Valet Parking Robot 개발

카메라 캘리브래이션

영상처리2팀

2019. 9. 27

ADAS ONE

목차

카메라 캘리브레이션

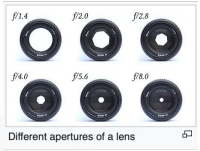
카메라캘리브레이션이란 , 카메라 센서의 “파라미터”의 알맞은 값을 찾고, 그 값의 정확한지 평가하는 작업

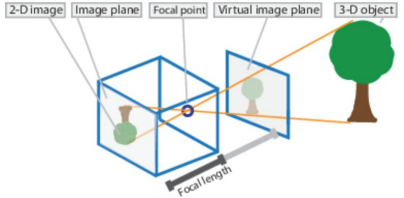
입체적인 현실을 평면에 투사할 때, 투사될 좌표를 추정하는 특별한 방정식이 존재함.
특정한 조건하에 사진을 많이 찍으면, 거기서 영상좌표&물리좌표를 쌍을 얻어, 그 방정식을 품.
방정식내의 미지수는 카메라의 스펙과 설치위치를 의미함. 필요한 미지수의 개수와 종류는 불변.
그 미지수의 해를, 변환함수의 인자로 보아, 카메라 파라미터, 짧게는 파라미터라고 부름.

카메라의 몇 픽셀이 움직이면, 현실에서 몇 센치 움직인지 계산이 가능해, 여러 응용이 가능함.
렌즈는 둥그래서 사진은 의외로 렌즈를 중심으로 휘었는데, 이를 반듯이 핀다거나 (dist
비스듬하게 찍은 것을, 반듯이 나타내거나 (image warp)
픽셀을 계산해 찍힌 사물의 길이를 측정함 (measurements)

Remove Lens Distortion

카메라 투영

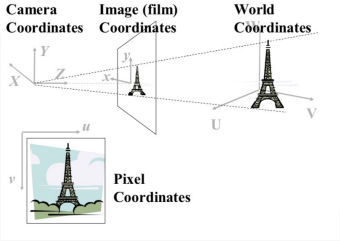


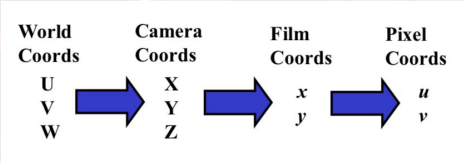


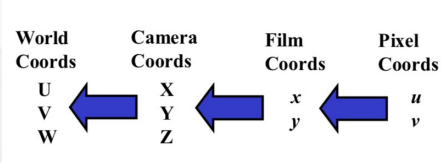
Pinhole 카메라 : 작은 조리개를 가진 카메라

카메라 모델 : pinhole 모델, 3차원 공간의 모든 빛은 바늘구멍을 지나 2차원 평면에 투영

카메라 투영 : 삼차원 공간상의 점들을 이미지평면에 투영.







Forward projection : 3차원 월드 좌표들이 어디로 투사되는지 나타내는 수학식 찾기
Backward projection : 이를 거꾸로하여, 2D에서 3D복원하기. 주된 관심사.

사영 행렬 : 전산학 관점에선, 행렬변환의 순열을 하나의 큰 행렬로 나타내는 것

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

$$= A[R \mid t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (1)$$

영상기하학

영상 기하학 : 좌표계간 불변의 관계를 다루는 수학 공식들. 닮음비 응용

좌표계 : 발명품 → 관습적 정의.

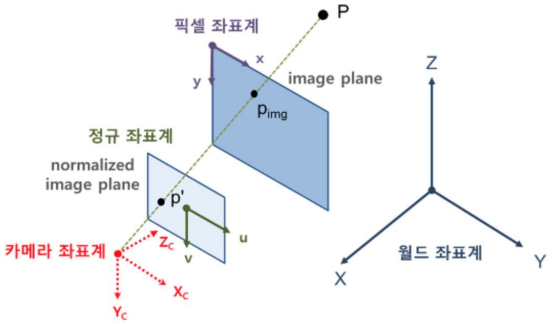
World 좌표계 : UVW, 현실의 좌표계로 사용자가 지정하기 나름

카메라 좌표계 : XYZ, 빛이 들어오는 광학축이 z축임

정규화 좌표계 : uv, 원래 이미지를 초점거리가 1인 지점으로 옮긴 것, 내부파라미터 무효화

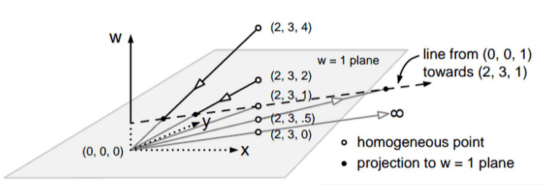
사진 좌표계 : xy, 이미지

픽셀 좌표계 : px, py 픽셀로 바꾼 것.



동차좌표 : 무한직선의 방향을 표현하는 법. 가중치로 나누어 좌표를 표

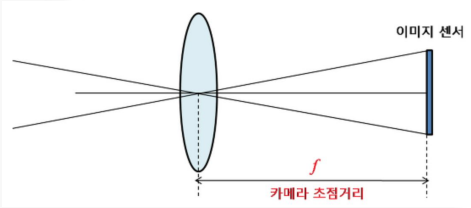
$$(x, y, w) \sim (x/w, y/w)$$



카메라 파라미터

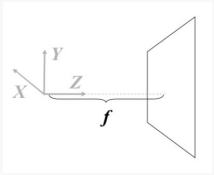
내부파라미터 : 카메라의 센싱부품의 내재된 변수

초점거리 fx, fy
주점거리 cx, cy
비대칭 계수 skew



왜곡계수 : 이차원 평면의 비선형 변환

방사왜곡계수 k1, k2, (k3)
접선왜곡계수 p1, p2



외부파라미터 : 카메라와 외부환경과의 관계

평행이동 x, y, z
회전 roll, pitch, yaw

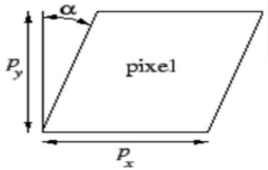


그림 5. 카메라 비대칭 계수

내부 파라미터

내부파라미터 : 카메라의 센싱부품의 내재된 변수

초점거리 f : 렌즈 중심 → 센서부, 픽셀 표현

초점거리 fx, fy : f의 길이가 가로크기/세로크기의 몇 배

주점거리 cx, cy : 광학축과 영상 평면이 만나는 점

비대칭 계수 skew : y축의 기운 정도

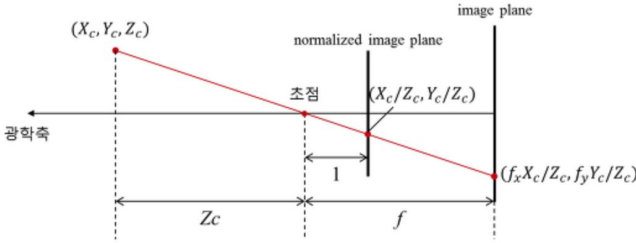
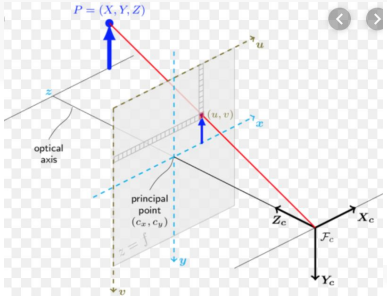


그림 4. 카메라 투영(projection) 모델



$$p_{img} = Kp' \quad \text{--- (5)}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \text{--- (6)}$$

p'=(u,v)

왜곡계수 보정

Backward Mapping

$$\begin{aligned} v' &= Av, \\ A^{-1}v' &= A^{-1}Av, \\ A^{-1}v' &= v. \end{aligned}$$

구해야하는 V'에서 V를 구함
코드 상 Mat output에 따라 Mat input 처리

normalized_undistorted : 내부파라미터 무효화

$$\begin{bmatrix} x_{n\_u} \\ y_{n\_u} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} x_{p\_u} \\ y_{p\_u} \\ 1 \end{bmatrix}$$

normalized_distorted : 왜곡계수, 왜곡모델에 적용

$$\begin{bmatrix} x_{n\_d} \\ y_{n\_d} \end{bmatrix} = (1 + k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6) \begin{bmatrix} x_{n\_u} \\ y_{n\_u} \end{bmatrix} + \begin{bmatrix} 2p_1 x_{n\_u} y_{n\_u} + p_2 (r_u^2 + 2x_{n\_u}^2) \\ p_1 (r_u^2 + 2y_{n\_u}^2) + 2p_2 x_{n\_u} y_{n\_u} \end{bmatrix}$$

r_u ($r_u{}^2 = x_{n\_u}{}^2 + y_{n\_u}{}^2$)

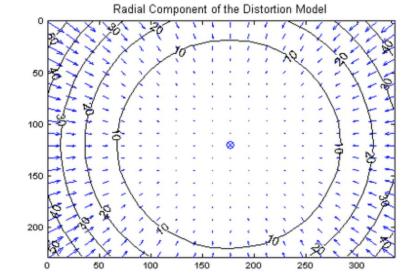
pixel_distorted : 내부파라미터 재적용

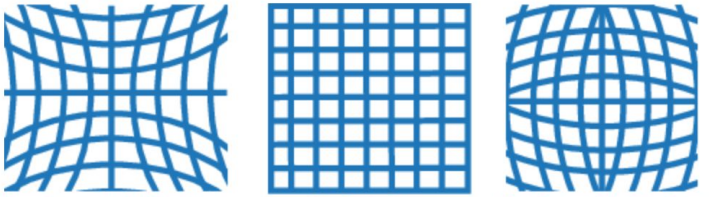
$$\begin{bmatrix} x_{p\_d} \\ y_{p\_d} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \text{skew\_c}f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{n\_d} \\ y_{n\_d} \\ 1 \end{bmatrix}$$

왜곡계수

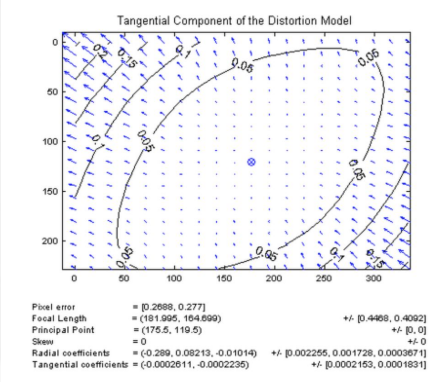
광각 = 왜곡 문제

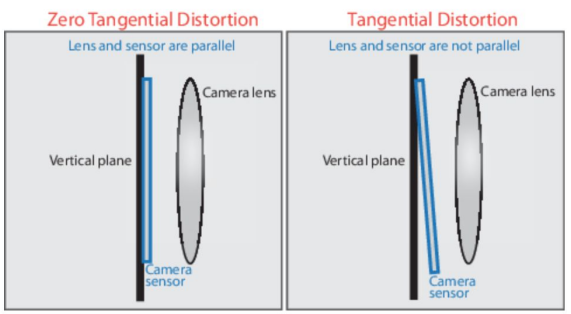
방사왜곡 k1, k2, (k3)
볼록렌즈의 굴절률
외곽부 굴절율 증가, 중심과의 거리 비례





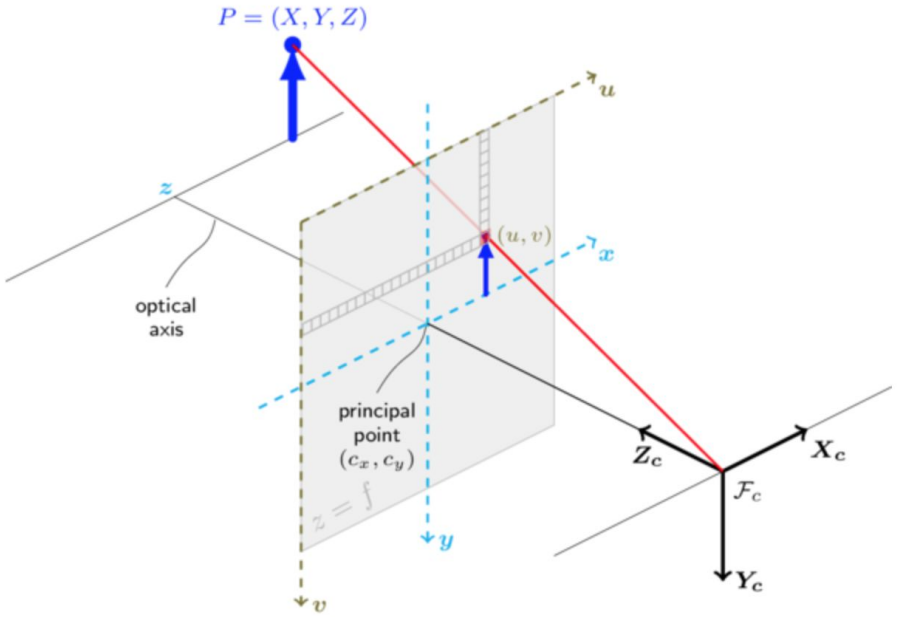
접선왜곡 p1, p2
렌즈와 센서가 평행하지 않아 생성
타원형태 왜곡분포





OpenCV의 구현

OpenCV의 카메라 캘리브레이션 구현



$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$

$$y' = y/z$$

$$x'' = x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2+2x'^2)$$

$$y'' = y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1(r^2+2y'^2) + 2p_2 x' y'$$

where $r^2 = x'^2 + y'^2$

$$u = f_x * x'' + c_x$$

$$v = f_y * y'' + c_y$$

k_1, k_2, k_3, k_4, k_5, and k_6 are radial distortion coefficients. p_1 and p_2 are tangential distortion coefficients. Higher-order coefficients are not considered in OpenCV.

캘리브래이션 패턴

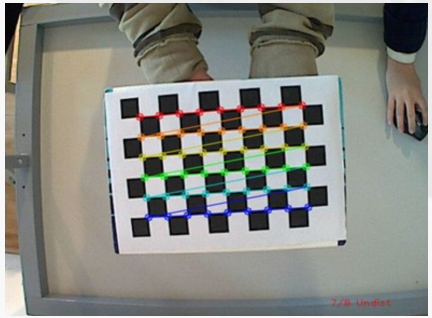
체스보드

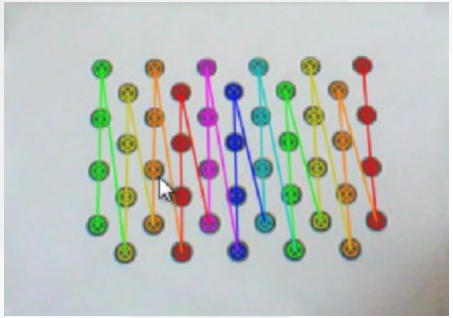
대칭형 원

비대칭형 원

```
vector<Point2f> pointBuf;

bool found;
switch( s.calibrationPattern ) // Find feature points on the input format
{
case Settings::CHESSBOARD:
  found = findChessboardCorners( view, s.boardSize, pointBuf,
  CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FAST_CHECK | CV_CALIB_CB_NORMALIZE_IMAGE);
  break;
case Settings::CIRCLES_GRID:
  found = findCirclesGrid( view, s.boardSize, pointBuf );
  break;
case Settings::ASYMMETRIC_CIRCLES_GRID:
  found = findCirclesGrid( view, s.boardSize, pointBuf, CALIB_CB_ASYMMETRIC_GRID );
  break;
}
```





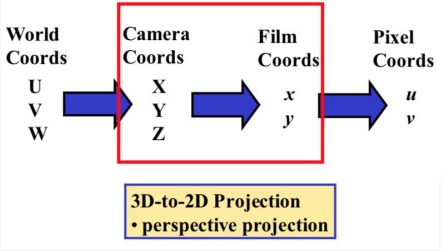
3D → 2D

3D-to-2D 변환

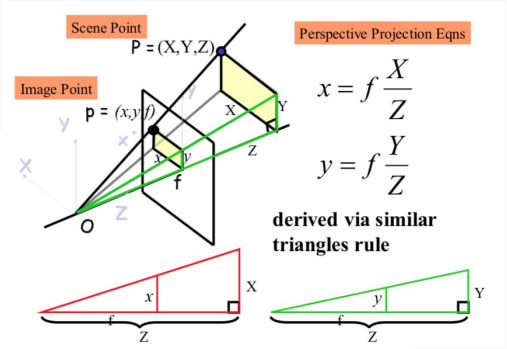
카메라 공간에서 Z축을 제거하고 2D로 투사

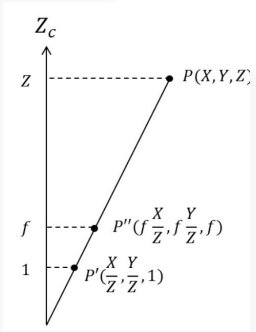
단순한 닮음비 이용

F / Z : 초점거리로 나눠주고 거리를 곱
행렬연산으로 가속



$$x = f\frac{X}{Z} \quad y = f\frac{Y}{Z} \quad \Leftrightarrow \quad \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$





월드-카메라 변환

Camera의 좌표축을 World의 좌표축에 정렬하는 작업

UVW축을 XYZ정렬을 위해 축을 회전(R)하고 평행이동

$$P_C = R\,(P_W - C)$$

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} U \\ V \\ W \\ 1 \end{pmatrix}$$

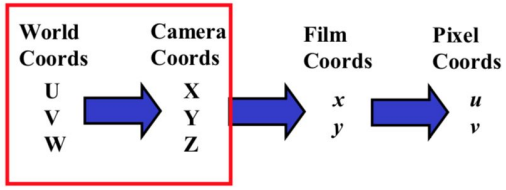
입력

내부파라미터와 왜곡계수 fx, fy, cx, cy, k1, k2, p1, p2

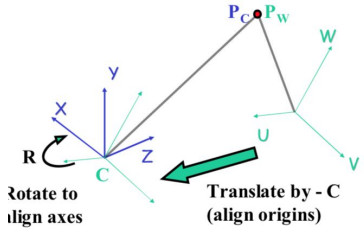
4개의 월드좌표와 이에 대응되는 2D 영상 좌표

출력

회전행렬 R에서 pan (yaw), tilt (pitch) 정보 도출



Rigid Transformation (rotation+translation) between world and camera coordinate systems

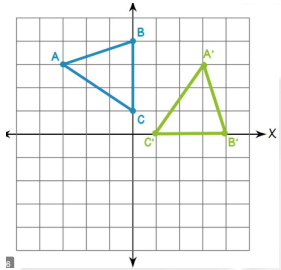


$P_C = R\,(P_W - C)$

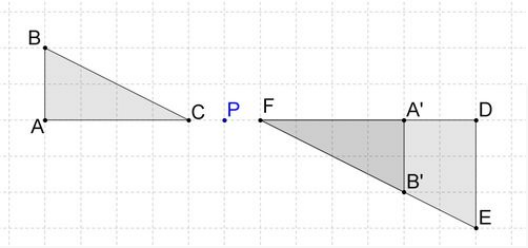
$$\begin{aligned} R &= R_z(p)R_x(t)R_x(-\pi/2) \\ &= R_z(p)R_x(-\pi/2+t) \\ &= \begin{bmatrix} \cos(p) & -\sin(p) & 0 \\ \sin(p) & \cos(p) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin(t) & \cos(t) \\ 0 & -\cos(t) & \sin(t) \end{bmatrix} \\ &= \begin{bmatrix} \cos(p) & -\sin(p)\sin(t) & -\sin(p)\cos(t) \\ \sin(p) & \cos(p)\sin(t) & \cos(p)\cos(t) \\ 0 & -\cos(t) & \sin(t) \end{bmatrix} \end{aligned} \quad \text{--- (1)}$$

좌표계변환

유클리드 변환
회전, 평행이동



유사변환
회전, 이동 + 크기



어파인변환
회전, 이동, 크기 + 기울이기, 반전

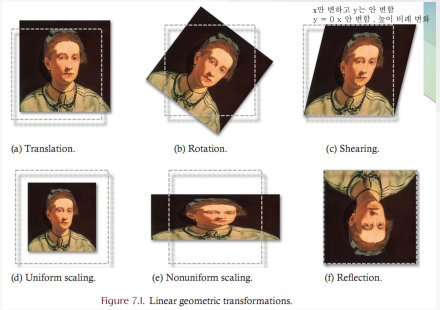
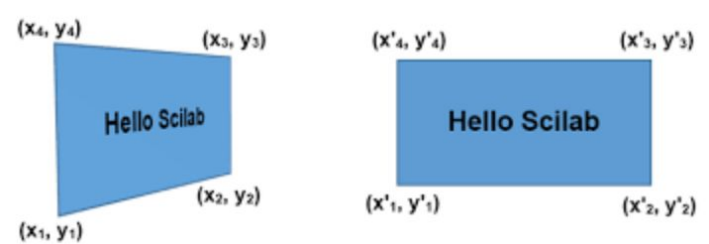


Figure 7.1. Linear geometric transformations.

원근변환 (Homography)



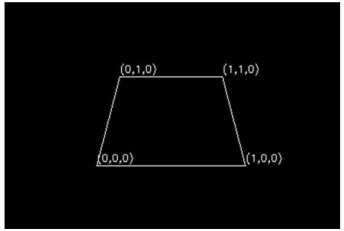
외부파라미터 구하기

내부파라미터 & 왜곡계수

```
// camera parameters
double m[] = {fx, 0, cx, 0, fy, cy, 0, 0, 1};      // intrinsic parameters
Mat A(3, 3, CV_64FC1, m);  // camera matrix
```

```
double d[] = {k1, k2, p1, p2};        // k1,k2: radial distortion, p1,p2: tangential distortion
Mat distCoeffs(4, 1, CV_64FC1, d);
```

4점의




```
Mat rvec, tvec;        // rotation & translation vectors
solvePnP(objectPoints, imagePoints, A, distCoeffs, rvec, tvec);
Mat R, T;
Rodrigues(rvec, R);
T = tvec;
```

```
// extract rotation matrix
Mat R;
Rodrigues(rvec, R);
Mat R_inv = R.inv();
```

$$R_u(\theta)=\begin{bmatrix} \cos\theta+u_x^2(1-\cos\theta) & u_xu_y(1-cos\theta)-u_z\sin\theta & u_xu_z(1-\cos\theta)+ \\ u_yu_x(1-\cos\theta)+u_z\sin\theta & \cos\theta+u_y^2(1-\cos\theta) & u_yu_z(1-\cos\theta)- \\ u_zu_x(1-\cos\theta)-u_y\sin\theta & u_zu_y(1-\cos\theta)+u_x\sin\theta & \cos\theta+u_z^2(1- \end{bmatrix}$$

```
// camera position (X,Y,Z)
Mat Cam_pos = -R_inv*tvec;
double* p = (double *)Cam_pos.data;
X = p[0];
Y = p[1];
Z = p[2];
```

```
// pan(yaw) & tilt(pitch)
double unit_z[] = {0,0,1};
Mat Zc(3, 1, CV_64FC1, unit_z);
Mat Zw = R_inv*Zc;            // world coordinate of optical axis
double* zw = (double *)Zw.data;

pan = atan2(zw[1], zw[0]) - CV_PI/2;
tilt = atan2(zw[2], sqrt(zw[0]*zw[0]+zw[1]*zw[1]));

// roll
double unit_x[] = {1,0,0};
Mat Xc(3, 1, CV_64FC1, unit_x);
Mat Xw = R_inv*Xc;            // world coordinate of camera X axis
double* xw = (double *)Xw.data;
double xpan[] = {cos(pan), sin(pan), 0};

roll = acos(xw[0]*xpan[0] + xw[1]*xpan[1] + xw[2]*xpan[2]); // inner product
if(xw[2]<0) roll = -roll;
```

$$X_w=R^{-1}X_c=[x_x,x_y,x_z]^T$$
$$X_{pan}=[\cos(\theta_{pan}),\sin(\theta_{pan}),0]^T$$
$$\theta_{roll}=sign(x_z)cos^{-1}\left(\frac{X_w\cdot X_{pan}}{\|X_w\|\,\|X_{pan}\|}\right)$$

외부파라미터 구하기

OpenCV 캘리브레이션 샘플

XML로 이미지 디렉토리 지정

```
<?xml version="1.0"?>
<opencv_storage>
<images>
images/CameraCalibration/VID5/xx1.jpg
images/CameraCalibration/VID5/xx2.jpg
images/CameraCalibration/VID5/xx3.jpg
images/CameraCalibration/VID5/xx4.jpg
images/CameraCalibration/VID5/xx5.jpg
images/CameraCalibration/VID5/xx6.jpg
images/CameraCalibration/VID5/xx7.jpg
images/CameraCalibration/VID5/xx8.jpg
</images>
</opencv_storage>
```

캘리브래이션 진행 후 파라미터 저장

```
bool runCalibrationAndSave(Settings& s, Size imageSize, Mat&  cameraMatrix, Mat& distCoeffs,vector<vector<Point2f> > imagePo
{
 vector<Mat> rvecs, tvecs;
 vector<float> reprojErrs;
 double totalAvgErr = 0;

 bool ok = runCalibration(s,imageSize, cameraMatrix, distCoeffs, imagePoints, rvecs, tvecs,
                          reprojErrs, totalAvgErr);
 cout << (ok ? "Calibration succeeded" : "Calibration failed")
     << ". avg re projection error = "  << totalAvgErr ;

 if( ok )   // save only if the calibration was done with success
     saveCameraParams( s, imageSize, cameraMatrix, distCoeffs, rvecs ,tvecs, reprojErrs,
                       imagePoints, totalAvgErr);
 return ok;
}
```

XML로 저장된 파라미터

```
<Camera_Matrix type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>d</dt>
<data>
 6.5746697944293521e+002 0. 3.1950000000000000e+002 0.
 6.5746697944293521e+002 2.3950000000000000e+002 0. 0. 1.</data></Camera_Matrix>
<Distortion_Coefficients type_id="opencv-matrix">
<rows>5</rows>
<cols>1</cols>
<dt>d</dt>
<data>
 -4.1802327176423804e-001 5.0715244063187526e-001 0. 0.
 -5.7843597214487474e-001</data></Distortion_Coefficients>
```

OpenCV의 카메라 캘리브래이션 API

OpenCV 캘리브레이션 API

findChessboardCorners ¶

Finds the positions of internal corners of the chessboard.

C++: bool **findChessboardCorners**(InputArray **image**, Size **patternSize**, OutputArray **corners**, int **flags**=CALIB_CB_ADAPTIVE_THRESH+CALIB_CB_NORMALIZE_IMAGE)

calibrateCamera ¶

Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.

C++: double **calibrateCamera**(InputArrayOfArrays **objectPoints**, InputArrayOfArrays **imagePoints**, Size **imageSize**, InputOutputArray **cameraMatrix**, InputOutputArray **distCoeffs**, OutputArrayOfArrays **rvecs**, OutputArrayOfArrays **tvecs**, int **flags**=0, TermCriteria **criteria**=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON))

solvePnP

Finds an object pose from 3D-2D point correspondences.

C++: bool **solvePnP**(InputArray **objectPoints**, InputArray **imagePoints**, InputArray **cameraMatrix**, InputArray **distCoeffs**, OutputArray **rvec**, OutputArray **tvec**, bool **useExtrinsicGuess**=false, int **flags**=ITERATIVE)

Rodrigues

Converts a rotation matrix to a rotation vector or vice versa.

C++: void **Rodrigues**(InputArray **src**, OutputArray **dst**, OutputArray **jacobian**=noArray())

fisheye::undistortImage ¶

Transforms an image to compensate for fisheye lens distortion.

C++: void fisheye::**undistortImage**(InputArray **distorted**, OutputArray **undistorted**, InputArray **K**, InputArray **D**, InputArray **Knew**=cv::noArray(), const Size& **new_size**=Size())

Parameters:
- **distorted** – image with fisheye lens distortion.
- **K** – Camera matrix $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$.
- **D** – Input vector of distortion coefficients (k_1, k_2, k_3, k_4).
- **Knew** – Camera matrix of the distorted image. By default, it is the identity matrix but you may additionally scale and shift the result by using a different matrix.
- **undistorted** – Output image with compensated fisheye lens distortion.

The function transforms an image to compensate radial and tangential lens distortion.