

رسالة محمد



مبانی بینایی کامپیوتر

مدرس: محمدرضا محمدی

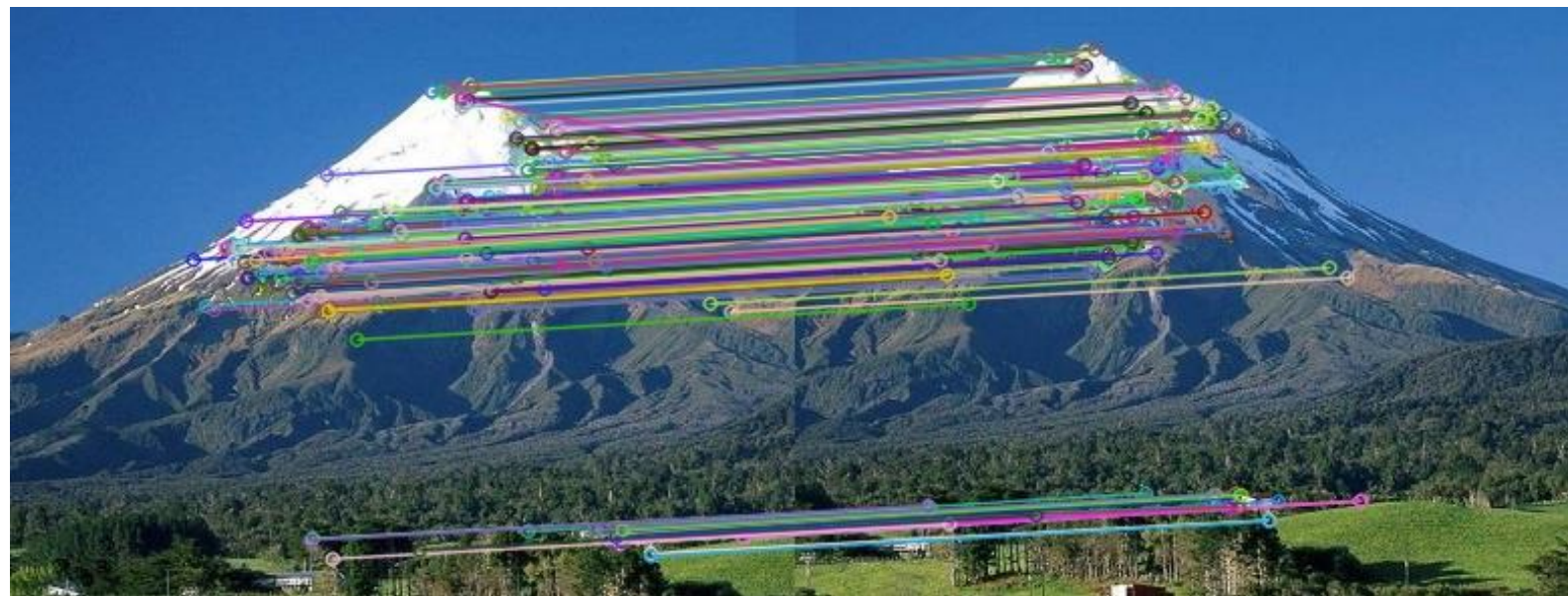
بهار ۱۴۰۳

تناظر و هم‌ترازی تصاویر

Correspondence and Image Alignment

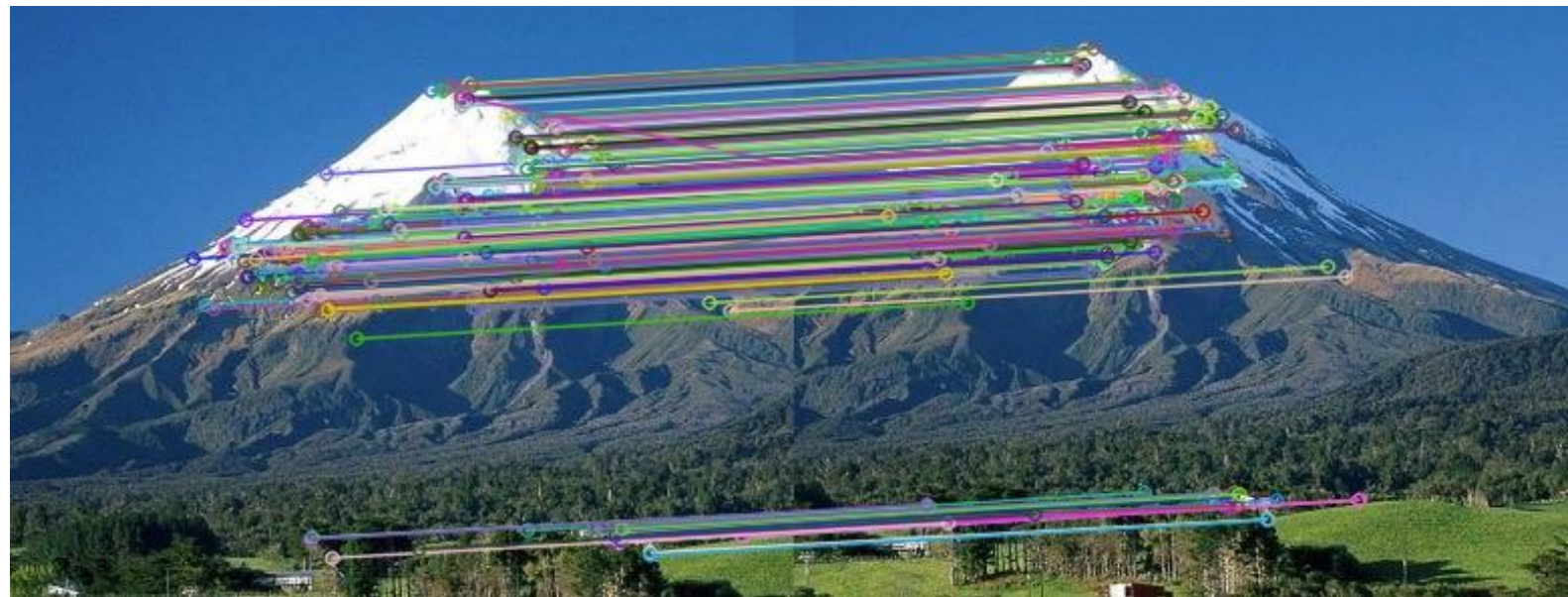
انطباق نقاط کلیدی

- پس از استخراج نقاط کلیدی از دو تصویر، نیاز است تا نقاط متناظر با یکدیگر مشخص شوند
- برای این منظور، ابتدا برای هر نقطه ویژگی یک توصیفگر محاسبه می‌شود
- سپس، دو به دو توصیفگرها از دو تصویر مقایسه می‌شوند و مشابه‌ترین توصیفگرها به عنوان نقاط متناظر انتخاب می‌شوند
- برای جلوگیری از تناظریابی اشتباه، حد آستانه‌ای بر روی میزان مشابهت گذاشته می‌شود



تابع تبدیل

- پس از یافتن نقاط متناظر، باید تابع تبدیلی را بدست آورد که نقاط تصویر اول را به نقاط تصویر دوم نگاشت کند
- برای این کار، ابتدا یک مدل برای تابع تبدیل انتخاب می‌شود و سپس پارامترهای آن بر اساس نقاط بدست آمده بهینه می‌شوند



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T \left(\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right)$$

توابع OpenCV

```
mat = cv2.getPerspectiveTransform(src_points, dst_points[, solveMethod])
```

```
// src_points:    Coordinates of quadrangle vertices in the source image
// dst_points:    Coordinates of the corresponding quadrangle vertices in the destination image
// mat:           Perspective transform from four pairs of the corresponding points
```

```
dst_points = cv2.perspectiveTransform(src_points, mat)
```

```
// src_points:    Input two-channel or three-channel floating-point array; each element is a 2D/3D vector to be transformed
// mat:           3x3 or 4x4 floating-point transformation matrix
// dst_points:    Output array of the same size and type as src
```

$$\text{dst}(x, y) = \text{src} \left(\frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}, \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \right)$$

```
dst = cv2.warpPerspective(src, mat, dsize[, flags[, borderMode[, borderValue]]])
```

```
// src_points:    Input image
// mat:           3x3 floating-point transformation matrix
// dsize:         Size of the output image
// flags:         Combination of interpolation methods and the optional flag WARP_INVERSE_MAP
// borderMode:    Pixel extrapolation method
// borderValue:   value used in case of a constant border; by default, it equals 0
// dst:           Output image that has the size dsize and the same type as src .
```

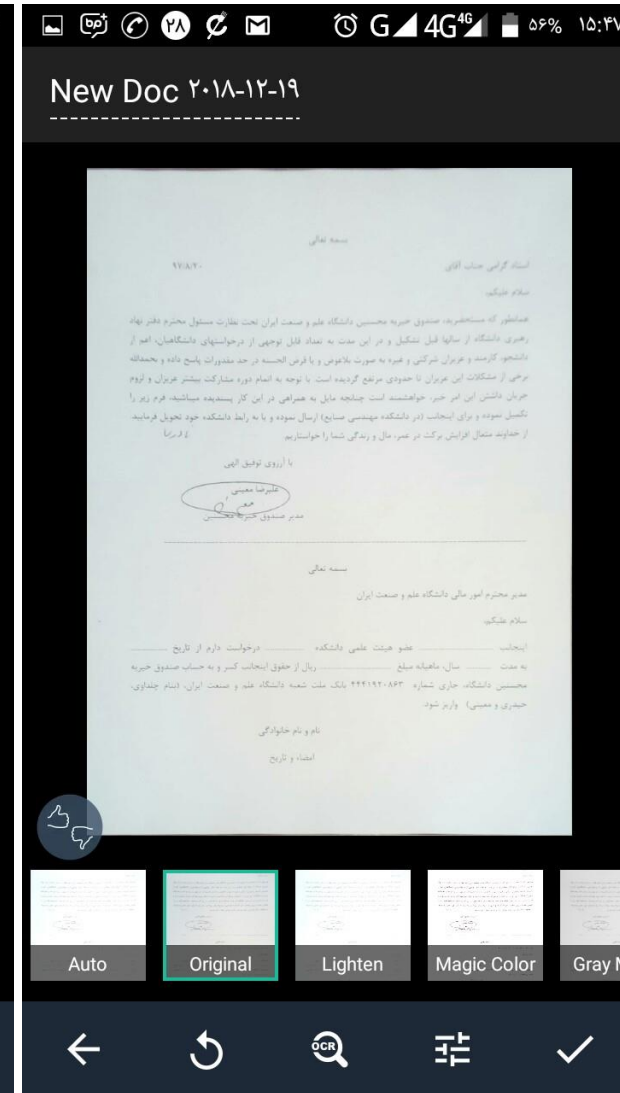
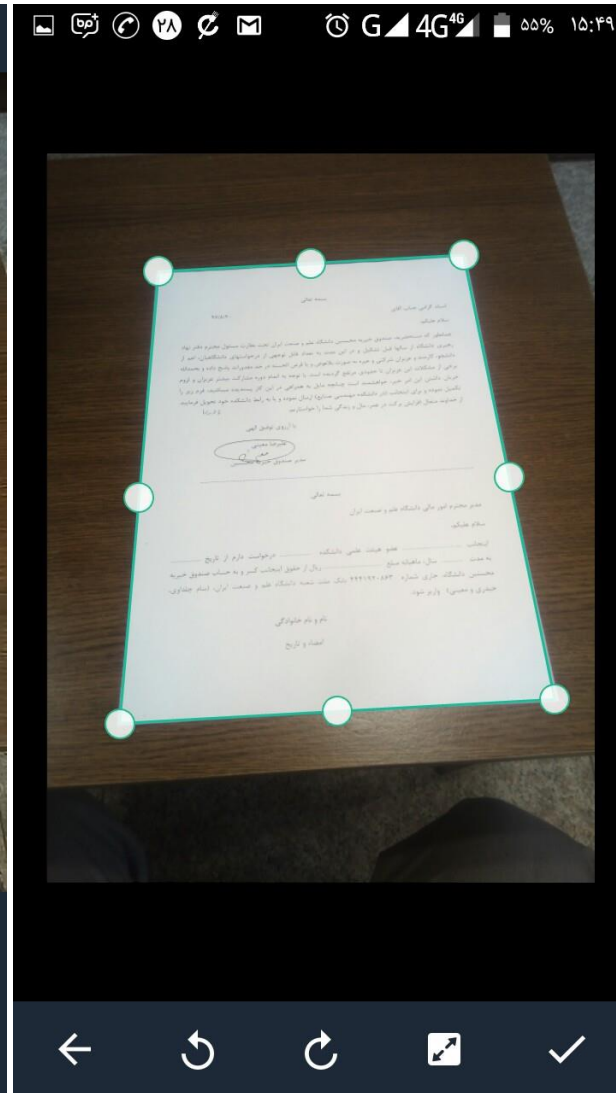
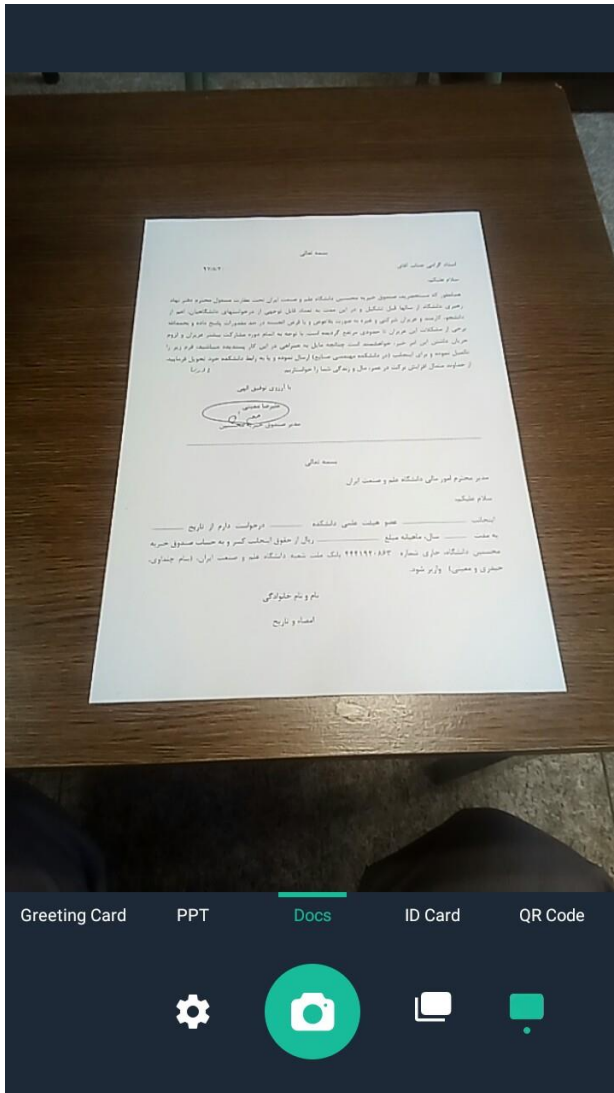

توابع OpenCV

```
mat, mask = cv2.findHomography(src_points, dst_points[, method[, ransacReprojThreshold[, maxIters[, confidence]]]])
```

```
// src_points:      Coordinates of the points in the original plane
// dst_points:      Coordinates of the points in the target plane
// method:          Method used to compute a homography matrix (least squares, RANSAC, LMEDS, RHO)
// ransacReprojThreshold: Maximum allowed reprojection error to treat a point pair as an inlier
// maxIters:        The maximum number of RANSAC iterations
// confidence:      Confidence level, between 0 and 1
// mask:            Optional output mask set by a robust method (RANSAC or LMEDS)
// mat:             Estimated perspective transform between two planes
```

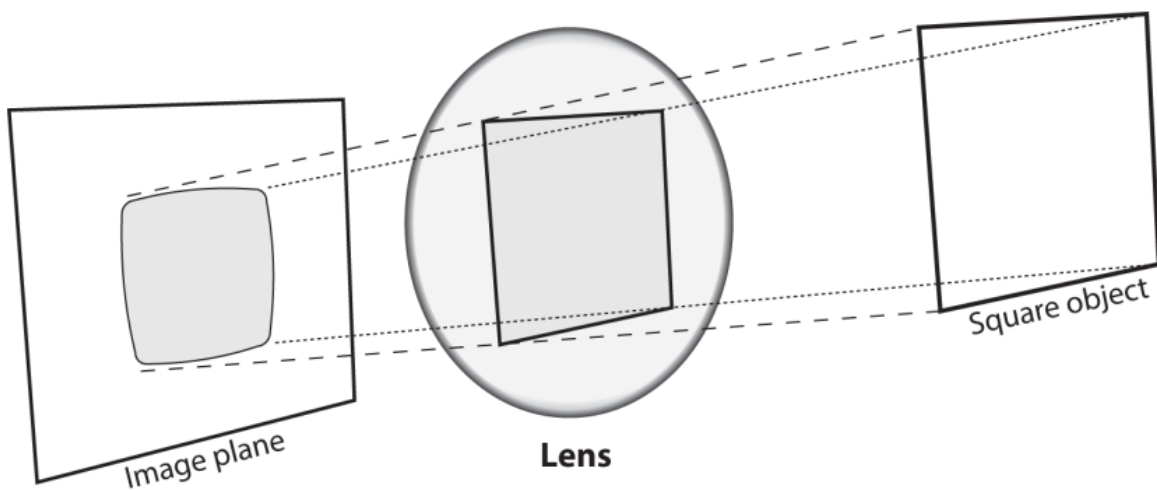
Function	Use
<code>cv::transform()</code>	Affine transform a list of points
<code>cv::warpAffine()</code>	Affine transform a whole image
<code>cv::getAffineTransform()</code>	Calculate affine matrix from points
<code>cv::getRotationMatrix2D()</code>	Calculate affine matrix to achieve rotation
<code>cv::perspectiveTransform()</code>	Perspective transform a list of points
<code>cv::warpPerspective()</code>	Perspective transform a whole image
<code>cv::getPerspectiveTransform()</code>	Fill in perspective transform matrix parameters

CamScanner



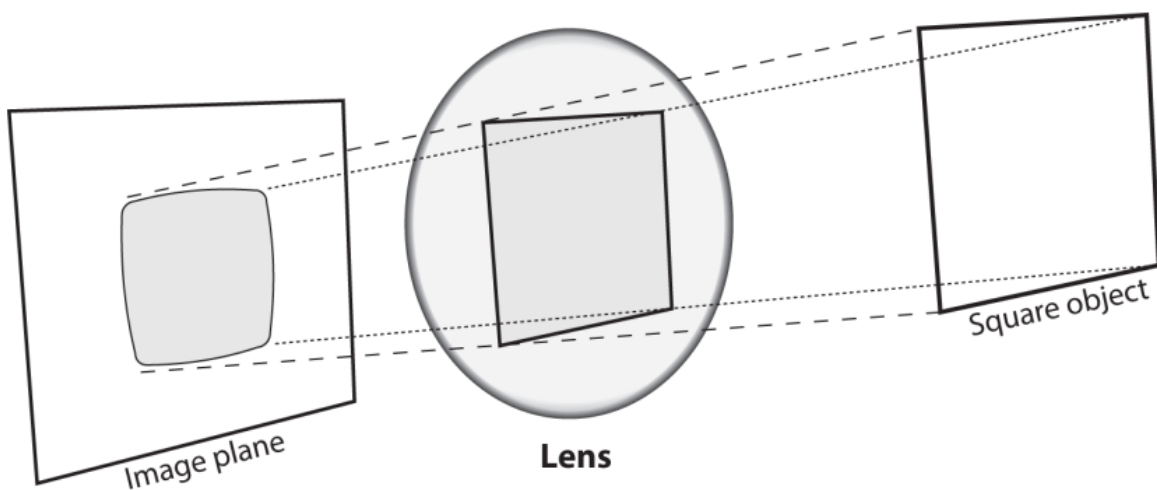
اعوجاج لنز (Lens Distortion)

- در عمل، هیچ لنز ساخته شده‌ای ایده‌آل نیست
- دو نوع اعوجاج اصلی وجود دارد:
 - اعوجاج شعاعی (Radial) که حاصل از شکل لنز است
 - اعوجاج مماسی (Tangential) که حاصل از فرآیند سوار کردن دوربین است



اعوجاج شعاعی

- به خصوص در لنزهای ارزان قیمت این اعوجاج به سادگی قابل تشخیص است
- اعوجاج در مرکز نوری ۰ است و با حرکت به سمت مرز تصویر افزایش می یابد
- این اعوجاج نسبت به مرکز متقارن فرض می شود و با استفاده از چند عبارت از سری تیلور می توان آن را مدل کرد



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

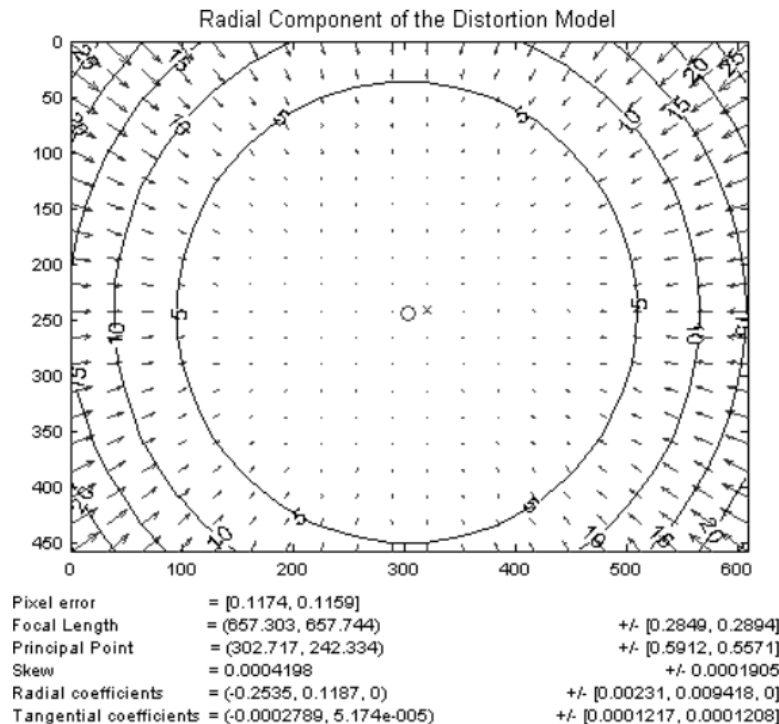
$$y_{corrected} = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

اعوجاج شعاعی

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

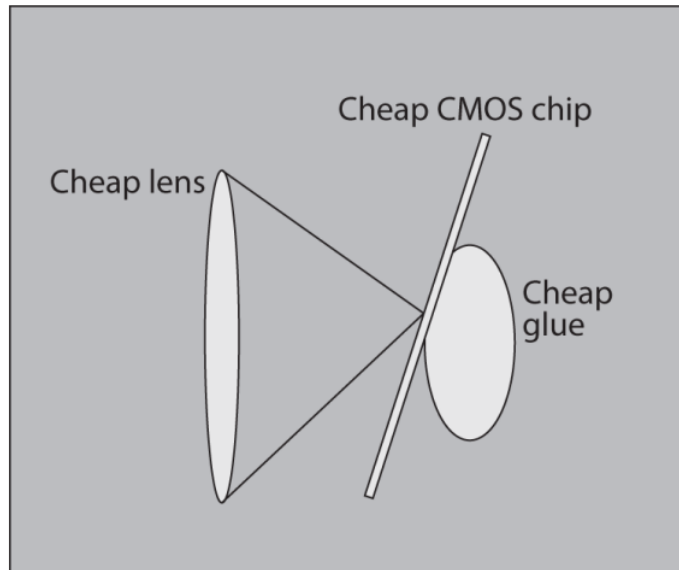
$$y_{corrected} = y \cdot (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

- که (x, y) مختصات اولیه برای نقطه دچار اعوجاج شده است و $(x_{corrected}, y_{corrected})$ مختصات جدید آن بعد از اصلاح است
- برای دوربین‌هایی که اعوجاج زیادی ندارند، می‌توان از عبارت سوم صرف نظر کرد ($k_3 = 0$)

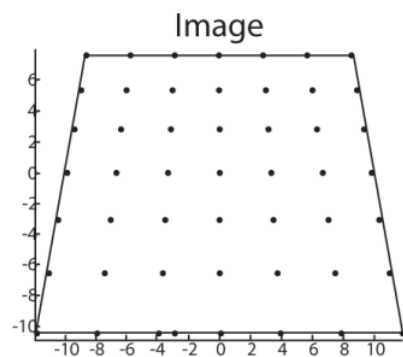


اعوجاج مماسی

- این اعوجاج در فرآیند ساخت دوربین ایجاد می‌شود و به دلیل موازی نبودن دقیق صفحه تصویربرداری با لنز است
- معمولاً به صورت زیر مدل می‌شود



Cheap camera



$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

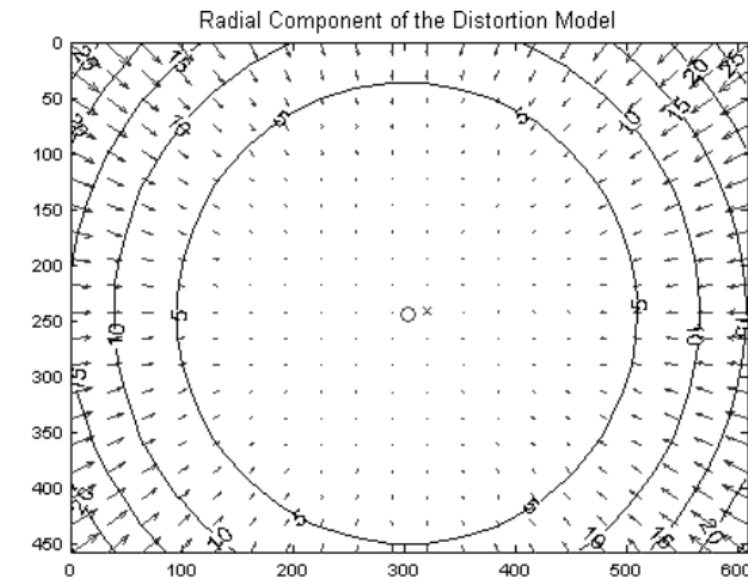
اعوجاج شعاعی و مماسی

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

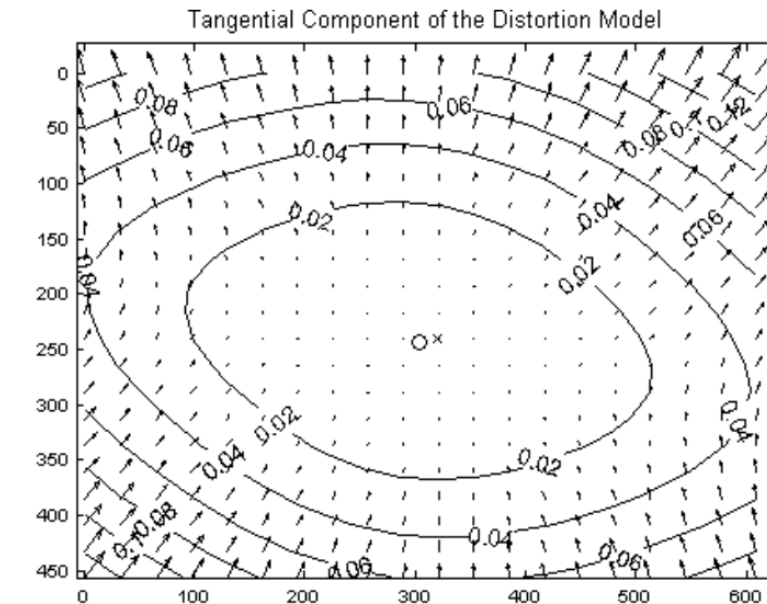
$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$

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

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



Pixel error = [0.1174, 0.1159]
 Focal Length = (657.303, 657.744) +/- [0.2849, 0.2894]
 Principal Point = (302.717, 242.334) +/- [0.5912, 0.5571]
 Skew = 0.0004198 +/- 0.0001905
 Radial coefficients = (-0.2535, 0.1187, 0) +/- [0.00231, 0.009418, 0]
 Tangential coefficients = (-0.0002789, 5.174e-005) +/- [0.0001217, 0.0001208]



Pixel error = [0.1174, 0.1159]
 Focal Length = (657.303, 657.744) +/- [0.2849, 0.2894]
 Principal Point = (302.717, 242.334) +/- [0.5912, 0.5571]
 Skew = 0.0004198 +/- 0.0001905
 Radial coefficients = (-0.2535, 0.1187, 0) +/- [0.00231, 0.009418, 0]
 Tangential coefficients = (-0.0002789, 5.174e-005) +/- [0.0001217, 0.0001208]

• در مجموع ۵ پارامتر دارند:

k_3, p_2, p_1, k_2, k_1 -

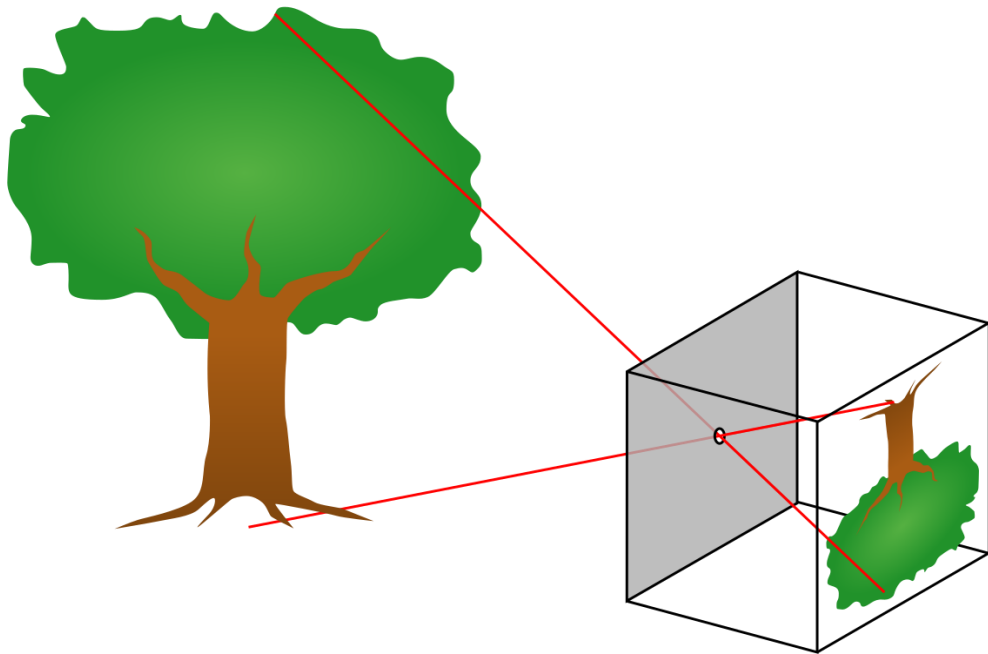
• در OpenCV:

Distortion coefficients

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

مدل دوربین Pinhole

- در محل تشکیل تصویر، با استفاده از یک حسگر نوری، تصویر ثبت می‌شود
- در عمل، مرکز حسگر تصویربرداری دقیقا منطبق بر محور نوری نیست
- همچنین، پیکسل‌ها دارای عرض و طول کاملاً یکسانی نیستند



$$x_{screen} = f_x \frac{X}{Z} + c_x$$

$$y_{screen} = f_y \frac{Y}{Z} + c_y$$

$$f_x = f s_x, f_y = f s_y$$

ماتریس داخلی (intrinsic) دوربین

- تبدیل افکنش (projective transform) به رابطه‌ای گفته می‌شود که یک نقطه \vec{Q} در دنیای فیزیکی را به مختصات صفحه تصویر (x,y) تبدیل می‌کند
 - برای محاسبات به طور معمول از مختصات همگن (homogeneous coordinates) استفاده می‌شود که در آن یک بعد اضافه می‌شود و مقادیر به صورت نسبی هستند
- M ماتریس داخلی نامیده می‌شود

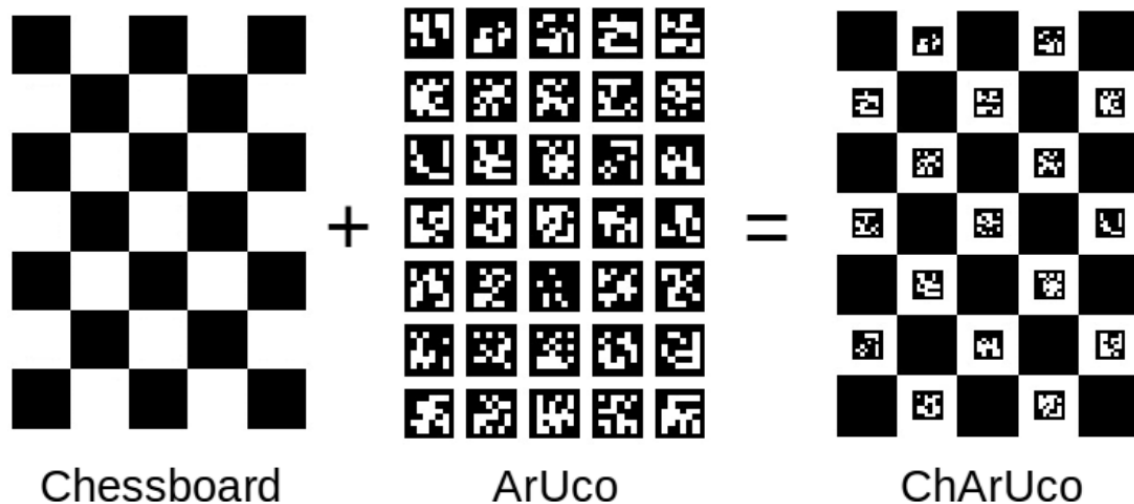
$$x_{screen} = f_x \frac{X}{Z} + c_x$$

$$\vec{q} = M \cdot \vec{Q}$$

$$\vec{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \vec{q} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

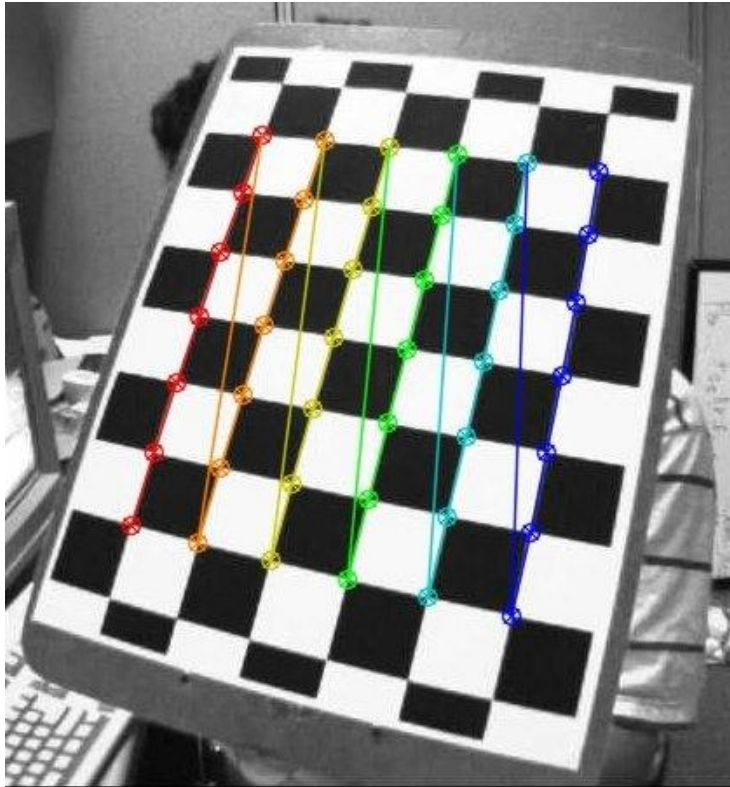
Calibration

- می‌خواهیم پارامترهای مربوط به دوربین و لنز را تخمین بزنیم
- برای این کار، یک الگوی دارای ساختار مشخص که تعداد زیادی نقطه متمایز و قابل تشخیص داشته باشد را روبروی دوربین قرار می‌دهیم
- با مشاهده این ساختار از زوایای مختلف، می‌توانیم موقعیت نسبی دوربین در بین فریم‌ها و همچنین پارامترهای دوربین و لنز را محاسبه کنیم



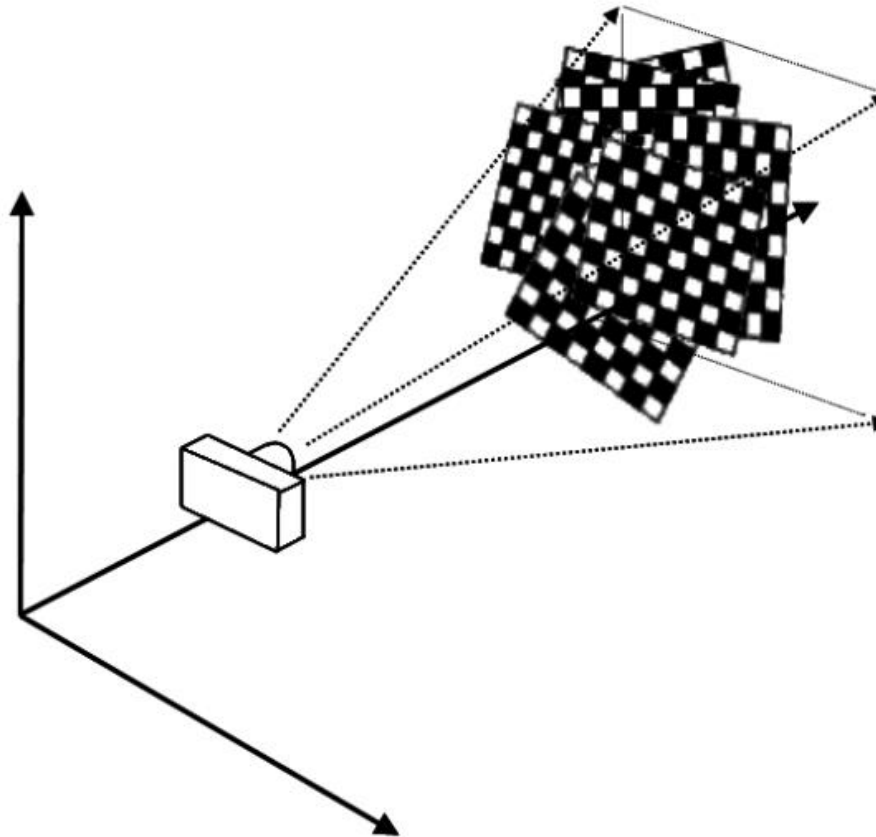
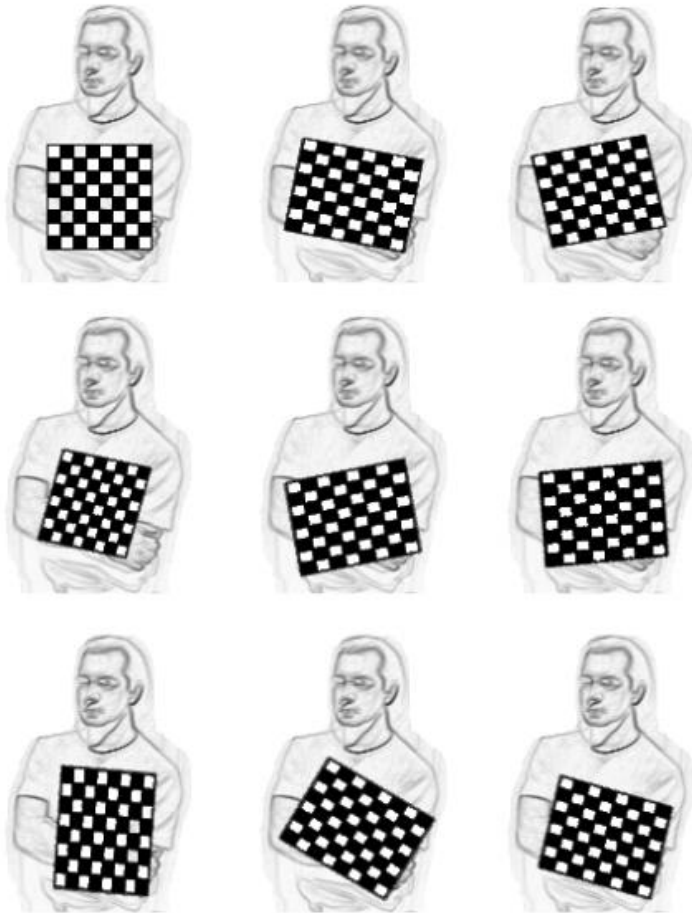
Calibration

- با استفاده از تکنیک‌های ساده پردازش تصویر، می‌توان موقعیت گوشه‌های صفحه شطرنجی در تصویر دوبعدی را مشخص کرد



- اگر موقعیت این نقاط در دنیای فیزیکی سه‌بعدی را بدانیم، می‌توانیم $4+5$ پارامتر را از متناظر قرار دادن نقاط تخمین بزنیم
- اما موقعیت سه‌بعدی این نقاط را نمی‌دانیم
- می‌توانیم چندین تصویر در شرایط مختلف تهیه کنیم تا از ارتباط میان آنها بتوانیم پارامترها را تخمین بزنیم

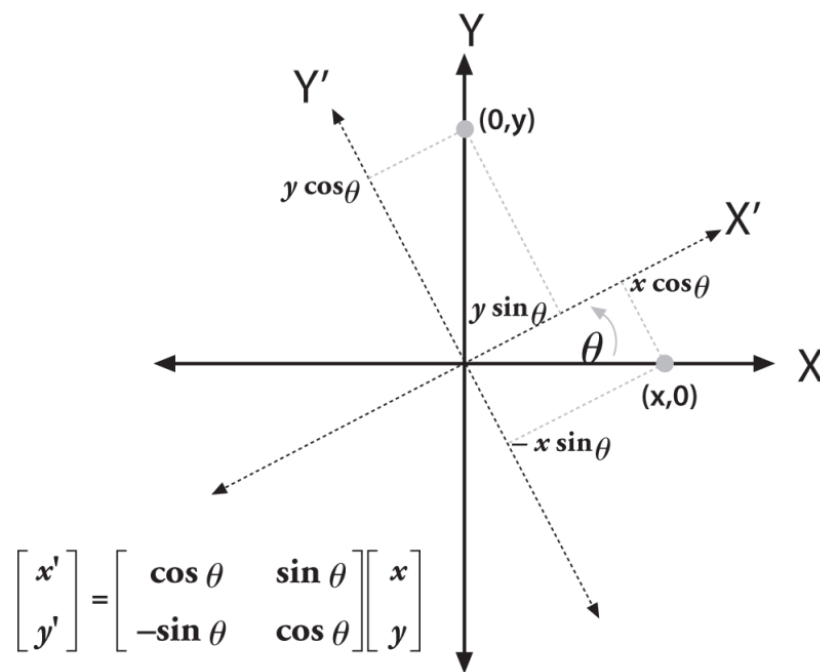
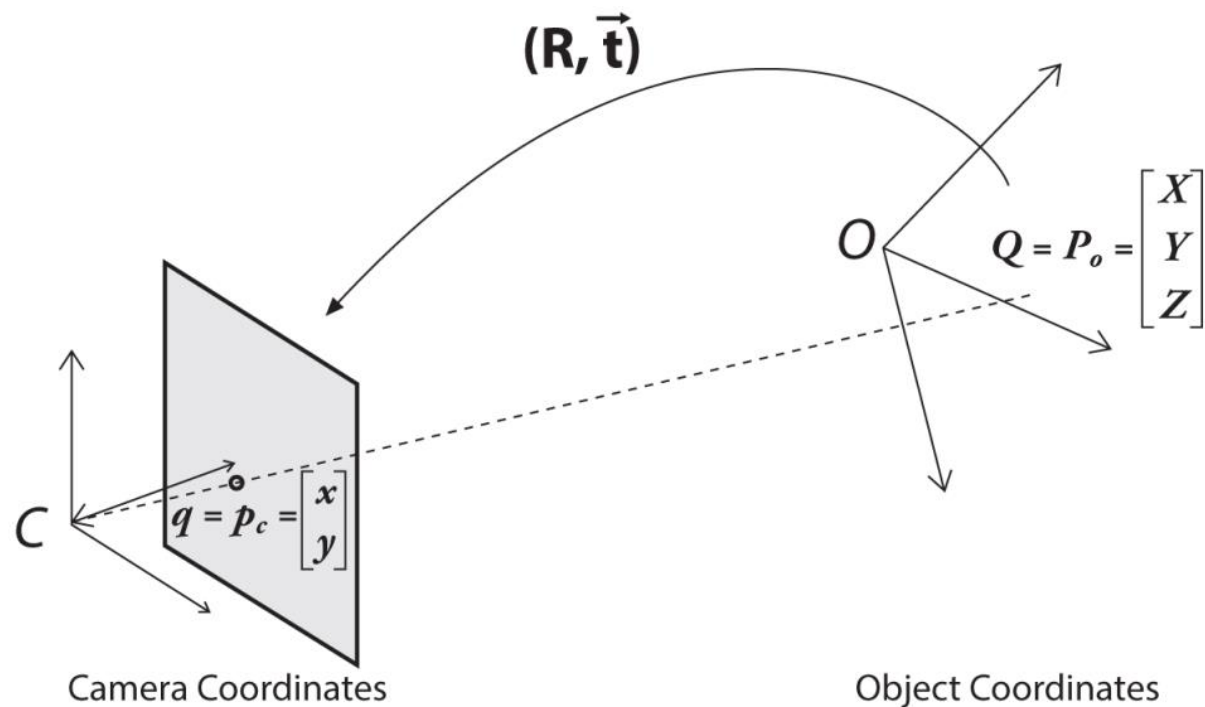
Calibration



ماتریس چرخش و بردار جابجایی

- برای هر تصویر که دوربین از یک شیء خاص گرفته است، می‌توان pose شیء نسبت به دستگاه مختصات دوربین را با استفاده از چرخش و جابجایی بیان کرد

- ماتریس چرخش دوبعدی:



$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

ماتریس چرخش

- چرخش در سه بعد، می تواند به صورت سه چرخش دوبعدی نسبت به محورهاى مختلف تجزیه شود
- ماتریس چرخش نهایی از ضرب سه ماتریس قابل محاسبه است

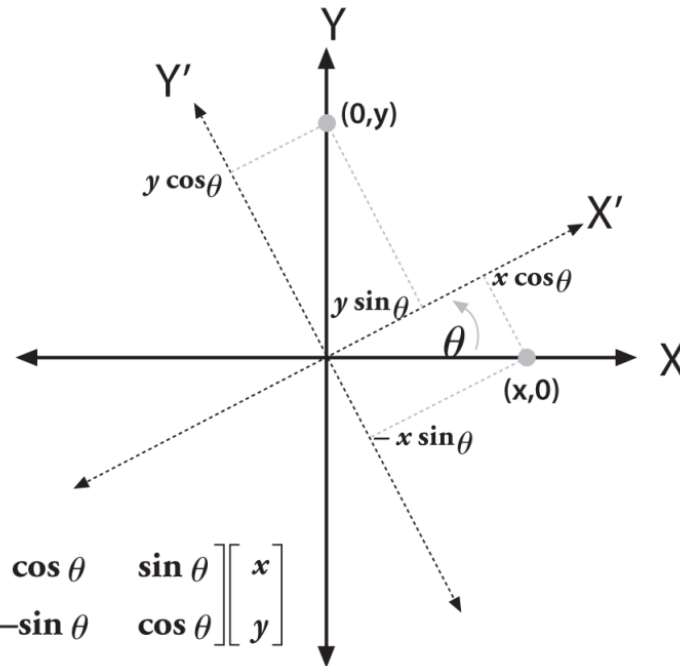
- ماتریس چرخش خواصی دارد از جمله آنکه معکوس آن برابر با ترانهاده آن است

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & \sin\psi \\ 0 & -\sin\psi & \cos\psi \end{bmatrix}$$

$$R_y(\psi) = \begin{bmatrix} \cos\varphi & 0 & -\sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\varphi & 0 & \cos\varphi & 0 \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$



$$R = R_x(\psi) \cdot R_y(\varphi) \cdot R_z(\theta)$$

$$R \cdot R^T = I$$

Calibration

- بردار جابجایی برابر با اختلاف مرکز دستگاه مختصات دوربین و شیء است

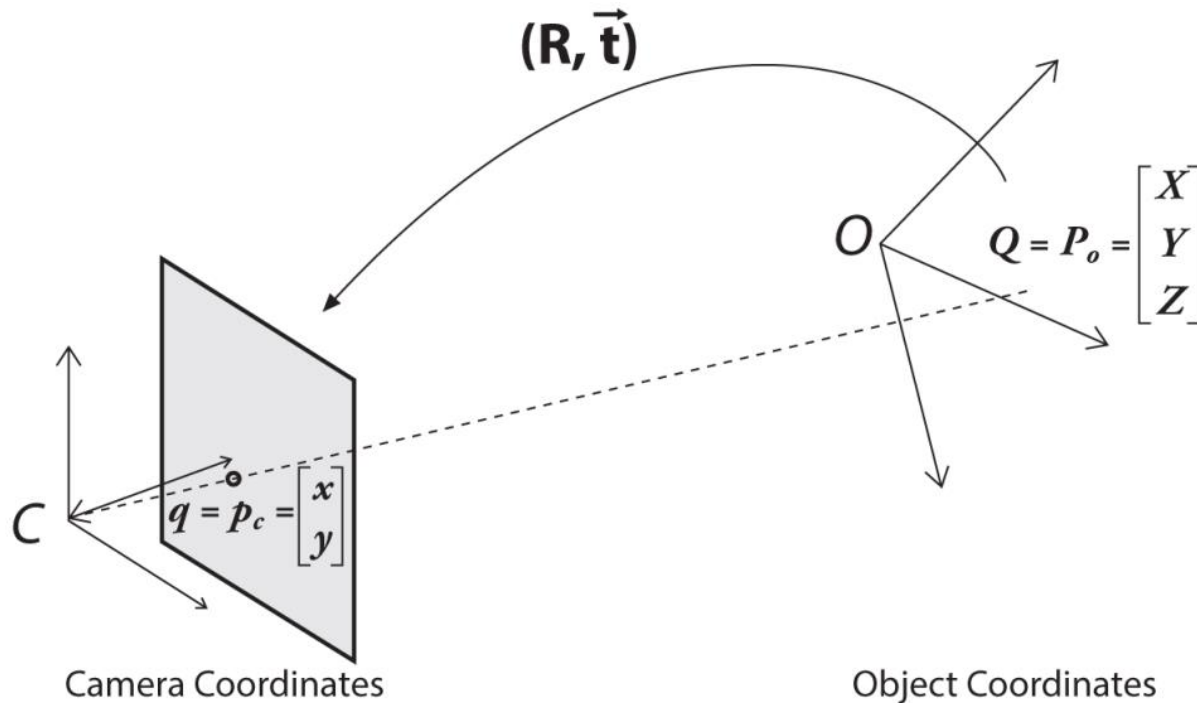
$$\vec{t} = origin_{object} - origin_{camera}$$

- یک نقطه در دستگاه مختصات شیء \vec{P}_o با رابطه زیر به دستگاه مختصات دوربین \vec{P}_c قابل تبدیل است

$$\vec{P}_c = R \cdot (\vec{P}_o - \vec{t})$$

- این رابطه ۶ مجهول دارد

- حداقل به دو زاویه دید برای کالیبراسیون نیاز است

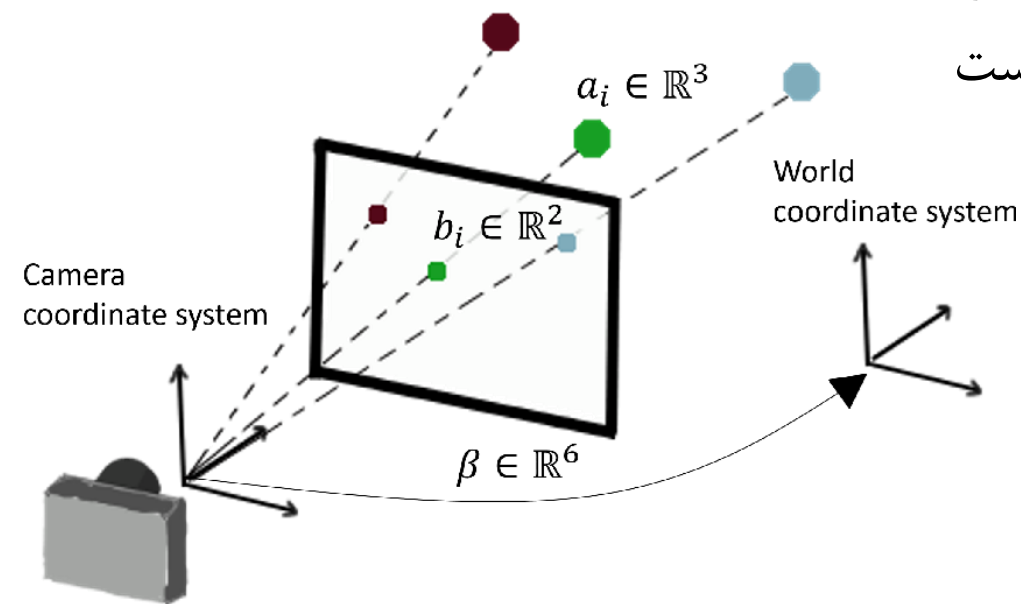


بینایی سه بعدی

Three-Dimensional Vision

محاسبه پارامترهای خارجی (extrinsics)

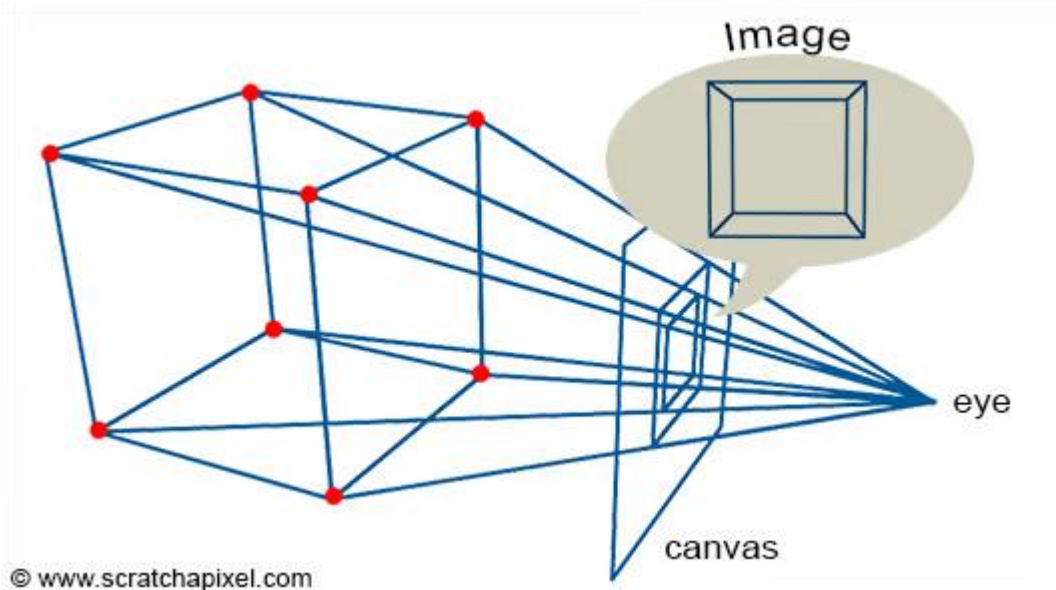
- با انجام کالیبراسیون، پارامترهای داخلی دوربین (camera intrinsics matrix و distortion coefficients) و پارامترهای خارجی دوربین (rotation and translation) تخمین زده می‌شوند
- فرض کنید پارامترهای داخلی دوربین را داشته باشیم، چطور می‌توانیم موقعیت اشیاء را محاسبه کنیم؟
 - به صورت عمومی، به این کار Perspective n-Points (PnP) گفته می‌شود
 - در حالت عادی حداقل به ۳ نقطه برای تخمین این ۶ پارامتر نیاز است



```
bool cv::solvePnP(
    cv::InputArray  objectPoints,           // Object points (object frame)
    cv::InputArray  imagePoints,           // Found pt locations (img frame)
    cv::InputArray  cameraMatrix,          // 3-by-3 camera matrix
    cv::InputArray  distCoeffs,            // Vector of 4, 5, or 8 coeffs
    cv::OutputArray rvec,                  // Result rotation vector
    cv::OutputArray tvec,                  // Result translation vector
    bool            useExtrinsicGuess = false, // true='use vals in rvec and tvec'
    int             flags              = cv::ITERATIVE
);
```

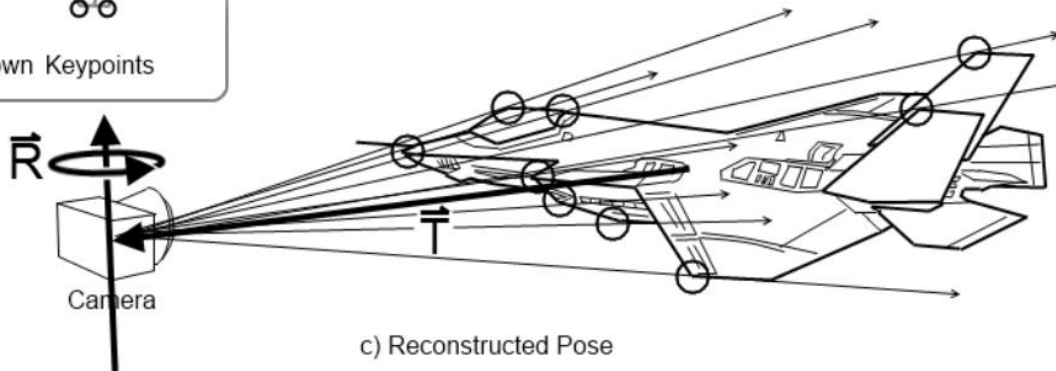
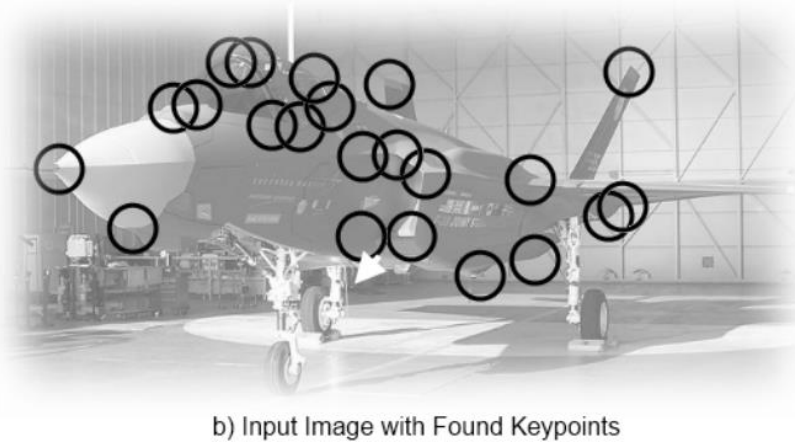
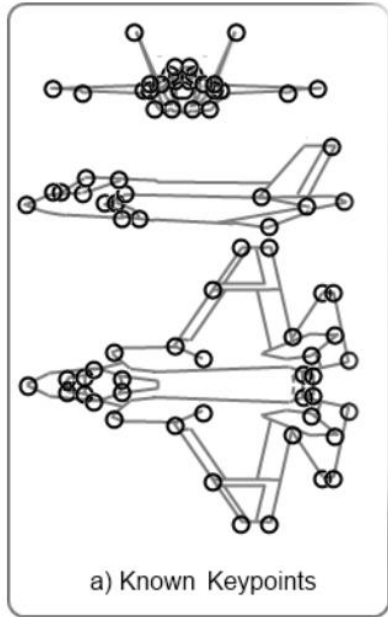
افکنش (Projection)

- هنگامی که دوربین کالیبره شد، می‌توان هر نقطه در دنیای فیزیکی را بدون ابهام به نقطه متناظر در تصویر نگاشت کرد
- اما رابطه معکوس برقرار نیست و نمی‌توان مختصات سه‌بعدی متناظر با یک پیکسل را به سادگی محاسبه کرد



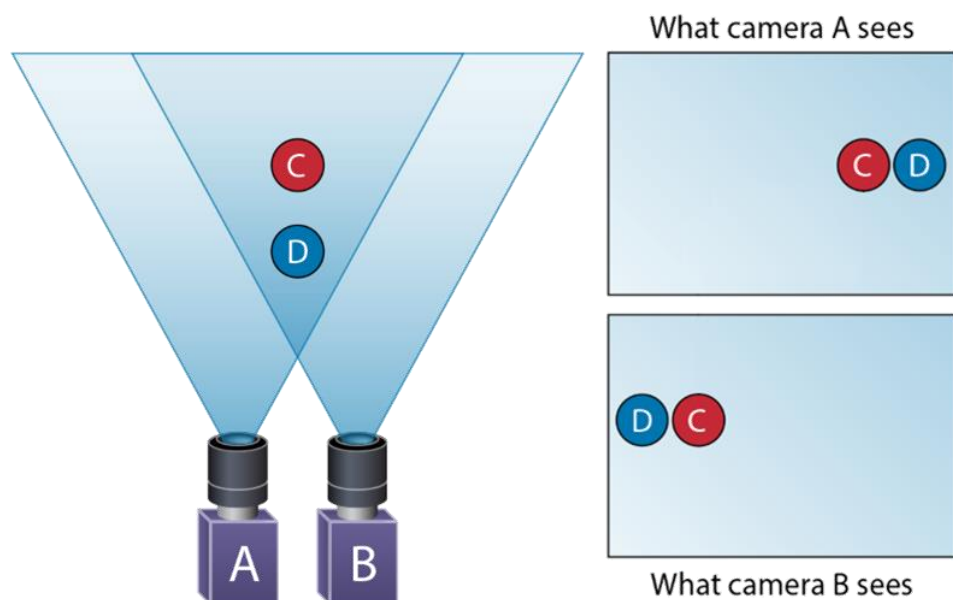
تخمین Pose از یک دوربین

- اگر شیء مورد بررسی شناخته شده باشد، می‌توان تعدادی نقطه کلیدی (keypoint) روی آن مشخص کرد
- سپس، موقعیت این نقاط کلیدی در تصویر جدید را مشخص کرد
- با استفاده از PnP، موقعیت شیء و عمق نقاط آن را محاسبه کرد



تصویربرداری Stereo

- با استفاده از دو دوربین، می‌توان از دو زاویه دید متفاوت اشیاء را مشاهده کرد و از تفاوت آنها، عمق را تخمین زد
- در این مثال، اشیائی که دورتر باشند، در دو تصویر فاصله کمتری خواهند داشت



مثلبندی

• ابتدا یک سیستم استریوی ایده‌آل را در نظر بگیرید

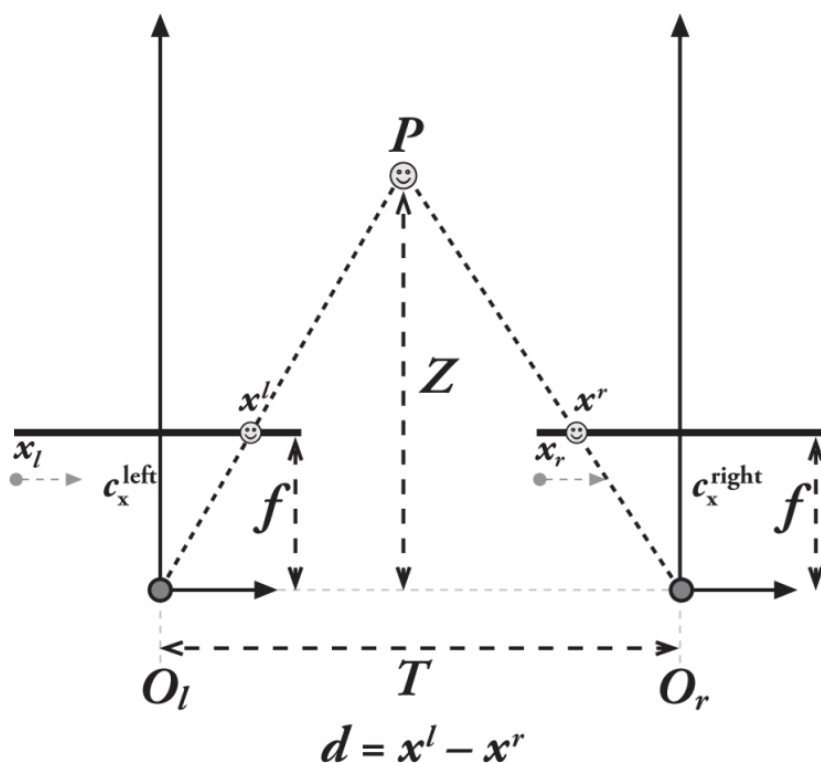
- فرض کنید به طور کامل اعوجاج تصاویر حذف شده است و با هم تراز شده‌اند و صفحه‌های تصاویر دقیقا با یکدیگر هم‌سطح هستند

- با محورهای نوری کاملا موازی

- فاصله معلوم (T)


- فاصله‌های کانونی یکسان ($f_l = f_r$)

- مختصات پیکسلی یکسان برای c_x^{right} و c_x^{left}

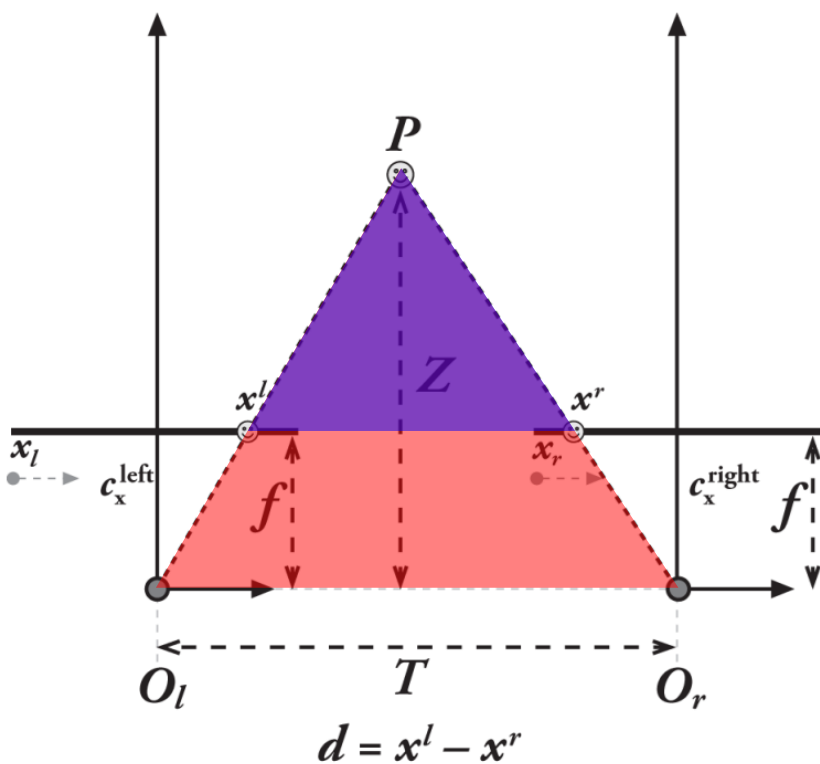


مثلت بندی

- فرض کنید نقطه P در دنیای فیزیکی، در دو دوربین چپ و راست به ترتیب در مختصات p_r و p_l دیده شود که مختصات افقی آنها x_r و x_l باشد
- از تشابه دو مثلث داریم:



$T = T = (x_r - x_l)$



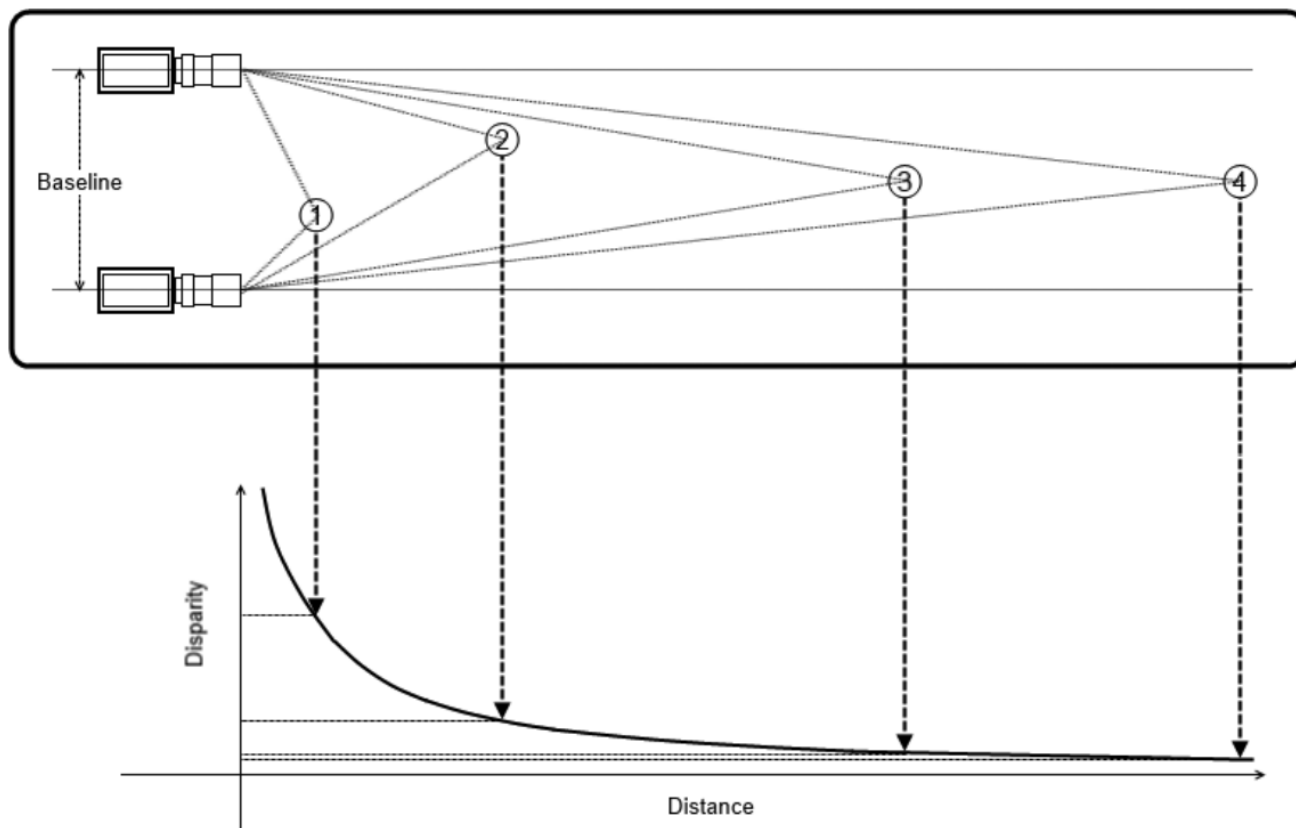
$$\frac{T}{Z} = \frac{T - (x_l - x_r)}{Z - f}$$

$$Z T - f T = Z T - Z (x_l - x_r)$$

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

مثلبندی

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



- عمق (Z) رابطه معکوس با disparity ($x_l - x_r$) دارد
- زمانی که disparity نزدیک به صفر است، تفاوت کوچک در آن منجر به تفاوت بسیار زیاد در عمق می شود
- سیستم های بینایی استریو دارای رزولوشن عمق زیاد تنها برای اشیائی هستند که در فاصله نزدیکی از دوربین قرار دارند