CV Practice Class
4. Camera Calibration and 3D Reconstruction

2017-07-06

FIRA 인공지능 에이전트 과정
SNUVL Lab

Contents

About 3D Reconstruction

1. If you want to know A-to-Z about Multi-View Geometry,
   there is AWESOME tutorial, SFMedu, from Princeton university.

2. If you want to build Multi-View Geometry application, OpenCV is NOT a good choice.
   Try OpenMVG and OpenMVS and PCL

3. If you want to know useful materials about 3D vision,
   visit awesome_3DReconstruction_list

Goal of Multi-view Stereo System

Get 3D structure from multiple 2D images!

(Image credit: SFMedu)

Camera Calibration

Camera Model
Lens Distortion
Camera Calibration

Camera Model

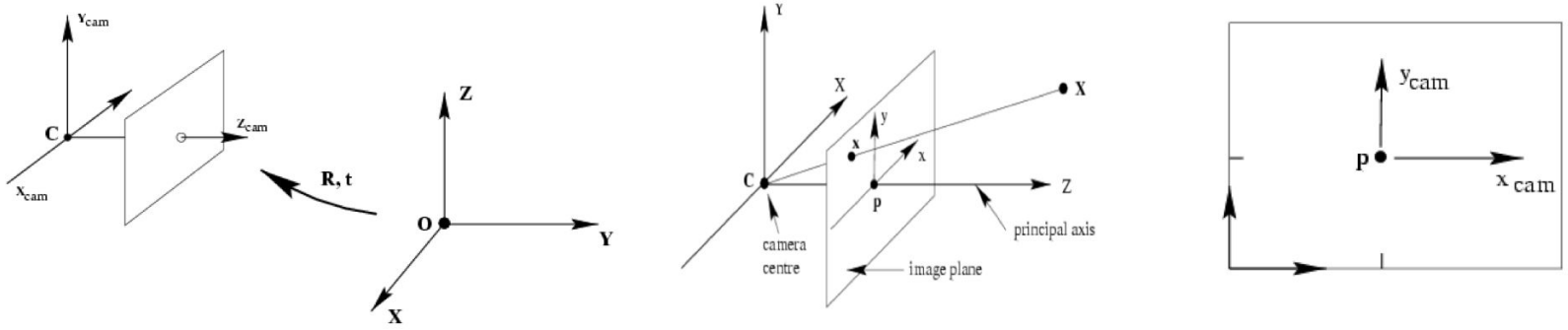
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & T \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Image coord. | Camera internal parameter | Perspective projection | 3-D point in camera coord. | 3-D point in world coord.

$$\mathbf{x} = \mathbf{A}\,[\mathbf{R} \mid \mathbf{t}]\,\mathbf{X}$$

Intrinsic **Extrinsic**

World ⇒ Camera ⇒ Image



Camera Model

[Camera model demo]

f_x, f_y : focal lengths expressed in pixel units

c_x, c_y : principal point that is usually at the image center

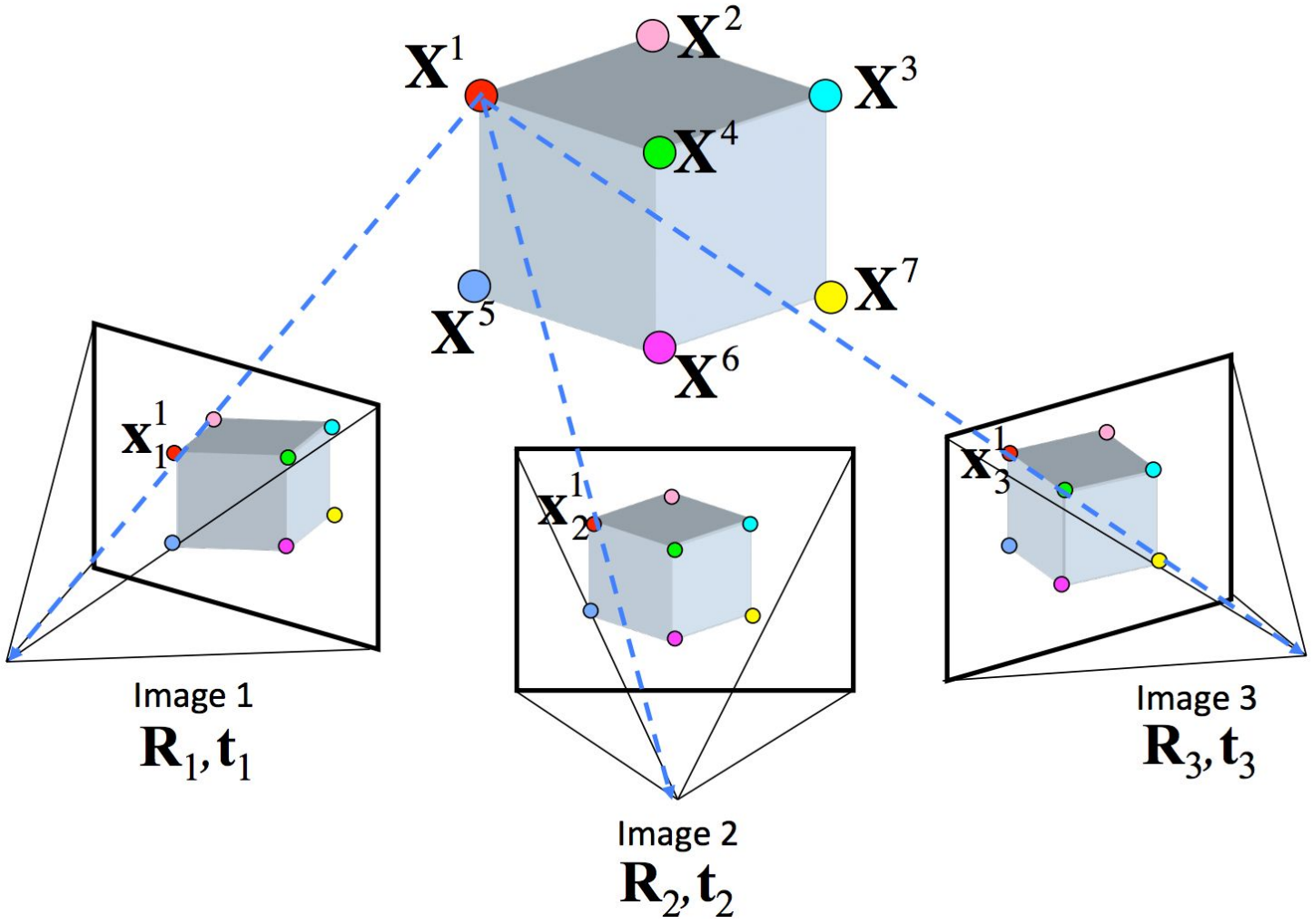
s : skew coefficient. Usually 0

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}}$$

Camera Model

When people take SAME object with SAME camera...



(Image credit: SFMedu)

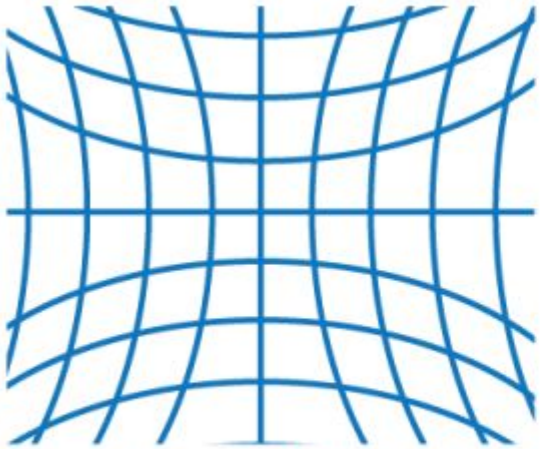
Camera Calibration

	Point 1	Point 2	Point 3
Image 1	$\mathbf{x}_1^1 = \mathbf{K}\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right]\mathbf{X}^1$	$\mathbf{x}_1^2 = \mathbf{K}\left[\mathbf{R}_1 \middle| \mathbf{t}_1\right]\mathbf{X}^2$	
Image 2	$\mathbf{x}_2^1 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^1$	$\mathbf{x}_2^2 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^2$	$\mathbf{x}_2^3 = \mathbf{K}\left[\mathbf{R}_2 \middle| \mathbf{t}_2\right]\mathbf{X}^3$
Image 3	$\mathbf{x}_3^1 = \mathbf{K}\left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]\mathbf{X}^1$		$\mathbf{x}_3^3 = \mathbf{K}\left[\mathbf{R}_3 \middle| \mathbf{t}_3\right]\mathbf{X}^3$

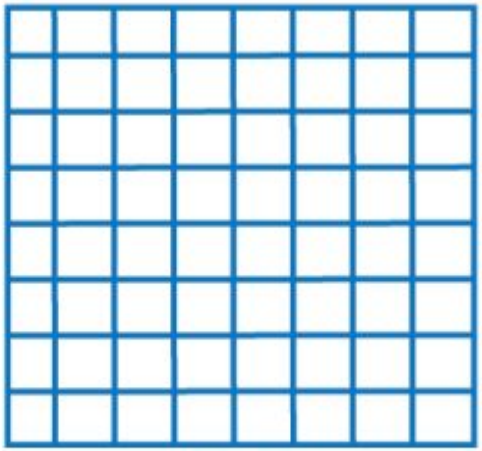
Same Camera Same Setting = Same $\mathbf{K}$

(Image credit: SFMedu)

Lens Distortion



negative radial distortion
"pincushion"



no distortion

positive radial distortion
"barrel"

Lens Distortion

- We will use Brown's distortion model with 5-parameters

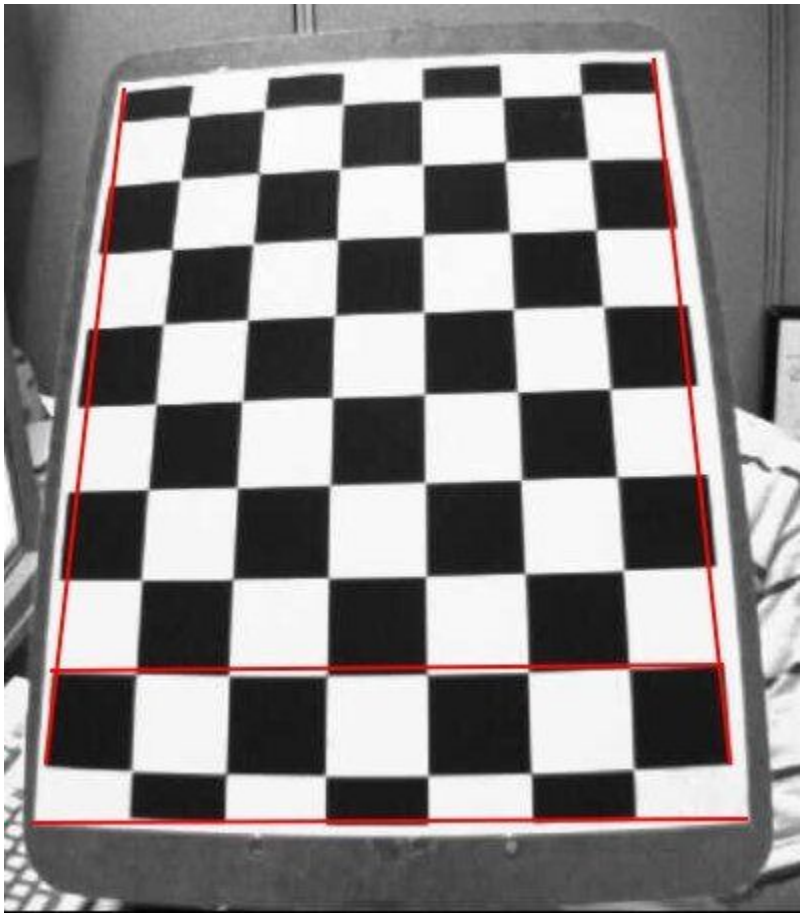
$$[k_1, k_2, p_1, p_2, p_3]$$

- The distortion is solved as follows:

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

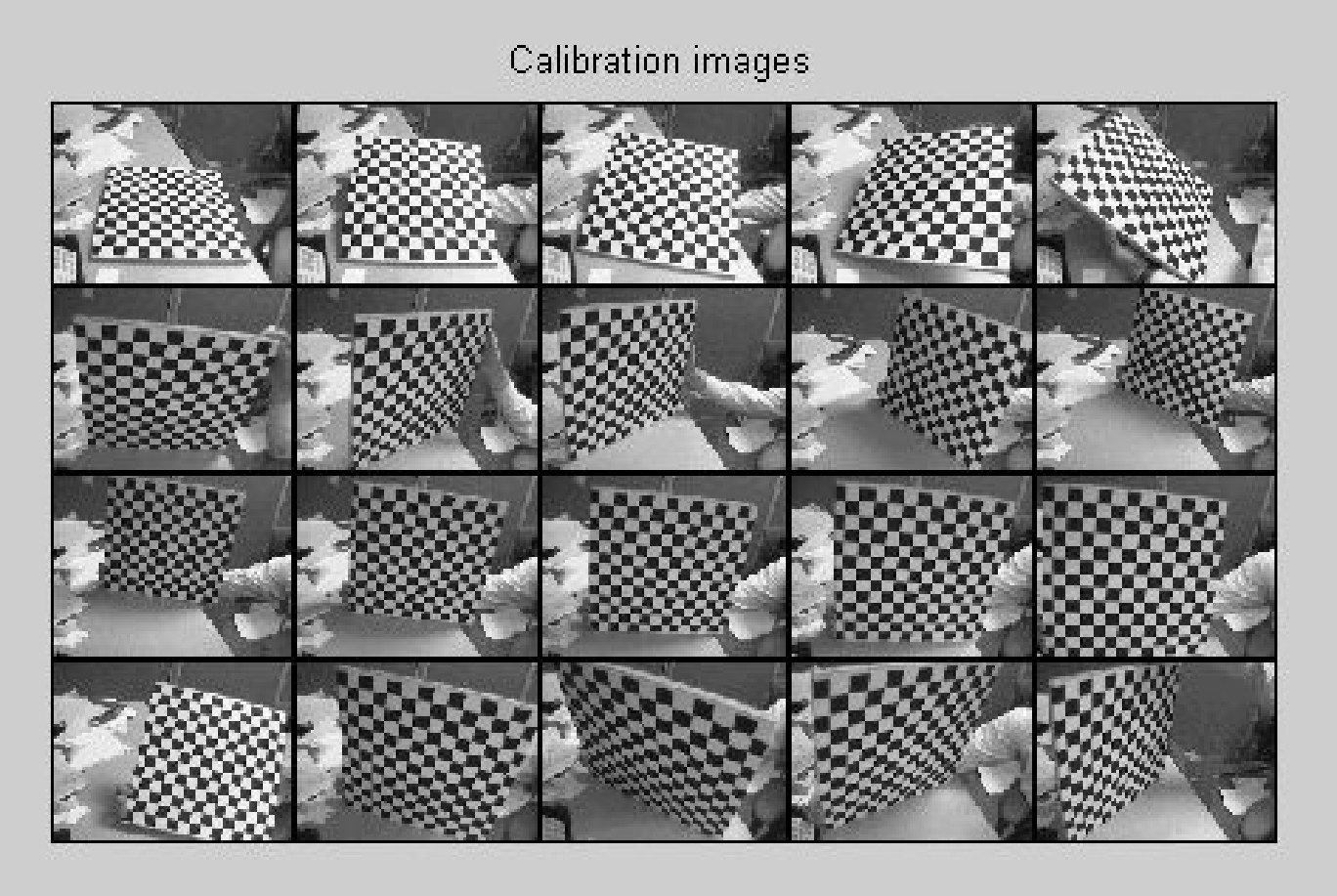
Camera Calibration

- We will use chessboard for camera calibration



Camera Calibration

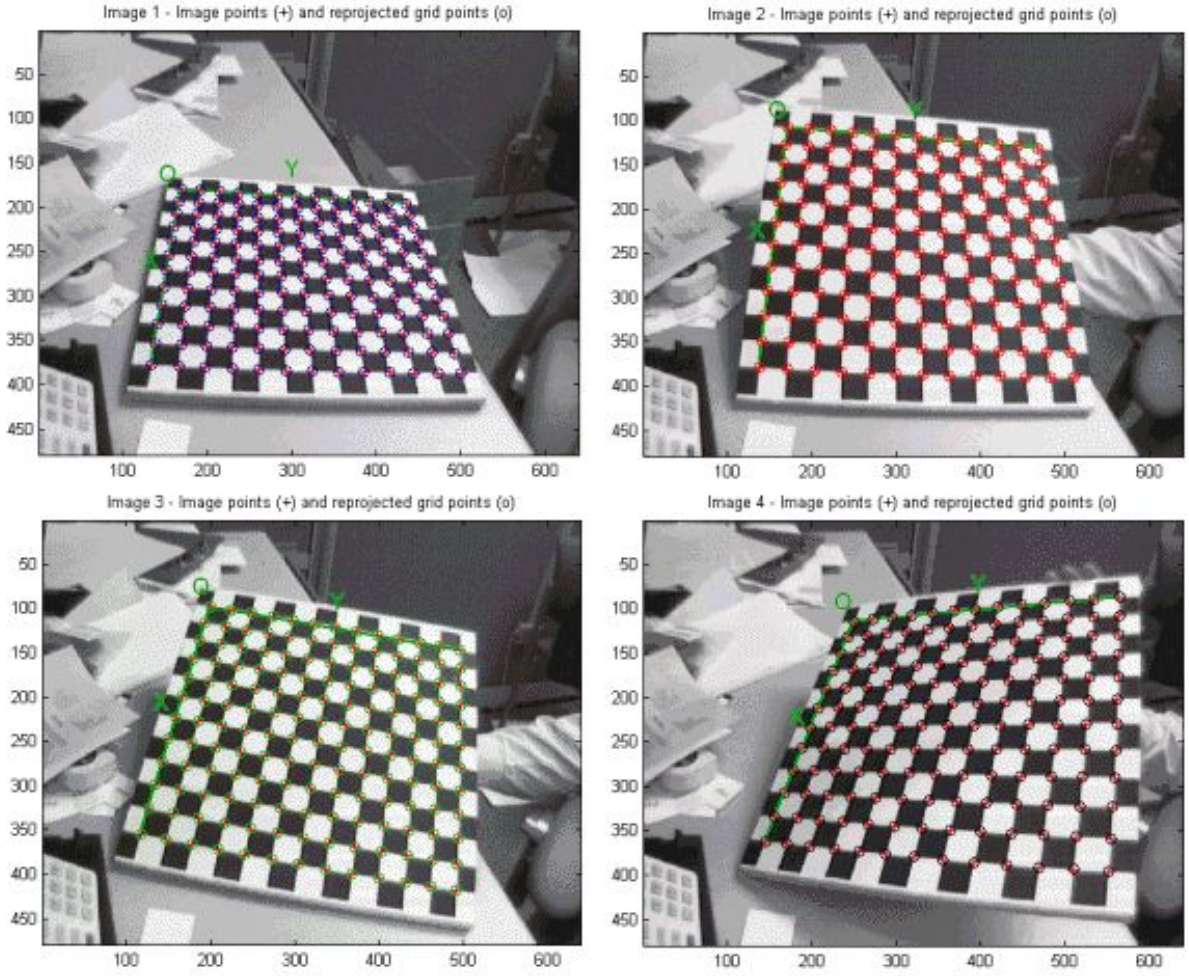
- First, take multiple chessboard images



(Image credit: Caltech Vision lab)

Camera Calibration

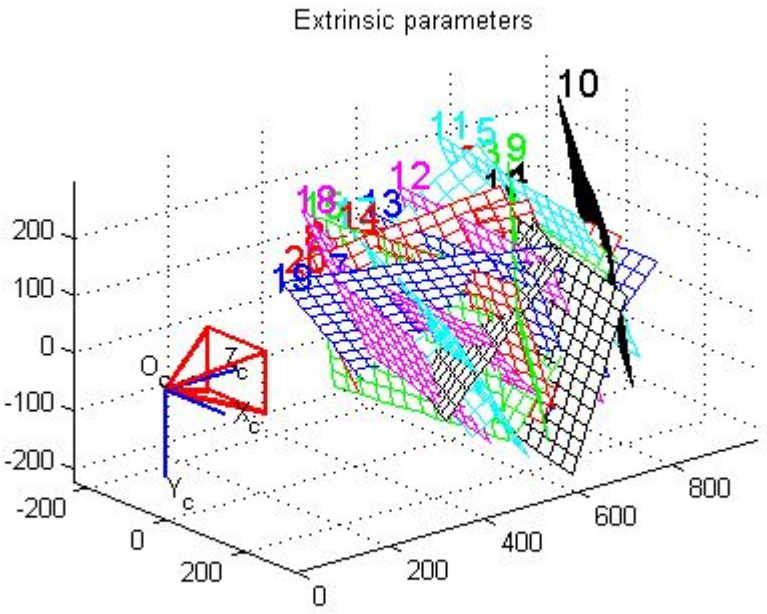
- Second, run calibration algorithms



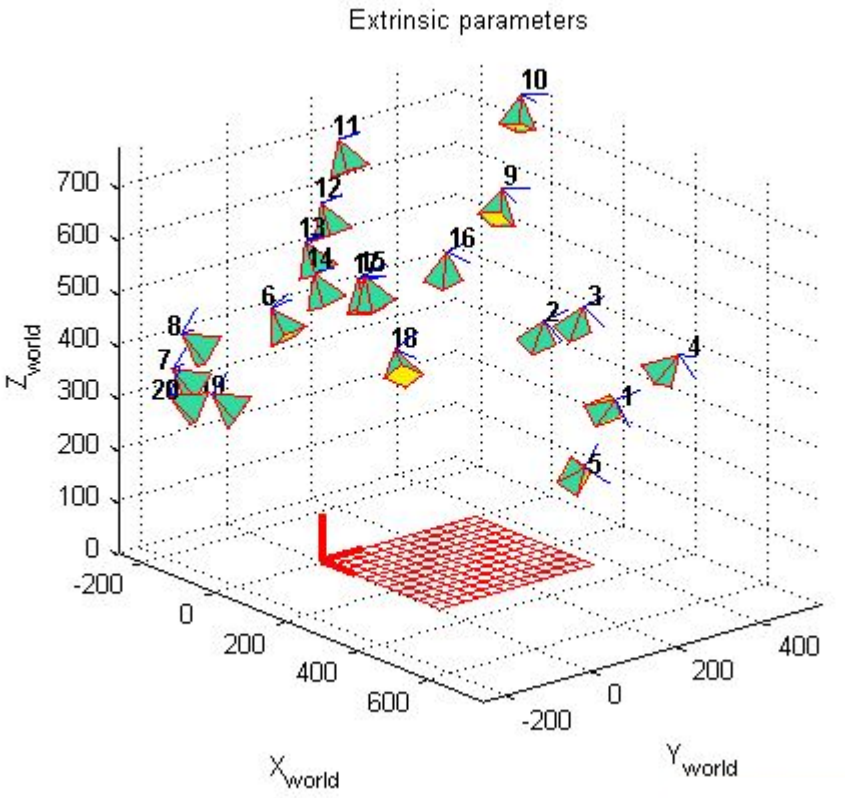
(Image credit: Caltech Vision lab)

Camera Calibration

- Calibration results



Before calibration



After calibration

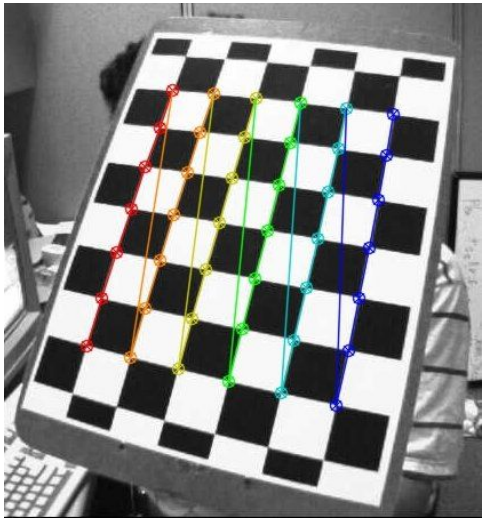
(Image credit: Caltech Vision lab)

Camera Calibration: Find Chessboard Patterns

- Find chessboard corners from gray-scale image

```
found, corners = cv2.findChessboardCorners(img, pattern_size)
```

  - `img`: chessboard image. 8-bit grayscale or color image.
  - `pattern_size`: Number of inner corners per a chessboard row and column. e.g. (7, 6)
  - `found`: True if chessboard found else False.
  - `corners`: output array of detected corners

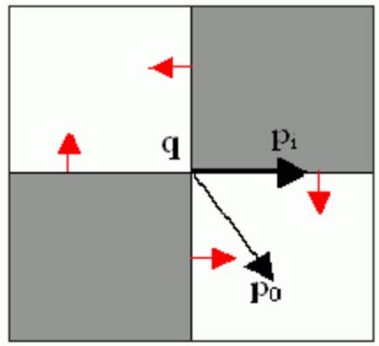


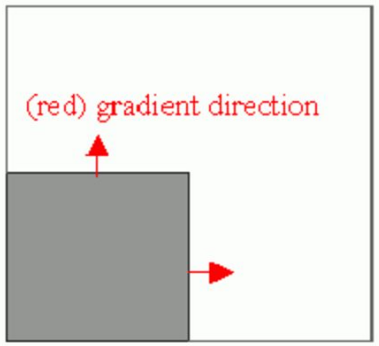
Camera Calibration: Find Chessboard Patterns

- Refines corner locations (Note that there is no return value!)

```
cv2.cornerSubPix(gray,corners,winsize,zerozone,criteria)
```

  - `gray`: input image
  - `corners`: initial coordinates of the input corners and refined coordinates provided for output
  - `winsize`: half of the side length of the search window
  - `zerozone`: output array of detected corners. (-1, -1) if there is no such region
  - `criteria`: criteria for termination of the iterative process of corner refinement





Camera Calibration: Calibrate!

- Chessboard operator in OpenCV

```
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints,
imgpoints, imgsize, None, None)
```

  - `objpoints`: (X, Y, Z) coordinates of chessboard patterns
  - `imgpoints`: (X, Y) coordinate of chessboard pattern in image
  - `imgsize`: image size in (width, height)
  - `ret`: RMS error of camera calibration
  - `mtx`: camera intrinsic parameter matrix
  - `dist`: distortion coefficients
  - `rvecs`: output vector of rotation vectors
  - `tvecs`: output vector of translation vectors

Camera Calibration: Refine Camera Matrix

- Refine camera intrinsic matrix

```
newcameramatrix, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, imgsize, alpha, new_imgsize)
```

  - `mtx:` input camera matrix
  - `dist:` input factor of distortion coefficients
  - `imgsize:` original image size
  - `alpha:` free scaling parameter between 0 and 1
  - `new_imgsize:` image size after rectification
  - `newcameramatrix:` output new camera matrix
  - `roi:` output rectangle that outlines all-good-pixels region in the undistorted image

Camera Calibration: Undistort Image

- Refine camera intrinsic matrix

```
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
```

  - `img`: input (distorted) image
  - `mtx`: input camera matrix
  - `dist`: input vector of distortion coefficients
  - `newcameramtx`: refined camera matrix
  - `dst`: output (corrected) image that has the same size as img

Camera Calibration: Calculate RMS Error

- Projection operator in OpenCV

```
imagePoints, _ = cv2.projectPoints(objPoints, rvec, tvec,
cameraMatrix, dist)
```

  - `objpoints`: (X, Y, Z) coordinates of chessboard patterns
  - `rvec`: rotation vector
  - `tvec`: translation vector
  - `cameraMatrix`: camera intrinsic matrix
  - `dist`: distortion coefficients
  - `imagePoints`: projected points

Camera Calibration: Calculate RMS Error

- Norm operator in OpenCV

```
error = cv2.norm(src, tgt, normType)
```

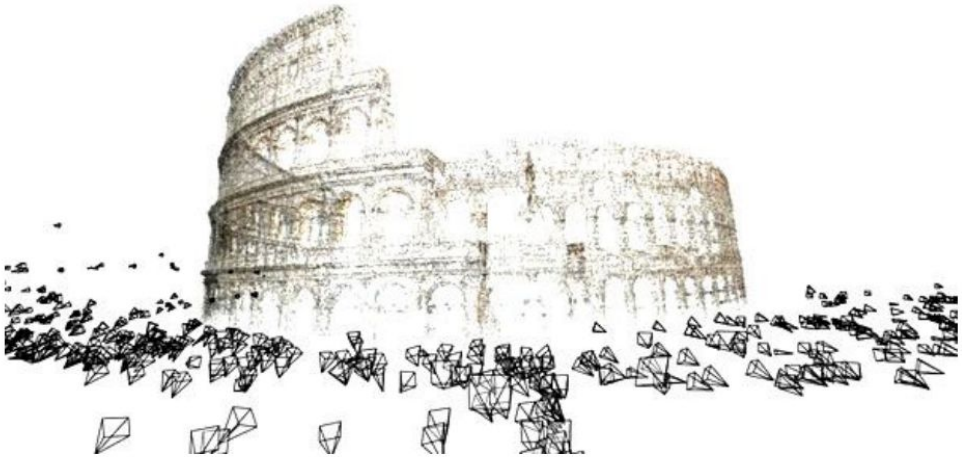
  - `src`: input array
  - `dst`: target array
  - `normType`: normalization type
    - `cv2.NORM_INF`
    - `cv2.NORM_L1`
    - `cv2.NORM_L2`

Let’s Check the Code

1_camera_calibration.ipynb

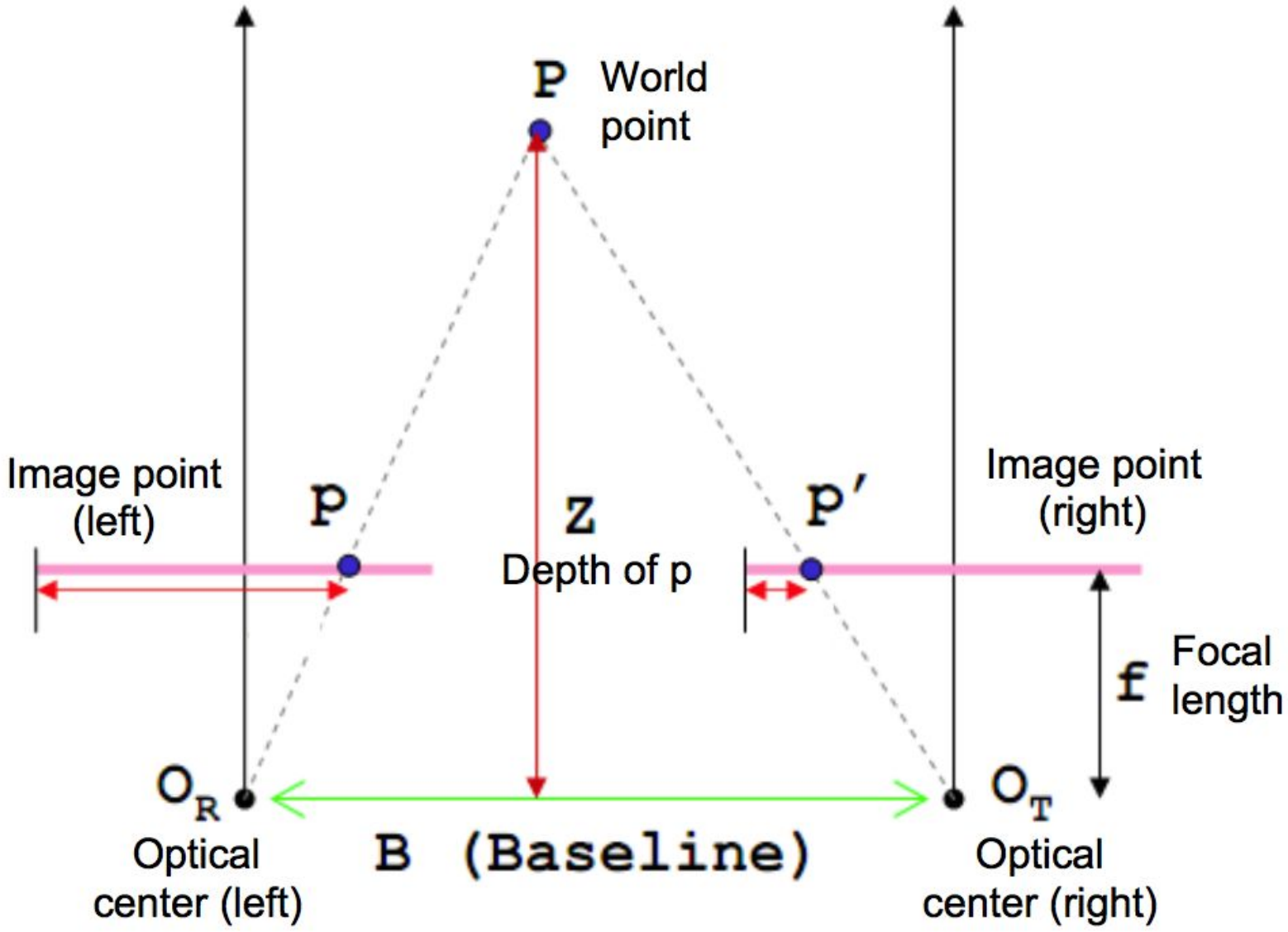
Binocular Stereo

Disparity and Depth
Window search



Stereo Camera

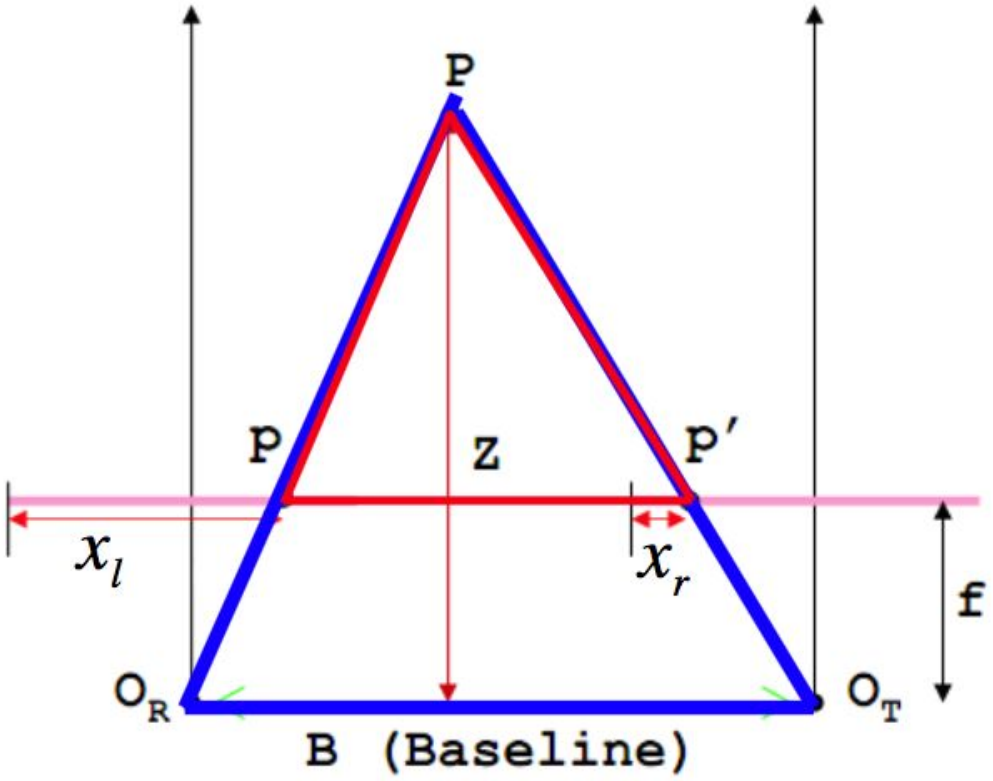
Disparity and Depth



(Slide credit: Gunhee Kim's CV lecture slide)

Disparity and Depth

Assume parallel optical axes, known camera parameters. **What is expression for Z?**



$$\frac{B + x_l - x_r}{Z - f} = \frac{B}{Z}$$

$$Z = f\frac{B}{x_r - x_l}$$

disparity

(Slide credit: Gunhee Kim's CV lecture slide)

Depth from Disparity

image I(x,y)

Disparity map D(x,y)

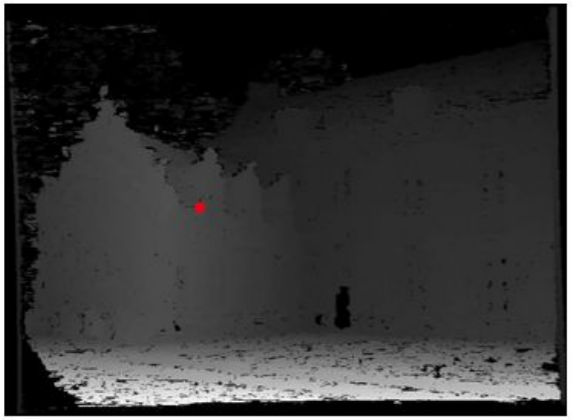


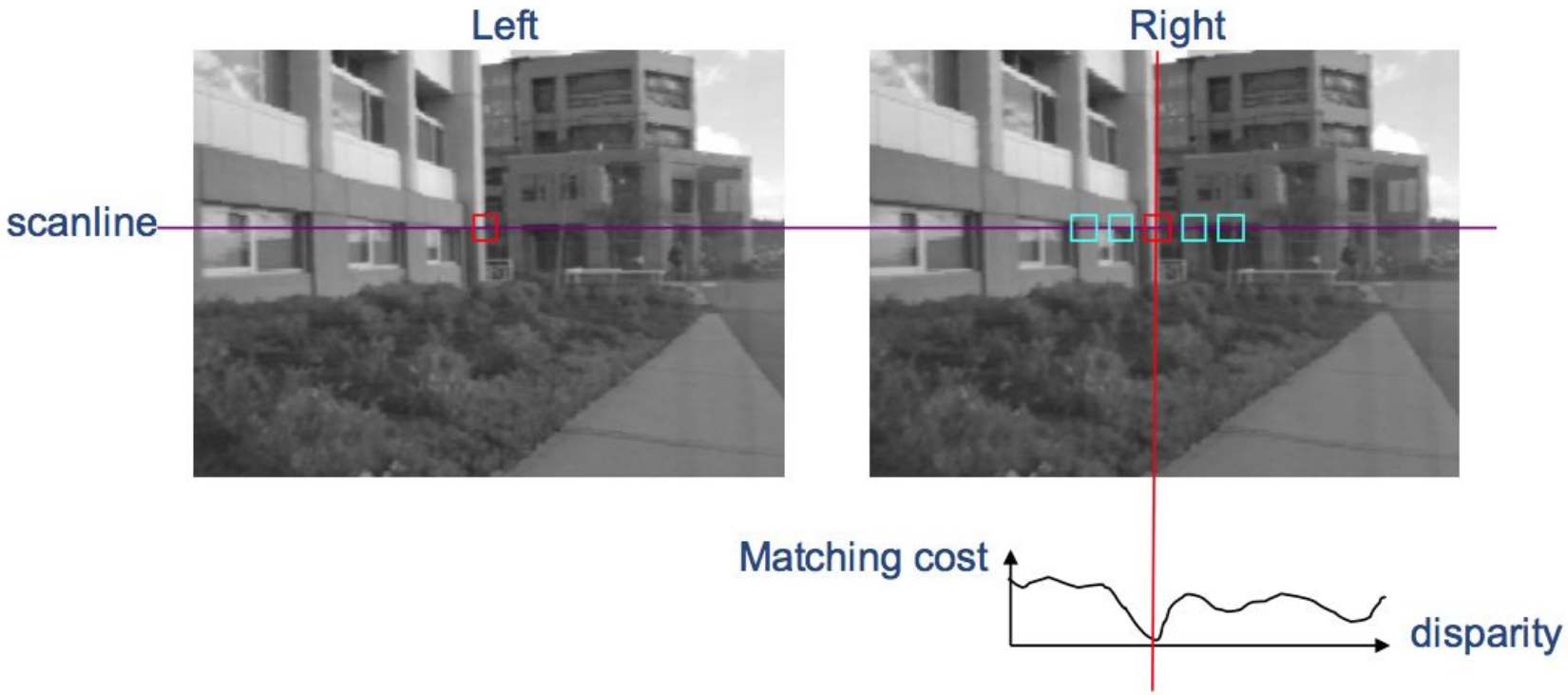
image I´(x´,y´)

$(x´,y´)=(x+D(x,y), y)$

So if we could find the **corresponding points** in two images, we could **estimate relative depth**…

(Slide credit: Lana Lazebnik)

Correspondence Search with Similarity

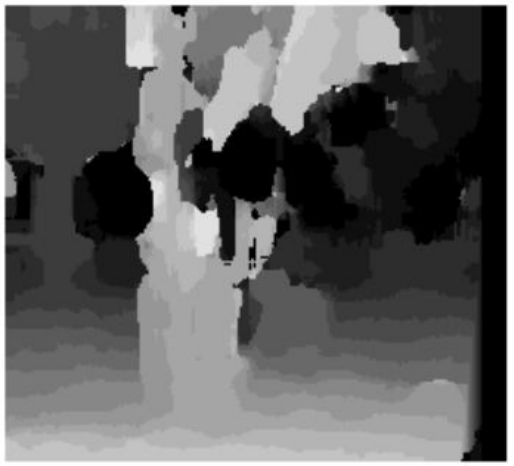


Slide a window along the right scanline and compare contents of that window with the reference window in the left image

(Slide credit: Lana Lazebnik)

Effect of Window size

W = 3



W = 20

Smaller window

+ More detail
- More noise

Larger window

+ Smoother disparity maps
- Less detail

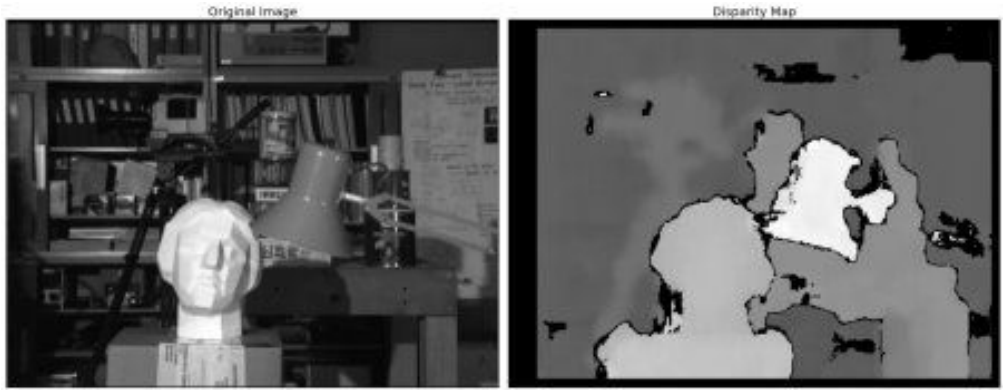
(Slide credit: Lana Lazebnik)

Create Disparity Map

- Binocular stereo operator in OpenCV

```
stereo = cv2.StereoBM_create(numDisparity, blockSize)
disparity = stereo.compute(imgL, imgR)
```

  - `numDisparity`: disparity search range
  - `blockSize`: size of blocks for window search
  - `stereo`: StereoBM object
  - `imgL`: left image
  - `imgR`: right image
  - `disparity`: output disparity of input images



Let’s Check the Code

2_binocular_stereo.ipynb

Reference

- OpenCV-Python Tutorials
  http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html