Chapter 4

3D Geometry and Camera Model

In this chapter, we will learn basics of Geometry, particularly how an object motion formulated in maths. Mathematically, this sounds simple in an Euclidean space, nevertheless motion analysis is, in real-life, a very complex problem due to many aspects which arise differently from different sensors in use. Typically motion analysis is also a complex problem for human vision (human eye and brain), because human vision system works in a projective space, but not Euclidean space. Why is this matter? Projective space means viewing scenes from a virtual single perspective view, which means depths or scales are somehow ambiguous. This is why you can tell roughly how things moving but exactly talking about speeds and distances is difficult. Similar to human vision system, camera also works in a projective space, and thus facing a similar challenge.

In fact, since sensors are mounted on a target (robot, car, human-human eyes, etc.), it is naturally that its view is similar to a virtual single perspective view, except for a combination of sensors. However, an active sensor (LIDAR, RADAR, SONAR, etc.) can still achieve a high accurate distance and speed of a moving object because of using its "active" signal (Laser, radio sound, ultrasound, etc.). Unfortunately, active sensors usually provide a sparse data, and just about collecting a particular information of objects, for example 3D structure from laser scanning. Meanwhile, vision system can provide very detailed information of objects which are straightforward inferred by human interpretation. Given those facts, it has been a trend in Robotics and Autonomous Car communities to exploit a combination of both systems. This leads to a need of understanding a Geometric mapping between different spaces, different coordinates and different systems, the so called 2D-3D mapping. For that aim, we will learn some basic knowledge of Camera Systems and its homogeneous coordinate in this chapter.

4.1 3D Geometry: Rigid-body Motion

This section will contain a sufficient knowledge of 3D geometry with Lie Algebra to understand 3D vision in deep, but not including all related topics on 3D Geometry and Lie Algebra. Therefore, if you wish to get deeper into 3D geometry and Lie Algebra, please refer to [9] or other pure mathematics books [7].

Consider an object moving in a Euclidean space. In order to describe its motion one should specify the trajectory of every small piece, or even every single point, on the object. For example, different parts (hands, head, legs, etc.) of human body move differently, so it is not correct to describe all at once. In fact each separate part moves consistently, so such an object like human body consists of many parts which move consistently, so called rigid-body. For a simplification, we only discuss a motion of a rigid-body in this chapter.

A rigid-body motion in a Euclidean space can be 1) Translation; 2)Rotation; 3) Combination of both Translation and Rotation, see Fig.4.1.

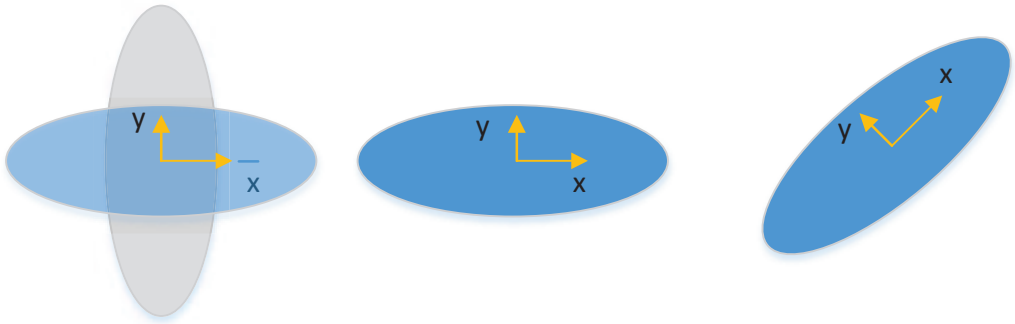

Figure 4.1: Illustration of rigid-body motion

Translation vector $T(t_1, t_2, t_3)$. If a rigid-body, denoted as O_1, has a pure translation of T to a new position O_2, then each point on O_2 corresponds to a point in O_1 following the below expression

$$P_2 = P_1 + T$$

If R is a 3×3 rotation matrix, similar to the case of having a pure rotation

$$P_2 = RP_1$$

In the case of having a combination of rotation and translation, we have

$$P_2 = RP_1 + T \tag{4.1}$$

4.1.1 Rotation in 3D space

Remember that in high school maths, we usually deal with an example of a Rotation Matrix which represents a rotation about the Z-axis by an angle θ_z as follow

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

How about the case of having an arbitrary rotation? Any arbitrary rotation can be decomposed into three rotation R_x, R_y, R_z where

$$R = R_z R_y R_x$$

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

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

Note that we already decompose a transformation into a combination of a pure rotation and a pure translation. Thereby, a rotation should keep the same shape or scale of the object. If we express R into column representation: $R = [r_1 \quad r_2 \quad r_3]$, then $r_1, r_2, r_3]$form an orthonormal frame:

$$r_i^T r_j = \begin{cases} 1 & for \quad i = j \\ 0 & for \quad i \neq j \end{cases}$$

Hence

$$R^T R = I$$

Rotation in 3D space, which transform axes O_x, O_y, O_z into new axes O_u, O_v, O_w, can be decomposed into 3 rotations against three Euclidean axes X, Y, Z. In fact, it is sufficient to know the transformations from O_x, O_z to O_u, O_w, the remaining transformation from O_y to O_v can be derived from the first two.

4.1.2 Derivation of Rodrigues Rotation Matrix

Subsection 4.1.1 shows a rigorous representation of a rotation matrix. In real-life problems, we commonly observe a rotation of a rigid-body object around a fixed axis, see Fig.4.1.2. Since the fixed axis does not coincide with our working Euclidean coordinate, to analyse a motion of a part of the rigid-body, we need to decompose the motion of it into three rotations R_x, R_y, R_z and three translation T_x, t_y, T_z. Let assume that Doener is rotated slowly at the beginning and faster later when it is almost done. Speed analysis will involve taking a derivative of such transformation, which will end up with a quite complex and ugly maths.

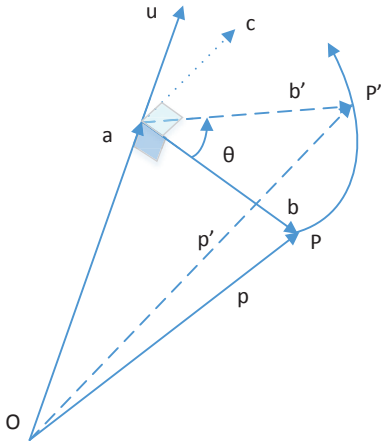

Figure 4.2: Left: Grilling Doener; Right: Rotating a vector p of θ degree around the axis u

On the other hand, one might see that the object is rotating around the fixed axis u, and thus any point P on the object is just rotating around u from a distance b. Therefore, it is trivial to express its speed in time. For that aim, the rotation must

be a function of θ. Let denote a vector p from the origin O to P, after a rotation θ around u we have p'.

Note that a dot product of u, p can be described as

$$(u.p) = u^T p = ||u||||p||cos\theta = ||p||cos\theta$$

The projection of vector p on u is represented by $a = uu^T p$, u is a unit vector.

$$a = u(u.p) = uu^T p = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

Using basic vector operations in 3D space, we have

$$\begin{cases} a = uu^T p \\ c = u \times p \\ b = p - a = (1 - uu^T)p \end{cases}$$

Where $(\times)$ is a cross product, mentioned in Equation 3.4.

Rotated version of p is p'

$$p' = a + b' = a + bcos\theta + csin\theta = uu^T p + (1 - uu^T)pcos\theta + u \times psin\theta$$

$$p' = [Icos\theta + (1 - cos\theta)uu^T + u_\times sin\theta]p$$

and thus we end-up with the Rodrigues form

$$R = Icos\theta + (1 - cos\theta)uu^T + u_\times sin\theta \tag{4.2}$$

Note that $Icos\theta + (1 - cos\theta)uu^T$ is a symmetric matrix. Meanwhile $u_\times sin\theta$ is anti-symmetric matrix, see definition of a cross-matrix in Equation 3.5.

Hence, it is trivial to derive: $R - R^T = 2u_\times sin\theta$, and $trace(R) = 3cos\theta + (1 - cos\theta) = 2cos\theta + 1$. Or,

$$cos\theta = \frac{r_{11} + r_{22} + r_{33} - 1}{2}$$

4.1.3 Connection to Lie Group SO(3)

Equation 4.2 in matrix representation is

$$R = cos\theta \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1-cos\theta) \begin{bmatrix} u_1^2 & u_1u_2 & u_1u_3 \\ u_1u_2 & u_2^2 & u_2u_3 \\ u_3u_1 & u_3u_2 & u_3^2 \end{bmatrix} + sin\theta \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}$$

Or,

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} + (1-cos\theta) \begin{bmatrix} u_1^2 - 1 & u_1u_2 & u_1u_3 \\ u_1u_2 & u_2^2 - 1 & u_2u_3 \\ u_3u_1 & u_3u_2 & u_3^2 - 1 \end{bmatrix} + sin\theta \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}$$

Note that u is a unit vector, thus $u_1^2 + u_2^2 + u_3^2 = 1$,

$$R = I + sin\theta u_\times + (1 - cos\theta)u_\times^2 \tag{4.3}$$

A matrix $\omega_\times$ is called a skew symmetric matrix if it is written in the following form

$$\omega_\times = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$

So $u_\times$ is a special a skew symmetric matrix which has the length of 1. For a generic purpose, let denote a skew matrix $\omega_\times = \theta u_\times$. It is trivial to derive the following property of a skew matrix

$$\omega_\times^3 = -(\omega^T\omega)\omega_\times \tag{4.4}$$

where $\omega = [\omega_1 \quad \omega_2 \quad \omega_3]^T$, and $\omega^T\omega = \theta^2$

Re-write the Rodrigues as follow

$$R = I + \frac{sin\theta}{\theta}\omega_\times + \frac{1 - cos\theta}{\theta^2}\omega_\times^2 \tag{4.5}$$

When θ is very small, applying Taylor expansion on Equation 4.5 we have

$$R = I + (1 - \frac{\theta^2}{3!} + \frac{\theta^4}{5!} + ...)\omega_\times + (\frac{1}{2} - \frac{\theta^2}{4!} + \frac{\theta^4}{6!} + ...)\omega_\times^2$$

Or,

$$R = I + (\sum_{i=1}^{\infty} \frac{(-1)^i\theta^{2i}}{(2i+1)!})\omega_\times + (\sum_{i=1}^{\infty} \frac{(-1)^i\theta^{2i}}{(2i+2)!})\omega_\times^2$$

From the Property 4.4, we have

$$\omega_\times^{2i+1} = (-1)^i\theta^{2i}\omega_\times$$
$$\omega_\times^{2i+2} = (-1)^i\theta^{2i}\omega_\times^2$$

Therefore

$$R = exp(\omega_\times) = I + \sum_{i=1}^{\infty} \left[\frac{\omega_\times^{2i+1}}{(2i+1)!} + \frac{\omega_\times^{2i+2}}{(2i+2)!}\right] \tag{4.6}$$

$exp(\omega_\times)$ is called exponential map in Lie Algebra.

Definition 4. The exponential map, which takes a skew symmetric matrix to a rotation matrix, is simply the matrix exponential over a linear combination of the generators

For every $\omega_\times$ where $\omega \in \mathbb{R}^3$, we can re-write the skew symmetric matrix as follow

$$\omega_\times = \omega_1 G_1 + \omega_2 G_2 + \omega_3 G3$$

where

$$G_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Definition 5. The 3D rotation group, often denoted SO(3), is the group of all rotation about the origin of 2D Euclidean space $\mathbb{R}$ under the operation of composition or exponential map. In Lie Algebra, $SO(3)$ is a set of 3×3 skew-symmetric matrices.

We have already derived connection between Lie group $SO(3)$ to a rotation matrix in 3D space.

4.1.4 Lie Group SE(3)

Representation

When taken into account a whole transformation of a rigid-body, an additional element related to the translation is added

$$R \in SO(3), t \in \mathbb{R}^3$$

$$C = \left(\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right) \tag{4.7}$$

$$C^{-1} = \left(\begin{array}{c|c} R^T & -R^T t \\ \hline 0 & 1 \end{array}\right)$$

For each vector x,

$$x = (x \quad y \quad z \quad 1)^T \in \mathbb{RP}^3$$

$$C.x = \begin{pmatrix} R(x & y & z)^T + t \\ & 1 & \end{pmatrix}$$

Definition 6. In Lie Algebra, $SE3(3)$ is a set of 4×4 matrices corresponding to differential translations and rotations (as in $SO(3)$).

Let define some generators as below

$$G_1 = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$G_4 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad G_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$(u \quad \omega)^T \in \mathbb{R}^6$$

$$\xi = u_1G_1 + u_2G_2 + u_3G_3 + \omega_1G_4 + \omega_2G_5 + \omega_3G_6 \in SE(3)$$

$$\xi = \left[\begin{array}{c|c} \omega_\times & t \\ \hline 0 & 0 \end{array}\right]$$

Exponential Map

$$exp(\xi) = exp\left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & 0 \end{array}\right)$$

$$exp(\xi) = I + \left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & 0 \end{array}\right) + \frac{1}{2!}\left(\begin{array}{c|c} \omega_\times^2 & \omega_\times u \\ \hline 0 & 0 \end{array}\right) + \frac{1}{3!}\left(\begin{array}{c|c} \omega_\times^3 & \omega_\times^2 u \\ \hline 0 & 0 \end{array}\right) + ...$$

Rotation block is the same as for SO(3)

$$exp\left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & 0 \end{array}\right) = \left(\begin{array}{c|c} exp(\omega_\times) & Vu \\ \hline 0 & 1 \end{array}\right) \tag{4.8}$$

Where

$$V = I + \sum_{i=0}^{\infty}\left(\frac{\omega_\times^{2i+1}}{(2i+2)!} + \frac{\omega_\times^{2i+2}}{(2i+3)!}\right)$$

$$V = I + \left(\sum_{i=0}^{\infty}\frac{(-1)^i\theta^{2i}}{(2i+2)!}\right)\omega_\times + \left(\sum_{i=0}^{\infty}\frac{(-1)^i\theta^{2i}}{(2i+3)!}\right)\omega_\times^2$$

$$V = I + \frac{1 - cos\theta}{\theta^2}\omega_\times + \frac{\theta - sin\theta}{\theta^3}\omega_\times^2$$

4.1.5 $Sim(3)$: Similarity Transformation in 3D space

Representation

Similarity transformation are combinations of rigid transformation and scaling, denoted as $Sim(3)$. $Sim(3)$ has a nearly identical representation of to $SE(3)$ with an additional scale factor:

$$R \in SO(3), t \in \mathbb{R}^3, s \in \mathbb{R}$$

$$T = \left(\begin{array}{c|c} R & t \\ \hline 0 & s^{-1} \end{array}\right) \in Sim(3) \tag{4.9}$$

It is trivial to derive

$$T^{-1} = \left(\begin{array}{c|c} R^T & -sR^T t \\ \hline 0 & s \end{array}\right)$$

$$T_1 T_2 = \left(\begin{array}{c|c} R_1 R_2 & R_1 t_2 + s_2^{-1} t_1 \\ \hline 0 & (s_1 s_2)^{-1} \end{array}\right)$$

T is commonly used in computer vision to encode the scaling s, so called rigid transformation followed by scaling.

$$x = (x \quad y \quad z \quad 1)^T \in \mathbb{RP}^3$$

$$T.x = \begin{pmatrix} R(x & y & z)^T + t \\ & s^{-1} & \end{pmatrix}$$

The generators of $Sim(3)$ include all generators of $SE(3)$ and additionally add

$$G_7 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

An element of $Sim(3)$ is a combination of generators

$$(u \quad \omega \quad \lambda)^T \in \mathbb{R}$$

$$T = u_1 G_1 + u_2 G_2 + u_3 G_3 + \omega_1 G_4 + \omega_2 G_5 + \omega_3 G_6 + \lambda G_7 \in Sim(3)$$

Exponential Map

Similar to $SE(3)$, we have

$$\delta = (u \quad \omega \quad \lambda) \in Sim(3)$$

$$exp(\delta) = exp\left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & -\lambda \end{array}\right)$$

$$exp(\delta) = I + \left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & \lambda \end{array}\right) + \frac{1}{2!}\left(\begin{array}{c|c} \omega_\times^2 & \omega_\times u - \lambda u \\ \hline 0 & \lambda^2 \end{array}\right) + \frac{1}{3!}\left(\begin{array}{c|c} \omega_\times^3 & \omega_\times^2 u - \lambda\omega_\times u + \lambda^2 u \\ \hline 0 & -\lambda^3 \end{array}\right) + ...$$

$$exp\left(\begin{array}{c|c} \omega_\times & u \\ \hline 0 & -\lambda \end{array}\right) = \left(\begin{array}{c|c} \exp(\omega_\times) & Vu \\ \hline 0 & exp(-\lambda) \end{array}\right) \tag{4.10}$$

$$V = \sum_{n=0}^{\infty}\sum_{k=0}^{n} \frac{\omega_\times^{n-k}(-\lambda)^k}{(n+1)!}$$

$$V = \sum_{k=0}^{\infty}\sum_{n=k}^{\infty} \frac{\omega_\times^{n-k}(-\lambda)^k}{(n+1)!}$$

$$V = \sum_{k=0}^{\infty}\sum_{j=0}^{\infty} \frac{\omega_\times^{j}(-\lambda)^k}{(j+k+1)!}$$

Remind the property of the skew symmetric matrix $\omega_\times$: $\theta^2 = \omega^T\omega$, and thus

$$V = \left(\sum_{k=0}^{\infty}\frac{(-\lambda)^k}{(k+1)!}\right) I + \sum_{k=0}^{\infty}(-\lambda)^k \sum_{i=0}^{\infty}\left[\frac{\omega_\times^{2i+1}}{(2i+k+2)!} + \frac{\omega_\times^{2i+2}}{(2i+k+3)!}\right]$$

$$V = \left(\sum_{k=0}^{\infty}\frac{(-\lambda)^k}{(k+1)!}\right) I + \left(\sum_{k=0}^{\infty}\sum_{i=0}^{\infty}\frac{(-1)^i\theta^{2i}(-\lambda)^k}{(2i+k+2)}\right)\omega_\times + \left(\sum_{k=0}^{\infty}\sum_{i=0}^{\infty}\frac{(-1)^i\theta^{2i}(-\lambda)^k}{(2i+k+3)!}\right)\omega_\times^2$$

4.2 Cameras and Vehicle Coordinate Transformation

4.2.1 Camera Mounting

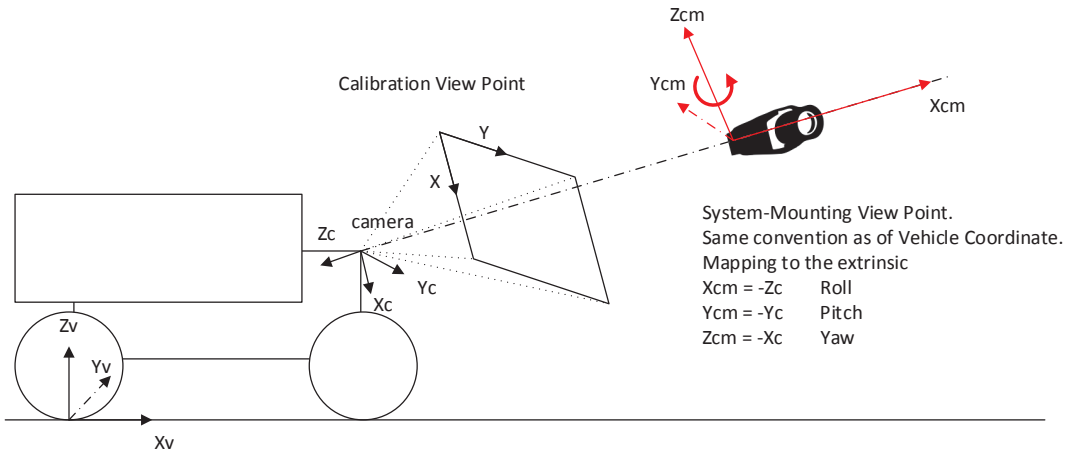

Figure 4.3: System Coordinate Convention

Fig.4.3 shows an example of robotic system or autonomous car system which mounts a camera in it. Other active sensors, including LIDAR, RADAR, SONAR, IMU, etc. can be mounted around the vehicle to obtain 3D or object position information, see an example of AMOR in Fig.2.1.

In common, rotation directions, R_x, R_y, R_z, follow counter clock-wise. I myself prefer to use clock-wise - just a personal habit, nothing to say good or bad in this matter.

4.2.2 Transformation between Coordinates

If choosing the middle of the rear axial of the ego-vehicle as the origin of our autonomous system, it is required to know the relative pose (position and rotation) of all sensors against the origin, so called the origin of vehicle. Assume a 3x3 rotation matrix R_{V2C} and a translation vector T_{V2C} are to form the transformation from a

camera to the vehicle coordinate. We will have $[R_{C2V}|T_{V2C}]$ are the transformation from vehicle back to the camera coordinate, where due to orthonormality

$$R_{C2V} = R_{V2C}^{-1}$$
$$T_{C2V} = -T_{V2C}$$

A 3D point $P^V(x_v, y_v, z_v)$ in vehicle coordinate, when transformed to camera coordinate we have

$$P^C = R_{C2V}P^V + T_{C2V}$$

Vice versa

$$P^V = [R_{C2V}]^{-1}P^C - [R_{C2V}]^{-1}T_{C2V}$$

4.3 Mapping from 2D image to 3D camera coordinate system

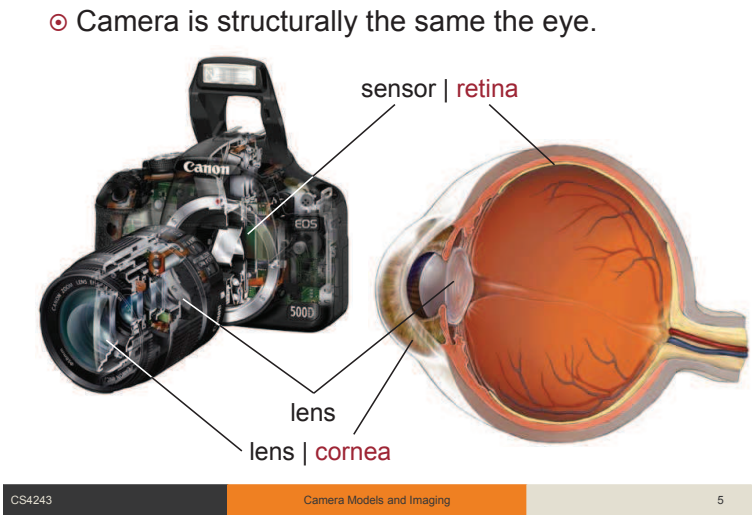

Figure 4.4: Similarity between Cameras and Eyes

In rough, a camera consists of an imaging sensor, lens, and other mechanical and electronic components. Similar to human eye, a camera will use its lens to gather all light information from environment to project into its image plane, then converted to image data by an imaging sensor. Raw image data from a camera is a flipped, upside down, image of the world scene. There is an Image Signal Processor (ISP) chip inside the camera, which, in some cases, include an inverter to flip back the image to be in a normal form as human eyes see, see Fig.4.4.

At an initiative state, to be easy for understanding and implementing, a camera is designed following a Pin-hole model,
please read Pinhole Camera [1] for a basic understanding. Later, due to different requirements and applications, more sophisticated lens and cameras have been introduced, remarkably Fish-eye lens, please read [2].

4.3.1 Pinhole-Camera Model

A pinhole imaging model is illustrated in Fig.4.5. The distance from the real image plane to the centre of camera, or often called centre of projection, is called Focal Length, denoted by f. For an easy intuition, we mainly work on the virtual image plane which is a mirror of the real image plane with respect to the centre of projection of the camera O. The distance from the virtual image plane to the centre of camera is also f. **From now on, let forget the real image plane and call the virtual image plane as the image plane**. This helps us be easier to imagine what camera sees in a similar manner as of human-eyes.

A point $p_i(x, y)$ in an image is corresponding to a world point $P_W(u, v, w)$, or a point $P_C(x, y, z)$ in the camera coordinate. It is clear that P_C **and** P_W **refer the the same physical point in world**, but just different representations in the camera coordinate and the world coordinate, respectively. Hence, P_C belongs to the line extension of the vector $\vec{Op}$, see Fig.4.5. Assume that $I(x_c, y_c)$ is the centre of the image, then $I(x_c, y_c)$ lies on the optical axis $\vec{OZ}$.

From basic geometric relations between the image $p_i(x, y)$ and the object $P_C(X, Y, Z)$

$$\begin{bmatrix} (x - x_c)\delta_x \\ (y - y_c)\delta_y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Or,

[1] https://en.wikipedia.org/wiki/Pinhole_camera
[2] https://en.wikipedia.org/wiki/Fisheye_lens

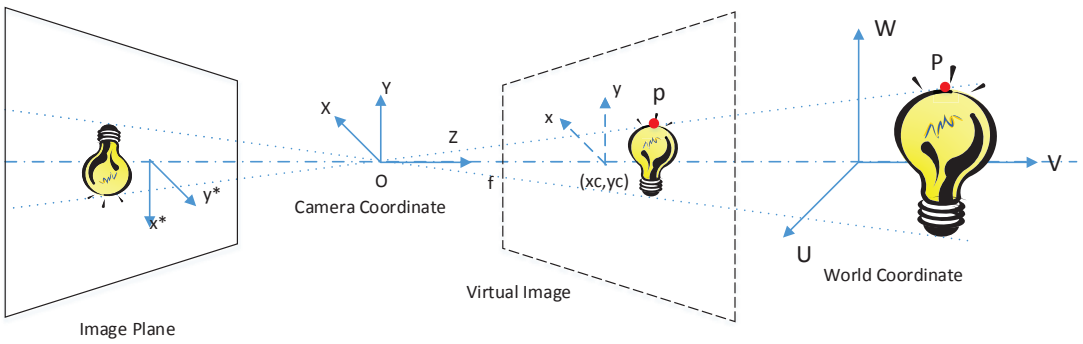

Figure 4.5: The Pinhole Imaging Model

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (f\delta_x)\frac{X}{Z} \\ \\ (f\delta_y)\frac{Y}{Z} \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{4.11}$$

Where (δ_x, δ_y) denote pixel sizes. Such multiplication with pixel sizes converts from the image coordinate to image plane coordinate - this fundamental difference between image and image plane is usually misunderstood by students. Talking about images, you are working on image coordinate where distance is measured by a number of pixels - this has nothing to do with the camera coordinate in Euclidean space. Thus, to link them, we first need such transformation to the image plane coordinate in Euclidean space. In most of computer vision books, you don't see such pixel size item. This is because they assume that the camera has square pixels, which is not necessary the case in real-life cameras.

Let denote $x_{norm} = X/Z, \quad y_{norm} = Y/Z$, then (x_{norm}, y_{norm}) is a projection of the point P_C on a normalized image plane (focal length = 1). Re-write Equation 4.11 with subject to (x_{norm}, y_{norm}),

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f\delta_x & 0 & x_c \\ 0 & f\delta_y & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{norm} \\ y_{norm} \\ 1 \end{bmatrix} \tag{4.12}$$

Let denote,

$$\mathbb{K} = \begin{bmatrix} f\delta_x & 0 & x_c \\ 0 & f\delta_y & y_c \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathbb{K}$ matrix represents intrinsic parameters of the camera. In fact, camera coordinate system might be skewed, due to some manufacturing errors, so the angle θ between two image axes is not equal to 90^0, but very close.

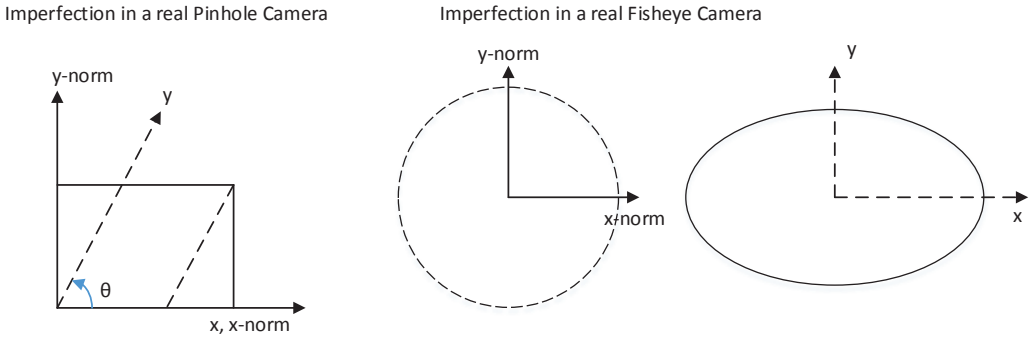

Figure 4.6: Imperfection or Manufacturing Error in a real Camera

From Fig.4.6, it is trivial to show that

$$\begin{cases} x - x_c = (f\delta_x)x_{norm} - (f\delta_x)cot\theta \\ y - y_c = \frac{f\delta_y}{sin\theta} y_{norm} \end{cases} \tag{4.13}$$

Hence, $\mathbb{K}matrix$ will be

$$\mathbb{K} = \begin{bmatrix} f\delta_x & -f\delta_x cot\theta & x_c \\ 0 & \frac{f\delta_y}{sin\theta} & y_c \\ 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

Finally, either lens are not perfect or installation of camera-components is not perfect, which creates distortions, and thus $\mathbb{K}$ should be much more complicated than Equation 4.14. In fact, the problem of estimating the distortions of Pinhole camera is a small subset of a general problem of estimating distortions of a Fisheye camera. Therefore, we will re-visit this topic in Fisheye camera section later.

4.3.2 Fisheye Camera: Scaramuzza Model

For Fisheye lens, we have many different models to approach or approximate the mapping in and out between camera coordinate and world coordinates [12].

The following equations describe the process of mapping a pixel $p = [x\ y]^T$ in the 2D image coordinate system to a ray r_p in the 3D camera coordinate system using the model of Scaramuzza [16].

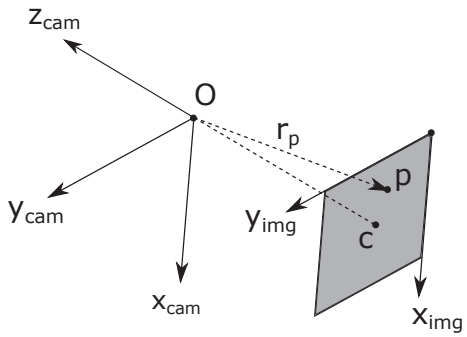

Figure 4.7: Mapping a pixel p in the 2D image coordinate system to a ray r_p in the 3D camera coordinate system. According to the Scaramuzza's model [16], x and y are components in the image coordinate system that denote row and column of a pixel, respectively.

First, the pixel coordinates are normalized by subtracting the principal point $c = [x_c\ y_c]^T$, and then applying the inverse affine transform A^{-1}, where the matrix A is expressed as

$$A = \begin{bmatrix} m_C & m_D \\ m_E & 1 \end{bmatrix} \tag{4.15}$$

The pixel coordinates after normalization are

$$p' = \begin{bmatrix} x' \\ y' \end{bmatrix} = A^{-1}(p - c) = \frac{1}{m_C - m_D m_E} \begin{bmatrix} 1 & -m_D \\ -m_E & m_C \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_c \\ y_c \end{bmatrix} \right). \tag{4.16}$$

We calculate next the n-order polynomial $P(\rho) = a_0 + a_1\rho + a_2\rho^2 + \cdots + a_n\rho^n$, using the L_2 norm of the normalized pixel p' as the argument ρ:

$$\rho = \|p'\| = \sqrt{x'^2 + y'^2}. \tag{4.17}$$

Finally, we obtain the ray r_p in the 3D camera coordinate system as

$$r_p \propto \begin{bmatrix} x' \\ y' \\ P(\rho) \end{bmatrix}. \tag{4.18}$$

Typically $n = 4$ is good enough to model most of Fisheye lens in the market.

To be updated ...