimes called the “convolution kernel” (even if linear filtering uses correlation).

For symmetrical kernel, there's no difference between correlation and convolution.

Example: box (mean) filter

Simple method for reducing noise in an image

Convolution kernel

$g[\cdot,\cdot]$

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

Filtered image

Input image

$$h[m,n] = \sum_{k,l} g[k,l]\, f[m+k, n+l]$$

Example: box (mean) filter

$f[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

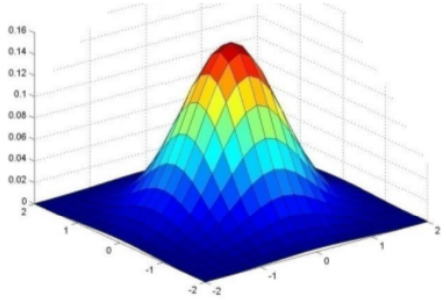
$h[.,.]$

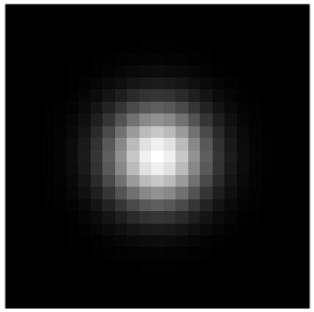
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

Gaussian Smoothing (1/2)

A better way to reduce noise and details from an image is to employ a Gaussian filter → **it removes the "high-frequency" components from the image** (low-pass filter)

Weight contributions of neighboring pixels by nearness:



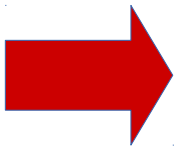


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

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

Gaussian Smoothing (2/2)



Edge detection

Edges are very simple but informative part of the image

Replace image with a binary “edge map”that highlights all the borders in the image

Sobel Operator

The Sobel operator is used to compute the 2D spatial derivatives of an image: higher gradient measurement emphasizes regions of high spatial frequency that correspond to edges.

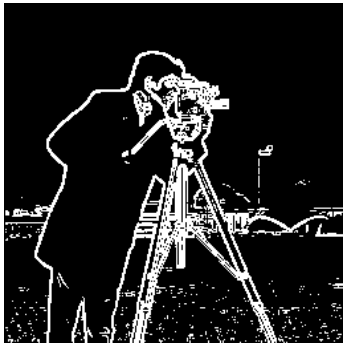
abs($\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$)

abs($\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$)

Sobel Operator

Typically it is used to find the approximate absolute gradient magnitude at each point → **edges**

abs(Sobel x) abs(Sobel y) Binary version (threshold = 0.4*max)

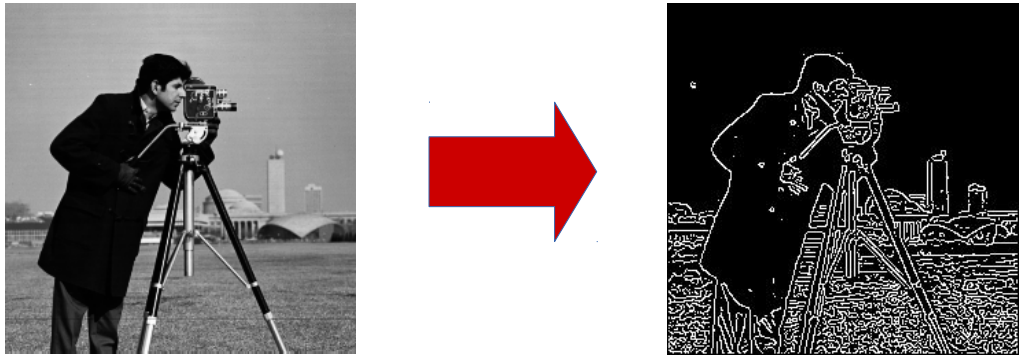
0.5*( **+** ) **=**  

Hints on the Canny edge detector

Edge localized using an operator very similar to the derivative of a Gaussian

Non-maximum suppression - remove edges orthogonal to a maxima

Hysteresis thresholding - Improved recovery of long image contours



[Canny86] J Canny, “A computational approach to edge detection”, Pattern Analysis and Machine Intelligence, IEEE Transactions on, 679-698

Filtering in OpenCV (1/2)

```
// Read an image from the disk, imposing gray level format
cv::Mat src_img_gl = cv::imread(image_name,
                     cv::IMREAD_GRAYSCALE), src_img_f;
// Convert to floating point
src_img_gl.convertTo( src_img_f,cv::DataType<float>::type );
// Map intensities in the range [0,1]
cv::normalize(src_img_f, src_img_f, 0, 1.0,
              cv::NORM_MINMAX, cv::DataType<float>::type);
cv::Mat gaussian_blurred_img, box_blurred_img;
float gaussian_stddev = 1.0;
// Filter the image with a 3x3 box filter
cv::boxFilter(src_img_f, box_blurred_img, -1, cv::Size(3,3));
// Filter the image with a gaussian filter with std dev = 1
cv::GaussianBlur( src_img_f, gaussian_blurred_img,
                  cv::Size(0,0), gaussian_stddev );
```

OpenCV

Filtering in OpenCV (2/2)

```
// Compute the image derivatives along x and y using Sobel
cv::Mat dx_img, dy_img;
cv::Sobel(src_img_f, dx_img, cv::DataType<float>::type, 1, 0, 3);
cv::Sobel(src_img_f, dy_img, cv::DataType<float>::type, 0, 1, 3);

cv::Mat gradient_mag_img, abs_dx_img, abs_dy_img, binary_img;

// Compute the gradient magnitude image
abs_dx_img = cv::abs(dx_img);
abs_dy_img = cv::abs(dy_img);
gradient_mag_img = 0.5*(abs_dx_img + abs_dy_img);

// Binarize the image
cv::threshold ( gradient_mag_img, binary_img, 0.4,
                1.0,cv::THRESH_BINARY );
```

See: http://docs.opencv.org/modules/imgproc/doc/filtering.html

Feature Detection

A feature (also called keypoint) can be defined as a meaningful, detectable parts of the image

- Corners, blobs, stable regions

Used to match points in different images

Feature should be repeatable and distinctive

- **Distinctive** : features should be easily matched between them
- **Robust** to noise, blur, discretization, compression
- **Repeatable**

Good Features to Track

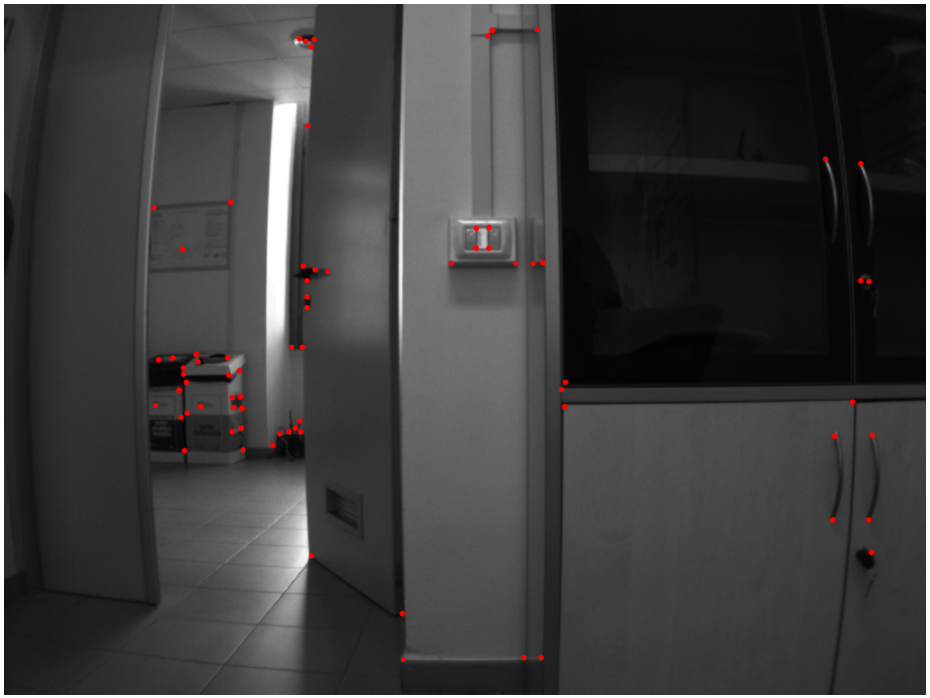
I.e., the Shi and Tomasi corner detector

Idea:

-A shifted corner produces some difference in the image

-A shifted uniform region produces no difference

→ **look for large difference in shifted image**



[GFTT] J. Shi and C. Tomasi, “Good features to track”, In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pages 593–600, 1994

Homework

The goal of this homework is to familiarize on the use of OpenCV's basic structures and algorithms

In a ROS node:

- Use OpenCV to read a video stream
- To each frame in the video apply at least 3 filters
- Send the modified frames to a topic

In another node:

- Read the frames from the topic
- Show the video with the modified frames
- Save the video with the modified frames

Hints: follow this tutorial https://www.learnopencv.com/read-write-and-display-a-video-using-opencv-cpp-python/