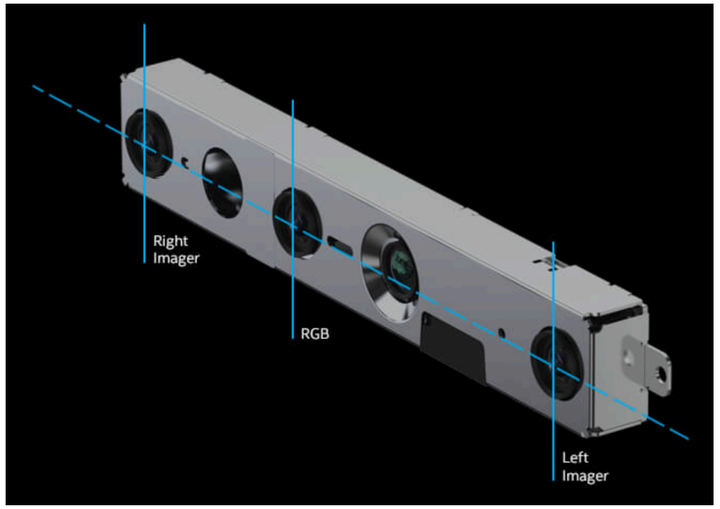
intel realsense

depth 값, kinect와 비교



인텔 리얼센스

인텔에서 개발한 3D 카메라 기술로, 3차원 공간 그 자체를 인지하고 “깊이감”을 이해할 수 있는 기술

Active IR Stereo(depth 인식 기술 중 하나)

- 인간을 비롯한 동물들은 한 쪽 눈만으로는 깊이감을 인지할 수 없음.
- “적외선”을 이용하여 사물을 인지
- 기본적인 원리: 1) 적외선을 이용하여 사물의 굴곡과 거리를 감지하여 3D 데이터를 추출
  2) 그 위에 메인카메라로 촬영한 2D 이미지를 덧씌워 사물의 입체적인 이미지를 촬영
- 실측을 통해 깊이 정확도를 알 수 있음.
  사물의 실제 입체 좌표를 인지하고 두 점의 입체 좌표 사이의 거리를 파악할 수 있음.

D455

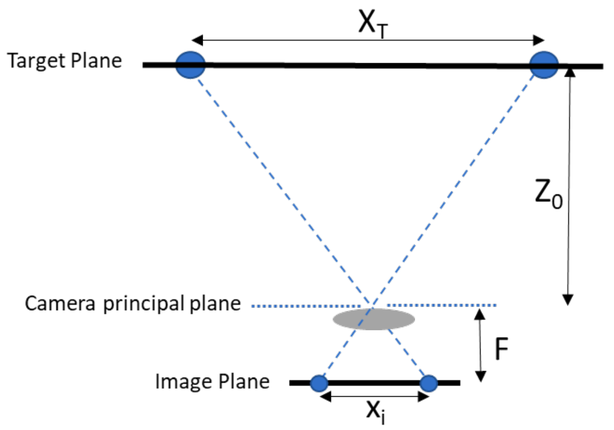
- IR 기능을 제외하고도 카메라가 3개
- 가운데 일반 RGB, 좌우에 카메라가 탑재되어 있음
- depth 값: 1) Depth카메라로 좌표의 Z값을 검출하고, 2) 왼쪽과 오른쪽의 이미저와 RGB 카메라를 비교하여 Z값의 오차를 줄임.

키넥트

첫번째 버전: 리얼센스와 같은 “적외선 굴곡 측정을 통한 객체 측정”

두번째 버전(V2)으로 부터: “ToF(Time of Flight)기술을 이용 - 더욱 정밀한 계측 가능

$$Z_0 = F \frac{X_T}{x_i} \quad \text{(Equation A1)}$$



좌표계

일종의 Protocol

좌표계란?

다양체의 점이나 기하학적 요소를 "**고유**"하게 결정하기 위해 좌표 사용

원점의 위치, x축, y축, z축, 좌표의 단위 결정

3D space

- World Coordinate system

$$P = (X, Y, Z)$$

- Camera Coordinate system

$$P_c = (X_c, Y_c, Z_c)$$

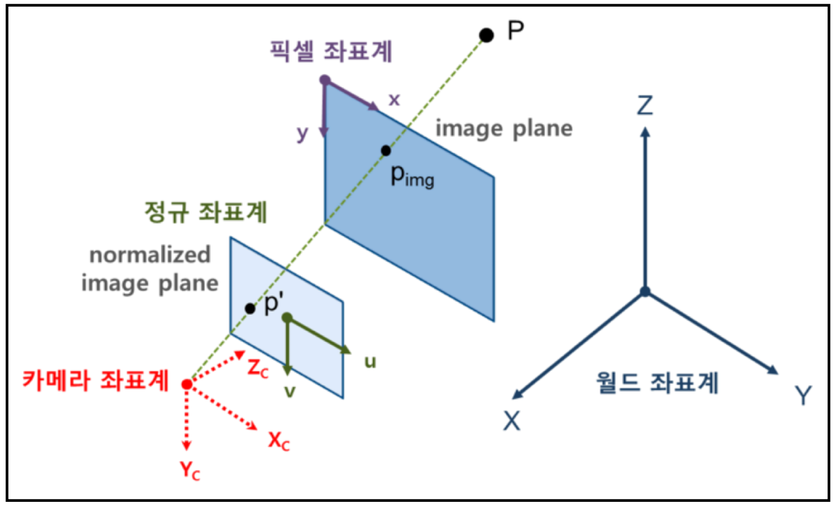
2D space

- Camera Pixel Coordinate system

$$p_{img} = (x, y)$$

- Nomarlized Image Coordinate system

$$p' = (u, v)$$



대표적인 4개의 좌표계

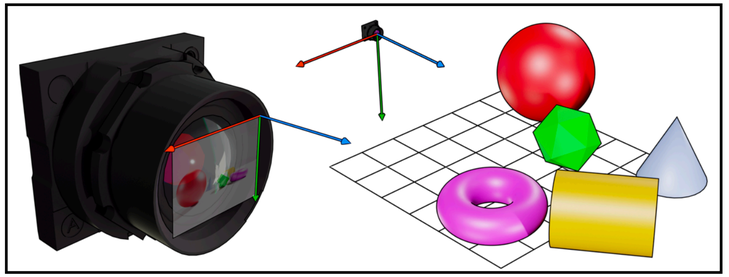
- (픽셀 좌표계 - 정규 좌표계) 변환

정규 좌표계 = 카메라의 내부 파라미터의 영향을 제거한 이미지 좌표계

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

위 식을 정리하면 아래처럼 변환된다.

$$x = f_x u + c_x$$
$$y = f_y v + c_y$$



Intel REALSENSE

intrinsic parameter

in Intel RealSense SDK 2.0

카메라 캘리브레이션

3D 공간좌표와 2D 영상좌표 사이의 변환관계를 설명하는 파라미터를 찾는 과정

- intrinsic parameter: 카메라의 고유 파라미터(카메라의 초점 거리, principal point, lens distortion model 등)

- extrinsic parameter: 카메라 설치 위치, 방향 & 월드 좌표계의 정의에 따라 달라짐

동작(intrinsic parameter 제거)

3D to 2D: projection

2D to 3D: deprojection

Parameter	Description
(w,h)	Width and height of video stream in pixels
$P=(p_x,p_y)$	Principal Point, as a pixel offset from the left edge
$F=(f_x,f_y)$	Focal Length in multiple of pixel size
Model	Lens Distortion Model
$Coeffs=(k_1,k_2,k_3,k_4,k_5)$	Lens Distortion Coefficients

Table 1. Camera Intrinsic Calibration Parameters.

No distortion, Positive Radial Distortion, Extreme Distortion with a wide-FOV lens

```
/* Given a point in 3D space, compute the corresponding pixel coordinates in an image with no distortion or forward distortion coefficients produced by the same
void rs2_project_point_to_pixel(float pixel[2], const rs2_intrinsics* intrin, const float point[3]);

/* Given pixel coordinates and depth in an image with no distortion or inverse distortion coefficients, compute the corresponding point in 3D space relative to
void rs2_deproject_pixel_to_point(float point[3], const rs2_intrinsics* intrin, const float pixel[2], float depth);
```

projection

in Intel RealSense SDK 2.0

```
typedef struct rs2_intrinsics
{
    int           width;     /**< Width of the image in pixels */
    int           height;    /**< Height of the image in pixels */
    float         ppx;       /**< Horizontal coordinate of the principal point of the image, as a pixel offset from the left edge */
    float         ppy;       /**< Vertical coordinate of the principal point of the image, as a pixel offset from the top edge */
    float         fx;        /**< Focal length of the image plane, as a multiple of pixel width */
    float         fy;        /**< Focal length of the image plane, as a multiple of pixel height */
    rs2_distortion model;     /**< Distortion model of the image */
    float         coeffs[5]; /**< Distortion coefficients */
} rs2_intrinsics;
```

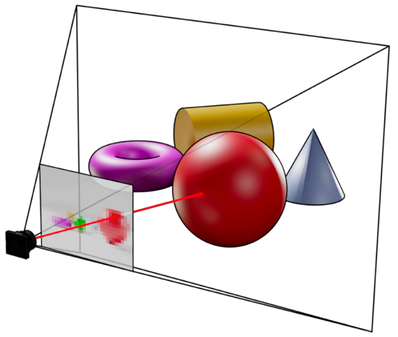
```
/* Given a point in 3D space, compute the corresponding pixel coordinates in an image with no distortion or forward distortion coefficients produced by the same
static void rs2_project_point_to_pixel(float pixel[2], const struct rs2_intrinsics * intrin, const float point[3])
{
    //assert(intrin->model != RS2_DISTORTION_INVERSE_BROWN_CONRADY); // Cannot project to an inverse-distorted image

    float x = point[0] / point[2], y = point[1] / point[2];

    if(intrin->model == RS2_DISTORTION_MODIFIED_BROWN_CONRADY)
    {

        float r2  = x*x + y*y;
        float f = 1 + intrin->coeffs[0]*r2 + intrin->coeffs[1]*r2*r2 + intrin->coeffs[4]*r2*r2*r2;
        x *= f;
        y *= f;
        float dx = x + 2*intrin->coeffs[2]*x*y + intrin->coeffs[3]*(r2 + 2*x*x);
        float dy = y + 2*intrin->coeffs[3]*x*y + intrin->coeffs[2]*(r2 + 2*y*y);
        x = dx;
        y = dy;
    }
    if (intrin->model == RS2_DISTORTION_FTHETA)
    {
        float r = sqrt(x*x + y*y);
        float rd = (1.0f / intrin->coeffs[0] * atan(2 * r* tan(intrin->coeffs[0] / 2.0f)));
        x *= rd / r;
        y *= rd / r;
    }

    pixel[0] = x * intrin->fx + intrin->ppx;
    pixel[1] = y * intrin->fy + intrin->ppy;
}
```



$$Proj(x, y, z) = F \cdot D_{Model}\left(\frac{x}{z}, \frac{y}{z}\right) + P$$

Where D_Model: R^2→R^2 is the lens distortion function

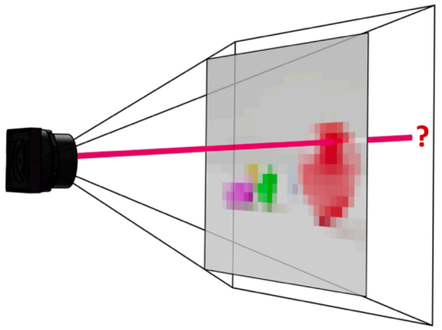
deprojection

in Intel RealSense SDK 2.0

```
typedef struct rs2_intrinsics
{
    int           width;     /**< Width of the image in pixels */
    int           height;    /**< Height of the image in pixels */
    float         ppx;       /**< Horizontal coordinate of the principal point of the image, as a pixel offset from the left edge */
    float         ppy;       /**< Vertical coordinate of the principal point of the image, as a pixel offset from the top edge */
    float         fx;        /**< Focal length of the image plane, as a multiple of pixel width */
    float         fy;        /**< Focal length of the image plane, as a multiple of pixel height */
    rs2_distortion model;     /**< Distortion model of the image */
    float         coeffs[5]; /**< Distortion coefficients */
} rs2_intrinsics;
```

```
/* Given pixel coordinates and depth in an image with no distortion or inverse distortion coefficients, compute the corresponding point in 3D space relative to
static void rs2_deproject_pixel_to_point(float point[3], const struct rs2_intrinsics * intrin, const float pixel[2], float depth)
{
    assert(intrin->model != RS2_DISTORTION_MODIFIED_BROWN_CONRADY); // Cannot deproject from a forward-distorted image
    assert(intrin->model != RS2_DISTORTION_FTHETA); // Cannot deproject to an ftheta image
    //assert(intrin->model != RS2_DISTORTION_BROWN_CONRADY); // Cannot deproject to an brown conrady model

    float x = (pixel[0] - intrin->ppx) / intrin->fx;
    float y = (pixel[1] - intrin->ppy) / intrin->fy;
    if(intrin->model == RS2_DISTORTION_INVERSE_BROWN_CONRADY)
    {
        float r2  = x*x + y*y;
        float f = 1 + intrin->coeffs[0]*r2 + intrin->coeffs[1]*r2*r2 + intrin->coeffs[4]*r2*r2*r2;
        float ux = x*f + 2*intrin->coeffs[2]*x*y + intrin->coeffs[3]*(r2 + 2*x*x);
        float uy = y*f + 2*intrin->coeffs[3]*x*y + intrin->coeffs[2]*(r2 + 2*y*y);
        x = ux;
        y = uy;
    }
    point[0] = depth * x;
    point[1] = depth * y;
    point[2] = depth;
}
```



$$Deproj(i,j,d) = \left(d \cdot U_{Model}\left(\frac{(i,j)-P}{F}\right), d\right)$$

Where U_{Model}: R^2→R^2 is the undo lens distortion function.

extrinsic parameter

Projection in Intel RealSense SDK 2.0

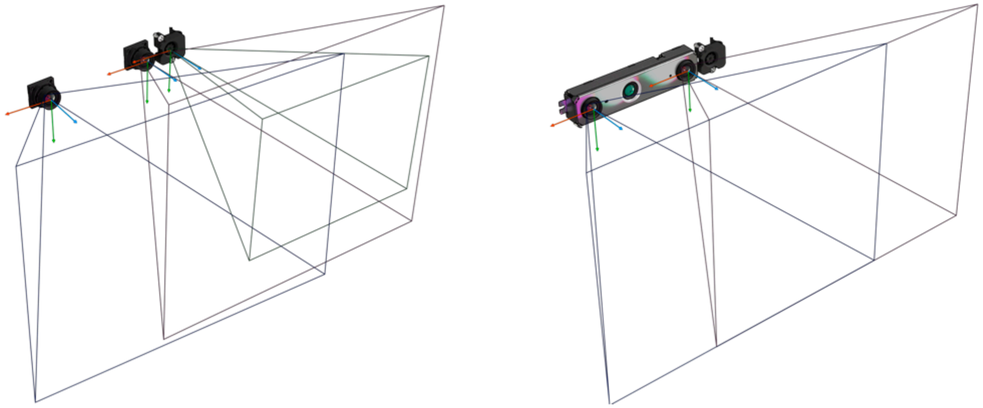
카메라 외부 파라미터

- 카메라 설치 위치, 방향 & 월드 좌표계의 정의에 따라 달라짐
- 카메라 좌표계와 월드 좌표계 사이의 변환 관계를 설명하는 파라미터, 두 좌표계 사이의 평행이동(translation) 및 회전(rotation) 변환으로 표현됨.
- 캘리브레이션을 통해 구한 intrinsic parameter를 먼저 구한 후, extrinsic parameter를 구한다.

multi sensor

센서는 서로에 대해 서로 다른 위치와 방향에 장착됨.

센서 A에 대한 월드 좌표계를 센서 B에 대한 월드 좌표로 이동 - 행렬 곱셈을 통해 수행



$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_B = (R \quad t) \begin{pmatrix} x \\ y \\ z \end{pmatrix}_A \qquad \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{R} \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + \mathbf{t}$$

```
/* Transform 3D coordinates relative to one sensor to 3D coordinates relative to another viewpoint */
static void rs2_transform_point_to_point(float to_point[3], const struct rs2_extrinsics * extrin, const float from_point[3])
{
    to_point[0] = extrin->rotation[0] * from_point[0] + extrin->rotation[3] * from_point[1] + extrin->rotation[6] * from_point[2] + extrin->translation[0];
    to_point[1] = extrin->rotation[1] * from_point[0] + extrin->rotation[4] * from_point[1] + extrin->rotation[7] * from_point[2] + extrin->translation[1];
    to_point[2] = extrin->rotation[2] * from_point[0] + extrin->rotation[5] * from_point[1] + extrin->rotation[8] * from_point[2] + extrin->translation[2];
}
```

변환행렬: (translation vector t and rotation matrix R)