Slambook 정리

Namtae Kim

July 17, 2023

Contents

0.1 Main reference

Introduction to Visual SLAM From Theory to Practice(Xiang Gao and Tao Zhang; 2021) 을 정리하였습니다.

1 Chapter 1 INTRODUCTION TO SLAM

1.1 1.3 Mathematical Formulation of SLAM Problems

로봇이 unknown environment에서 센서를 통해 자신의 위치를 알려고 한다. 이 때 로봇이 직접적으로 얻을 수 있는 데이터는, 센서로 부터 얻은 observation 과 구동장치에서 사용한 Control 정보가 전부이다. 센서도 연속적으로 데이터를 얻는 것이 아니라, discrete time 공간에서 주변 환경에 대한 정보를 sampling 한다.

이렇게 제한적인 환경에서 로봇은 어떻게 자신의 위치를 파악(추정) 할 수 있는가? mathematical Formulation 을 통해 문제를 명확히 하자.

Notation

위에서 센서데이터가 discrete 하게 얻어진다고 했는데, 이 때의 time step을 $1, \cdots, k$ 라 하자. 센서 데이터가 sampling 될 때의 robot position 을 $x_1, \cdots x_k$ 라 하자. 실제 unknown env 에서는 초기에 Landmark가 존재하지 않지만 Landmark가 이미 존재하는 상황이라 가정하자. 이 때, 센서를 통해 로봇에서의 상대적인 position을 관측할 수 있는 Landmark의 position 을 $y_1, \cdots y_N$ 이라 하자. 이 경우 맵에는 N개의 랜드마크가 존재하게 된다. SLAM 알고리즘을 돌리면 실시간으로 Landmark가 추가된다.

Formulation

Motion / observation equation을 formulation하자. Motion equation은 time step이 $k-1$ 에서 k 로 바뀔 때, 로봇의 위치 x 가 어떻게 변화하는지 기술하는 식이다. Observation equation은 로봇이 x_k 위치에

있을때 y_j landmark을 관측하는 현상을 기술한다.

- motion equation

$$x_k = f(x_{k-1}, u_k, w_k) \tag{1}$$

여기서 u_k 는 input control, w_k 는 noise를 의미한다. function f 는 사용하는 모델에 따라 바뀔수도 있다. 노이즈 w_k 를 추가하여 stochastic model로 표현하였다.

- observation equation

$$z_{k,j} = h(y_j, x_k, v_{k,j}) \tag{2}$$

여기서 z 는 관측 데이터, v 는 노이즈를 의미한다.

2 Chapter 2. 3D Rigid Body Motion

이 챕터에서는 3차원 공간에서 강체의 움직임을 설명하는 방법에 대해 다룬다. 3차원 강체의 움직임은 rotation 과 translation으로 구성되는데, 회전은 rotation matrix, quaternions, Euler angles 와 같은 다양한 방법으로 표현 가능하다.

2.1 Rotation Matrix

2.1.1 Points, Vectors, and Coordinate Systems

3차원 공간에 있는 점을 표현하기 위해서는 3d vector만 필요하다. 하지만 3차원 강체의 경우 position 과 orientation을 모두 표현해야 한다.

수학적으로 강체의 움직임을 표현하기에 앞서, 간단한 선형대수학 개념을 다루고 넘어가자.

1. $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ base로 표현되는 좌표계에서 vector $\mathbf{a}$ 의 coordinate가 $\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$ 일 때, 다음이 성립한다.

$$\mathbf{a} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3. \tag{3}$$

2. 두 벡터 $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$ 사이의 내적 inner product는 아래와 같다.

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b} = \sum_{i=1}^{3} a_i b_i = |\mathbf{a}|\,|\mathbf{b}| \cos\langle \mathbf{a}, \mathbf{b} \rangle, \tag{4}$$

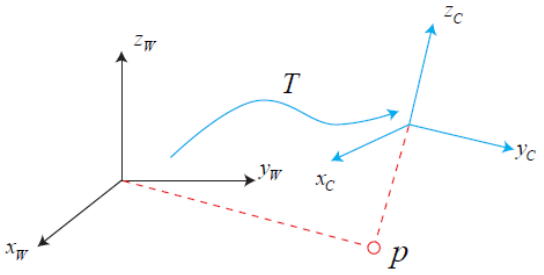

Figure 1: Coordinate transform. For the same vector $\mathbf{p}$, its coordinates in the world $\mathbf{p}_W$ and the coordinates in the camera system $\mathbf{p}_C$ are different. This transformation relationship is described by the transform matrix $\mathbf{T}$.

3. $\langle \mathbf{a}, \mathbf{b} \rangle$ 가 벡터 $\mathbf{a}, \mathbf{b}$ 사이의 각도를 표현할 때, 외적 outer product은 아래와 같다.

$$\mathbf{a} \times \mathbf{b} = \begin{Vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{Vmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \mathbf{b} \triangleq \mathbf{a}^{\wedge} \mathbf{b}. \tag{5}$$

여기서 $^{\wedge}$ operator는 vector $\mathbf{a}$ 를 skew-symmetric matrix 형태로 변형시킴을 의미한다. 아래와 같이$^{\wedge}$ 을 정의하면 $\mathbf{a} \times \mathbf{b} = \mathbf{a}^{\wedge} \mathbf{b}$ 이 성립한다.

$$\mathbf{a}^{\wedge} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{6}$$

2.1.2 Euclidean Transforms Between Coordinate Systems

좌표계 변환이란, point $\mathbf{p}$ 가 한 좌표계에서는 $\mathbf{p}_c$ 로 표현되고, 다른 좌표계에서는 $\mathbf{p}_w$ 로 표현 될 때, $\mathbf{p}_c$ 와 $\mathbf{p}_w$ 사이의 변환을 기술하는 방법이다. Fig. 1 에서 이를 잘 표현하고 있다. 여러 좌표계 변환중에서 우리가 관심있는 것은 rotation과 translation으로 표현되는 rigid body motion 이다. rigid body motion 에서 벡터의 길이와 , 벡터 사이의 각도는 변화하지 않는다.

world coordinate 에서 두 좌표계가 각각 $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$, $(\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3)$ unit-length orthogonal base로 표현된다고 해보자. 그리고 각 좌표계가 같은 점을 $[a_1, a_2, a_3]^{\mathrm{T}}$, $[a'_1, a'_2, a'_3]^{\mathrm{T}}$ 으로 표현하고 있다고 해보자. 그러면 다음 식이 성립한다.

$$[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3] \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix}. \tag{7}$$

양변에 $\begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix}$ 을 multiplication 하면, 다음이 성립한다.

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{e}_1^T\mathbf{e}_1' & \mathbf{e}_1^T\mathbf{e}_2' & \mathbf{e}_1^T\mathbf{e}_3' \\ \mathbf{e}_2^T\mathbf{e}_1' & \mathbf{e}_2^T\mathbf{e}_2' & \mathbf{e}_2^T\mathbf{e}_3' \\ \mathbf{e}_3^T\mathbf{e}_1' & \mathbf{e}_3^T\mathbf{e}_2' & \mathbf{e}_3^T\mathbf{e}_3' \end{bmatrix}}_{\text{rotation matrix}} \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} \triangleq \mathbf{R}\mathbf{a}'. \tag{8}$$

중간에 있는 matrix를 rotation matrix $\mathbf{R}$ 이라 부른다. unit-length orthogonal base 사이의 product이기 때문에, rotation matrix는 determinant 가 1인 orthogonal matrix라 부른다. 선형대수학에서는 determinant 가 1 인 n dimensional rotation matrix를 아래와 같이 정의한다.

$$\mathrm{SO}(n) = \{\mathbf{R} \in \mathbb{R}^{n\times n} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\}. \tag{9}$$

$\mathrm{SO}(n)$ 는 *special orthogonal group* 의 약자이다.

만약 opposite rotation 적용하고 싶다면 아래 식을 사용하면 된다. 은 아래와 같다.

$$\mathbf{a}' = \mathbf{R}^{-1}\mathbf{a} = \mathbf{R}^T\mathbf{a}. \tag{10}$$

rigid body motion , 즉 Euclidean transformation은 rotation 외에도 translation 이 존재한다.

vector $\mathbf{a}$ 에 rotation $\mathbf{R}$ 과 translation $\mathbf{t}$ 를 적용하면, 변환 후 벡터 $\mathbf{a}'$ 는 아래와 같이 표현된다.

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t}, \tag{11}$$

좌표계 변환을 명확하게 명시하기 위해, 다음과 같은 subscript를 고려하자.

1. vector $\mathbf{a}_1$: 좌표계 1 에서의 좌표
2. rotation matrix $\mathbf{R}_{12}$: 좌표계 2에서 1로 변환하는 rotation
3. translation vector $\mathbf{t}_{12}$: 좌표계 2에서 1로 변환하는 translation

위와 같은 notation 을 고려하면, 위 변환식은 아래와 같이 바뀐다.

$$\mathbf{a}_1 = \mathbf{R}_{12}\mathbf{a}_2 + \mathbf{t}_{12}. \tag{12}$$

2.1.3 Transform Matrix and Homogeneous Coordinates

위의 변환식은 선형이 아니기 때문에, 선형으로 표현할 필요가 있다. 선형으로 표현하지 않을 경우, 변환이 중첩될 때 식이 다소 지저분해진다. 예를 들어 $\mathbf{R}_1, \mathbf{t}_1$ 과 $\mathbf{R}_2, \mathbf{t}_2$ 변환을 중첩한다고 가정해보자. 그러면

아래식이 성립한다.

$$\mathbf{b} = \mathbf{R}_1\mathbf{a} + \mathbf{t}_1, \quad \mathbf{c} = \mathbf{R}_2\mathbf{b} + \mathbf{t}_2.$$

a to **c** 사이의 변환으로 표기하면 아래와 같다.

$$\mathbf{c} = \mathbf{R}_2\left(\mathbf{R}_1\mathbf{a} + \mathbf{t}_1\right) + \mathbf{t}_2.$$

두 번만 중첩되어도 식이 상당히 지저분해진다.따라서 homogeneous coordinate를 도입해서 transformation matrix로 변환을 표현한다.

$$\begin{bmatrix} \mathbf{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix} \triangleq \mathbf{T} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}. \tag{13}$$

$\tilde{\mathbf{a}}$ 을 사용해서 homogeneous coordinate임을 명시할 예정이다. transformation matrix로 표현하면 변환이 중첩되어도 깔끔해진다.

$$\tilde{\mathbf{b}} = \mathbf{T}_1\tilde{\mathbf{a}}, \ \tilde{\mathbf{c}} = \mathbf{T}_2\tilde{\mathbf{b}} \quad \Rightarrow \tilde{\mathbf{c}} = \mathbf{T}_2\mathbf{T}_1\tilde{\mathbf{a}}. \tag{14}$$

transformation matrix $\mathbf{T}$ 의 좌상단은 rotation matrix, 우상단은 translation vector, 좌하단은 $\mathbf{0}$ vector , 우하단은 1로 구성되어 있다. 이러한 matrix를 special Euclidean group이라 부른다.

$$\mathrm{SE}(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} | \mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{15}$$

SE(3)의 inverse는 아래와 같다.

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{16}$$

위와 마찬가지로 좌표계를 명확하게 명시하기 위해 동일한 subscript notation을 사용한다. 즉 $\mathbf{T}_{12}$ 는 좌표계 2에서 1로 변환하는 transformation matrix이다. 뒤의 과정에서 homogeneous form과 non-homogeneous form이 섞여서 사용될 예정인데, transformation matrix 가 적용되면 (e.g. $\mathbf{Ta}$) homogenous form, rotation matrix가 적용되면 (e.g. $\mathbf{Ra}$) non-homogeneous form 으로 생각하면 된다.

2.2 Rotation Vectors and the Euler Angles

2.2.1 Rotation Vectors

rotation Matrix를 사용하여 rotation을 표현 할 경우 motion은 4×4transformation matrix 로 표현된다. 하지만 3d rigid body motion은 6-DOF 이기 때문에 다음과 같은 문제가 발생한다.

1. SO(3) 는 9 개의 quantities를 갖지만, 3D rotation은 3 DOF 이다. 따라서 불필요한 표현들이 늘어

나게 된다. 이와 비슷하게 transformation matrix는 6DOF이지만, 16개의 quantities를 가진다.

2. rotation matrix는 determinant가 1인 orthogonal matrix여야 한다. 이러한 비선형제약조건은 rotation/transform matrix의 값을 추정하거나 optimize 할 때 어려움을 야기한다.

따라서 rotation과 translation을 간단하게 표기하기 위해, rotation vector 를 사용한다. rotation vector는 direction 이 axis of rotation과 일치하고, 길이가 angle of rotation 인 벡터이다. (also called angle-axis / axis-angle) 이 표기법을 통해 rotation을 3차원 벡터로 표현하였으므로 motion은 rotation vector (3d vector)와 translation vector(3d vector), 즉 6d vector로 표현된다.

다음으로는 Rotation matrix $\mathbf{R}$ 과 rotation vector $\theta\mathbf{n}$ (unit-length vector인 rotation axis $\mathbf{n}$, roataion angleθ) 사이의 변환에 대해 알아보자. rotation vector에서 Rotation Matrix로의 변환은 Rodrigues' formula로 알려져 있다.

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\,\mathbf{n}\mathbf{n}^T + \sin\theta\mathbf{n}^\wedge. \tag{17}$$

반대로의 변환은, Rodrigues' formula에서 유도 가능하다. 양변에 trace를 적용하면 ,

$$\begin{aligned} \mathrm{tr}\,(\mathbf{R}) &= \cos\theta\,\mathrm{tr}\,(\mathbf{I}) + (1 - \cos\theta)\,\mathrm{tr}\left(\mathbf{n}\mathbf{n}^T\right) + \sin\theta\,\mathrm{tr}(\mathbf{n}^\wedge) \\ &= 3\cos\theta + (1 - \cos\theta) \\ &= 1 + 2\cos\theta. \end{aligned} \tag{18}$$

θ 를 좌변에 몰아서 정리하면,

$$\theta = \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right). \tag{19}$$

rotation axis $\mathbf{n}$ 은 rotation을 적용한 후에도 변화하지 않는 성질이 있기 때문에, 다음이 성립한다.

$$\mathbf{R}\mathbf{n} = \mathbf{n}. \tag{20}$$

따라서 axis $\mathbf{n}$ 은 matrix $\mathbf{R}$ 의 eigenvalue 1에 대응하는 eigenvector라는 사실도 알 수 있다.

2.2.2 Euler Angles

다음으로는 Euler angle로 rotation을 표현해보자. Euler angle은 rotation vector와 마찬가지로 3d vector로 회전을 표현하는 방식이다. 다만 rotation vector로 회전을 표현 할 경우 직관적으로 어떤 회전인지 파악하기 힘든 반면, Euler angle은 직관적으로 이해 할 수 있는 표기법이다.

기본적으로 Euler angle은 3개의 primal axes를 사용하여, 전체 회전을 각 축에 대한 회전으로 분해하여 표현하는 방법이다. primal axes을 어떻게 잡느냐에 따라 다양한 정의가 존재한다. 예를 들어 X axis, Y axis, Z axis 순서대로 회전한다면 XYZ order 로 회전하였다고 표현한다. 이외에도 ZYZ , ZYX order로 회전을 표현하는 것도 가능하다. 이외에도 fixed axis로 회전하였는지, axis after rotation으로 회전하였는지

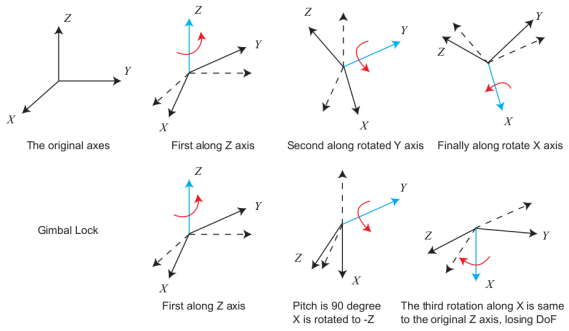

Figure 2: Euler angles. The top is defined for the ZYX order (ypr order). The bottoms shows when pitch=90°, the third rotation is using the same axis as the first one, causing the system to lose a degree of freedom. If you don't understand the gimbal lock, please take a look at the related videos and it will be more convenient to understand.

기술해야 한다.

가장 많이 사용하는 Euler Angle의 정의는 ZYX axis 로 회전하는 것이다. 이는 yaw-pitch-roll angle 로 분해하는 것과 동일한 정의이다. ZYX axis 로 회전하는 예시가 Fig. 2에 나타나 있다.

Fig. 2에서 볼 수 있듯이 회전을 $[y, p, r]^T$ 형태로 표현하여 회전을 직관적으로 이해 할 수 있다.

하지만 Euler Angle은 Gimbal lock 이라는 큰 단점을 갖고 있다. ypr 로 회전을 표현할 때 pitch angle이 $\pm 90°$ 이 된다면, 첫 번째 와 세 번째 회전이 같은 회전축을 사용하게 되고, 시스템의 DOF가 2로 감소하게 된다. 이러한 문제를 singularity problem 이라 부르며, 3개의 real number 로 3d rotation을 표현할 경우 항상 발생하는 문제이다. 이러한 문제 때문에 Euler angle을 interpolation이나 iteration 과정에서 사용하지는 않는다.

2.3 Quaternions

위에서 rotation matrix, Euler angles, rotation vector와 같은 rotation 표기법에 대해 배웠다. 하지만 rotation matrix은 compact하지 못하다는 단점이 있었고, Euler angles/rotation vector는 compact하지만 singularity problem이 존재했다. 사실 3d vector로 rotation 을 표현할 경우 singularity probelm이 항상 생긴다는 것이 밝혀졌기 때문에, 이번 챕터에서는 Quaternion이라는 4d vector를 도입해 회전을 표현할 예정이다.

2D 복소평면를 생각해보면, complex multiplication을 사용해 rotation을 표현 할 수 있었다. 예를 들어 i 을 곱할 경우, 반시계 방향으로 $90°$ 회전을 표현할 수 잇었다. 이처럼 3D space rotation도 complex number quaternion을 사용해 표현 할 수 있다.

2d complex plane에 대해 좀 더 자세히 이야기 하고, 이 개념을 3d space에 적용하는 과정으로 설명하고자 한다. 2d complex plane에서 vector를 θ 만큼 회전하는 것은 $e^{i\theta}$ 를 곱하는 것과 동일하다. $e^{i\theta}$ 는 Euler equation에 의해 다음과 같이 표현된다.

$$e^{i\theta} = \cos\theta + i\sin\theta. \tag{21}$$

$e^{i\theta}$ 는 길이가 1인 복소수, unit-length complex number인데, 이와 유사하게 3D rotation 도 unit quaternion 으로 표현된다.

quaternion $\mathbf{q}$ 는 하나의 실수부와 세개의 허수부로 구성된다. 식으로 표현하면 아래와 같다.

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k, \tag{22}$$

i, j, k 가 quaternion의 허수부인데, 다음과 같은 관계를 가지고 있다.

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k, ji = -k \\ jk = i, kj = -i \\ ki = j, ik = -j \end{cases}. \tag{23}$$

quaternion은 아래와 같이 scalar 와 vector을 합친 형태로도 표현된다.

$$\mathbf{q} = [s, \mathbf{v}]^T, \quad s = q_0 \in \mathbb{R}, \quad \mathbf{v} = [q_1, q_2, q_3]^T \in \mathbb{R}^3,$$

여기서 s 를 Quaternion의 실수부(real part of the quaternion) 라 부르며, $\mathbf{v}$ 를 Quaternion의 허수부 (imaginary part of the Qauternion)이라 부른다. Qauternion의 허수부가 $\mathbf{0}$ 이면 real quaternion이라 부르며, 실수부가 0 이면 imaginary quaternion 이라 부른다.

2.3.1 Quaternion Operations

Quaternion은 복소수와 비슷한 형태로 연산이 정의되어 있다. $\mathbf{q}_a, \mathbf{q}_b$ 를 사용해서 Quaternion 연산에 대해 알아보자.

$$\mathbf{q}_a = [s_a, \mathbf{v}_a]^{\mathrm{T}} = s_a + x_a i + y_a j + z_a k,$$

$$\mathbf{q}_b = [s_b, \mathbf{v}_b]^{\mathrm{T}} = s_b + x_b i + y_b j + z_b k.$$

1. **Addition and subtraction**.

$$\mathbf{q}_a \pm \mathbf{q}_b = [s_a \pm s_b, \mathbf{v}_a \pm \mathbf{v}_b]^T. \tag{24}$$

2. **Multiplication**. i, j, k 사이의 관계를 생각해보면 아래가 성립한다. (scalar form)

$$\begin{aligned}\mathbf{q}_a\mathbf{q}_b = & s_a s_b - x_a x_b - y_a y_b - z_a z_b \\ & + (s_a x_b + x_a s_b + y_a z_b - z_a y_b)\, i \\ & + (s_a y_b - x_a z_b + y_a s_b + z_a x_b)\, j \\ & + (s_a z_b + x_a y_b - y_a x_b + z_a s_b)\, k.\end{aligned} \tag{25}$$

(vector form)

$$\mathbf{q}_a\mathbf{q}_b = \left[s_a s_b - \mathbf{v}_a^T\mathbf{v}_b, s_a\mathbf{v}_b + s_b\mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b\right]^T. \tag{26}$$

아쉽게도 vector form 마지막 항에서 cross product가 존재하기 때문에, $\mathbf{v}_a$ 와 $\mathbf{v}_b$ 가 평행하지 않다면 교환법칙은 성립하지 않는다.

3. **Length**.

$$\|\mathbf{q}_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}. \tag{27}$$

따라서,

$$\|\mathbf{q}_a\mathbf{q}_b\| = \|\mathbf{q}_a\|\|\mathbf{q}_b\|. \tag{28}$$

4. **Conjugate**. quaternion의 Conjugate는 허수부에 $-$을 곱한 형태이다.

$$\mathbf{q}_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\mathbf{v}_a]^T. \tag{29}$$

따라서 다음이 성립한다.

$$\mathbf{q}^*\mathbf{q} = \mathbf{q}\mathbf{q}^* = [s^2 + \mathbf{v}^T\mathbf{v}, \mathbf{0}]^T. \tag{30}$$

위를 통해 conjugate와의 곱이 length of quaternion의 제곱과 동일함을 알 수 있다.

5. **Inverse**. inverse of quaternion은 아래와 같다.

$$\mathbf{q}^{-1} = \mathbf{q}^*/\|\mathbf{q}\|^2. \tag{31}$$

따라서 quaternion과 그 inverse의 product는 real quaternion $\mathbf{1}$ 이 된다.

$$\mathbf{q}\mathbf{q}^{-1} = \mathbf{q}^{-1}\mathbf{q} = \mathbf{1}. \tag{32}$$

만약 $\mathbf{q}$ 가 unint quaternion이라면 다음이 성립한다.

$$(\mathbf{q}_a\mathbf{q}_b)^{-1} = \mathbf{q}_b^{-1}\mathbf{q}_a^{-1}. \tag{33}$$

6. **Scalar Multiplication**. 일반적인 vector 와 유사하게 scalar multiplication이 성립한다.

$$k\mathbf{q} = [ks, k\mathbf{v}]^T. \tag{34}$$

2.3.2 Use Quaternion to Represent a Rotation

지금 까지는 quaternion이 무엇인지 설명했다. 이제 quaternion으로 회전을 표현해보자. 3d point $\mathbf{p} = [x, y, z]^T \in \mathbb{R}^3$ 가 회전으로 인해 $\mathbf{p}'$ 로 옮겨졌다고 해보자. 만약 Rotation Matrix $\mathbf{R}$ 로 회전이 표현되었다면 $\mathbf{p}' = \mathbf{R}\mathbf{p}$ 이 성립했을 것이다.

한번 unit quaternion $\mathbf{q}$ 를 사용하여 위 식을 표현해보자. 지금까지 quaterion 사이의 연산을 적용하였으므로, 3d point 들도 quaternion으로 변환하는 과정이 필요하다. 3d point의 좌표를 quaternion의 허수부에 대응시키자.

$$\mathbf{p} = [0, x, y, z]^T = [0, \mathbf{v}]^T.$$

이 때 회전된 점 $\mathbf{p}'$ 는 아래와 같이 표현된다.

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}. \tag{35}$$

다시 실수 형태의 좌표를 얻고 싶다면, quaternion $\mathbf{p}'$ 의 허수부만 취하면 된다.

2.3.3 Conversion of Quaternions to Other Rotation Representations

Quaternion과 다른 rotation representation 사이의 변환에 대해 알아보자. 변환에 앞서 quaternion multiplication 이 matrix multiplication 형태로 표현 될 수 있음을 알아야 한다. 아래와 같이 $^+$, $^\oplus$ symbol을 정의하자.

$$\mathbf{q}^+ = \begin{bmatrix} s & -\mathbf{v}^T \\ \mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix}, \quad \mathbf{q}^\oplus = \begin{bmatrix} s & -\mathbf{v}^T \\ \mathbf{v} & s\mathbf{I} - \mathbf{v}^\wedge \end{bmatrix}, \tag{36}$$

두 matrix 모두 4 ×4 이며, 두 심볼을 사용하여 quaternion multiplication을 표현 할 수 있다. $^+$ 의 경우,

$$\mathbf{q}_1^+\mathbf{q}_2 = \begin{bmatrix} s_1 & -\mathbf{v}_1^T \\ \mathbf{v}_1 & s_1\mathbf{I} + \mathbf{v}_1^\wedge \end{bmatrix} \begin{bmatrix} s_2 \\ \mathbf{v}_2 \end{bmatrix} = \begin{bmatrix} -\mathbf{v}_1^T\mathbf{v}_2 + s_1 s_2 \\ s_1\mathbf{v}_2 + s_2\mathbf{v}_1 + \mathbf{v}_1^\wedge\mathbf{v}_2 \end{bmatrix} = \mathbf{q}_1\mathbf{q}_2 \tag{37}$$

이와 유사하게 $^\oplus$ 도 다음이 성립한다.

$$\mathbf{q}_1\mathbf{q}_2 = \mathbf{q}_1^+\mathbf{q}_2 = \mathbf{q}_2^\oplus\mathbf{q}_1. \tag{38}$$

위의 $\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$ 의 회전 형태를 $^+$, $^\oplus$ symbol로 표현하면아래와 같다. (마지막 전개는 $\mathbf{p}_1^+\mathbf{1} = \mathbf{p}$ 임을 사용하면 된다.)

$$\begin{aligned}\mathbf{p}' &= \mathbf{q}\mathbf{p}\mathbf{q}^{-1} = \mathbf{q}^+\mathbf{p}^+\mathbf{q}^{-1} \\ &= \mathbf{q}^+\mathbf{q}^{-1^\oplus}\mathbf{p}.\end{aligned} \tag{39}$$

우변의 두 항을 matrix form으로 표현하면 아래와 같다.

$$\mathbf{q}^+\left(\mathbf{q}^{-1}\right)^\oplus = \begin{bmatrix} s & -\mathbf{v}^T \\ \mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix} \begin{bmatrix} s & \mathbf{v}^T \\ -\mathbf{v} & s\mathbf{I} + \mathbf{v}^\wedge \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0}^T & \mathbf{v}\mathbf{v}^T + s^2\mathbf{I} + 2s\mathbf{v}^\wedge + \left(\mathbf{v}^\wedge\right)^2 \end{bmatrix}. \tag{40}$$

$\mathbf{p}'$ 와 $\mathbf{p}$ 는 허수부만 존재하는 imaginary quaternion이기 때문에, 우하단 부분만 고려하면 다음 quaternion to rotation matrix 변환식이 얻어진다.

$$\mathbf{R} = \mathbf{v}\mathbf{v}^T + s^2\mathbf{I} + 2s\mathbf{v}^\wedge + \left(\mathbf{v}^\wedge\right)^2. \tag{41}$$

역변환은 양변의 trace를 적용하여 얻을 수 있다.

$$\begin{aligned}\mathrm{tr}(\mathbf{R}) &= \mathrm{tr}(\mathbf{v}\mathbf{v}^T) + 3s^2 + 2s \cdot 0 + \mathrm{tr}((\mathbf{v}^\wedge)^2) \\ &= v_1^2 + v_2^2 + v_3^2 + 3s^2 - 2(v_1^2 + v_2^2 + v_3^2) \\ &= (1 - s^2) + 3s^2 - 2(1 - s^2) \\ &= 4s^2 - 1.\end{aligned} \tag{42}$$

rotation matrix 와 rotation vector 사이의 변환도 위에서 다뤘기 때문에 적용하면 아래와 같다.

$$\begin{aligned}\theta &= \arccos\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right) \\ &= \arccos(2s^2 - 1),\end{aligned} \tag{43}$$

which means:

$$\cos\theta = 2s^2 - 1 = 2\cos^2\frac{\theta}{2} - 1, \tag{44}$$

and so we have:

$$\theta = 2\arccos s. \tag{45}$$

rotataion axis를 구하는 것은 더 쉽다. rotation axis 는 회전이 적용되어도 변화하지 않아야 한다. 따라서 $\mathbf{n} = \mathbf{q}^+\mathbf{q}^{-1^\oplus}\mathbf{n}$ 이 성립해야 한다. $[0, \mathbf{v}]^T$ 에 회전을 적용하면 $\mathbf{q}^+\mathbf{q}^{-1^\oplus}[0, \mathbf{v}]^T = [0, \mathbf{v}]^T$ 이다. 즉 quaternion 의 허수부만 취하면 된다. 다만 rotation axis는 normal length여야 하므로 normalize를 해줘야 한다. 따라서,

$$\begin{cases} \theta = 2\arccos q_0 \\ [n_x, n_y, n_z]^T = [q_1, q_2, q_3]^T / \sin\frac{\theta}{2} \end{cases} . \tag{46}$$

2.4 Affine and Projective Transformation

유클리드 변환 외에도 몇 가지 의미있는 변환이 존재한다.

1. **Similarity transformation**. similarity transformation 는 유클리드 변환에서 scale 에 관련된 변수, 1DOF 가 추가된 형태이다.

$$\mathbf{T}_S = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} . \tag{47}$$

   이러한 변환을 similarity transform group , Sim(3) 이라 부른다.

2. **Affine transformation**. affine transformation 은 아래와 같은 형태이다. $\mathbf{A}$ 가 invertible matrix 이기만 하면 된다. affine transformation은 orthogonal projection 으로 불리기도 한다.

$$\mathbf{T}_A = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} . \tag{48}$$

3. **Perspective transformation**. Perspective transformation 은 아래와 같은 형태이다.

$$\mathbf{T}_P = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{a}^T & v \end{bmatrix} . \tag{49}$$

   좌상단 $\mathbf{A}$ 은 invertible matrix 이며, homogeneous coordinate이기 때문에 $v \neq 0$ 여야 한다. homogenous coordinate임을 고려하면 15 DOF을 가진다.

table 1 에서는 위에서 언급한 변환들을 정리하고 있다.

Table 1: comparison of common transformation properties

Transform Name	Matrix Form	Degrees of Freedom	Invariance
Euclidean	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	6	Length, angle, volume
Similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	7	volume ratio
Affine	$\begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$	12	Parallelism, volume ratio
Perspective	$\begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{a}^T & v \end{bmatrix}$	15	Plane intersection and tangency

3 Ch3. Lie Group and Lie Algebra

ch2. 에서는 rigid body motion이 무엇인지 , 그리고 그 표현법(rotation matrix, rotation vector, quaternion ...) 들에 대해 다뤘다. SLAM 은 그 이름에서 알 수 있듯이, robot의 위치(SE3)를 추정(localization)하고 mapping(3d space point)하는 것을 목표로 한다. robot의 위치를 추정하는 과정에서 rotation의 제약조건 (SO3 ; $\mathbf{R}\mathbf{R}^T = \mathbf{I}$) 때문에, 최적화 과정에서 Lie Group and Lie Algebra 개념을 사용하게 된다. (단순히 생각해보면, In the last lecture, we introduced the description of rigid body motion in the three-dimensional world, including the rotation matrix, rotation vector, Euler angle, quaternion, and so on. We focused on the representation of rotation, but in SLAM, we have to estimate and optimize them in addition to the representation. Because the pose is unknown in SLAM, we need to solve the problem of which camera pose best matches the current observation. A typical way is to build it into an optimization problem, solving the optimal $\mathbf{R}, \mathbf{t}$ and minimizing the error.

rotation matrix는 constraint 가 존재하는 행렬이기 때문에, 최적화에서 rotation matrix를 바로 사용할 경우 여러가지 어려움이 야기된다. Lie group and Lie Algebra 간의 변환을 통해 pose estimation 을 constraint가 없는 최적화 문제로 바꿀수 있다.

3.0.1 Basics of Lie Group and Lie Algebra

우리가 관심있는 "3차원 공간의 rigid body motion"은 아래와 같은 SO(3), SE(3) 로 구성되어 있다.

$$\mathrm{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\}. \tag{50}$$

$$\mathrm{SE}(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} | \mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{51}$$

두 요소는 이름에서 부터 알 수 있듯이, Group(군) 이다. 다만 덧셈에 대해 닫혀 있지 않는 group인데 수학적으로 표기하면 다음과 같다. For any two rotation matrices $\mathbf{R}_1, \mathbf{R}_2$

$$\mathbf{R}_1 + \mathbf{R}_2 \notin \mathrm{SO}(3), \quad \mathbf{T}_1 + \mathbf{T}_2 \notin \mathrm{SE}(3). \tag{52}$$

하지만 곱셈(multiplication)에 대해서는 닫힌 group이다.

$$\mathbf{R}_1\mathbf{R}_2 \in \mathrm{SO}(3), \quad \mathbf{T}_1\mathbf{T}_2 \in \mathrm{SE}(3). \tag{53}$$

$$\begin{bmatrix} \mathbf{R_1} & \mathbf{t_1} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{R_2} & \mathbf{t_2} \\ \mathbf{0}^T & 1 \end{bmatrix} =$$

하나의 연산자에 대해서만 "잘 정의되어 있으면" group이라 부르는데, rotation matrix = SO3는 곱셈에 대해 닫혀있으므로 group이다. group에 대해 잘 이해가 되지 않더라도 일단 넘어가길 바란다. 바로 다음 subsection에서 group에 대해 다룰 예정이다.

3.0.2 Group

group 자체는 어려운 개념은 아니다. 수학에서 다루는 많은 구조가 group이다. "close (닫혀 있다)"는 초중등 수학과정에서 다뤘던 것으로 기억한다. 예를 들어 정수는 덧셈에 대해 닫혀있다. 정수인 any element 두 개를 선택해서 더해도, 정수가 된다. 예를 들어 $5 + 7 = 12$로, 정수 5와 7을 더한 값인 12는 정수이다. 이 짧은 예시에서 주목해야 할 개념은 "정수" 와 "덧셈" 이다. Group은 어떤 set과 operator이 같이 정의된 대수 구조를 의미한다. set A 와 operation $\cdot$ 로 구성된 group이라면 $G = (A, \cdot)$ 로 표기할 수도 있다. 다만 아무 set과 operator를 같이 정의하기만 하면 되는 것은 아니고, 위 예시처럼 closed 와 함께 다른 성질들도 보장한다면 group이라 부를 수 있다. Group이 만족해야 하는 성질은 다음과 같다.

1. **Closure**: $\forall a_1, a_2 \in A,\ a_1 \cdot a_2 \in A.$
2. **Combination**: $\forall a_1, a_2, a_3 \in A,\ (a_1 \cdot a_2) \cdot a_3 = a_1 \cdot (a_2 \cdot a_3).$
3. **Unit element**: $\exists a_0 \in A,$ s.t. $\forall a \in A,\ a_0 \cdot a = a \cdot a_0 = a.$
4. **Inverse element**: $\forall a \in A,\ \exists a^{-1} \in A,\ st\ a \cdot a^{-1} = a_0.$

가장 많이 예시로 사용되는 Group은 덧셈에 대한 정수군 $(\mathbb{Z}, +)$(the addition of integers) 와 the rational numbers with multiplication after removing 0 $(\mathbb{Q}\backslash 0, \cdot)$ 가 있다. 우리과 관심있어 하는 SO3와 SE3에 연관된 group은 다음과 같다.

- General Linear group GL(n). The invertible matrix of $n \times n$ with matrix multiplication.
- Special Orthogonal Group SO(n). Or the rotation matrix group, where SO(2) and SO(3) is the most common.
- Special Euclidean group SE(n). Or the n dimensional transformation described earlier, such as SE(2) and SE(3).

The group structure guarantees that the group's operations have very good properties. The group theory is the theory that studies the various structures and properties of the groups. Readers interested in group theory can refer to any of the modern algebra books. *Lie Group* refers to a group with continuous (smooth) properties. Discrete groups like the integer group $\mathbb{Z}$ have no continuous properties, so they are not Lie groups. And obviously, SO(n) and SE(n) are continuous in real space because we can intuitively imagine that a rigid body moving continuously in the space, so they are all Lie Groups. Since SO(3) and SE(3) are especially important for camera pose estimation, we mainly discuss

these two Lie groups. However, strictly discussing the concepts of "continuous" and "smooth" requires knowledge of analysis and topology. We don't want to write this book into a mathematics book, so only some important conclusions directly related to SLAM are introduced. If the reader is interested in the theoretical nature of Lie Groups, please refer to books like [**?**].

We usually have two ways to introduce the Lie Groups or Lie Algebras. The first is to directly introduce Lie group and Lie algebra and then present to the reader that each Lie group corresponds to a Lie algebra. But, in this case, the reader may think that Lie algebra seems to be a symbol that jumps out with no reason and does not know its physical meaning. So, I will take a little time to draw the Lie algebra from the rotation matrix, similar to the way of [**?**] and [**?**]. Let's start with the simpler SO(3), leading to the Lie algebra $\mathfrak{so}(3)$ above SO(3).

3.0.3 Introduction of the Lie Algebra

Consider an arbitrary rotation matrix $\mathbf{R}$, we know that it satisfies:

$$\mathbf{R}\mathbf{R}^T = \mathbf{I}. \tag{54}$$

Now, we say that $\mathbf{R}$ is the rotation of a camera that changes continuously over time, which is a function of time: $\mathbf{R}(t)$. Since it is still a rotation matrix, we have

$$\mathbf{R}(t)\mathbf{R}(t)^T = \mathbf{I}.$$

Deriving time on both sides of the equation yields (we use $\dot{\mathbf{R}}$ to represent the derivative of $\mathbf{R}$ on time t, just like many other control books):

$$\dot{\mathbf{R}}(t)\mathbf{R}(t)^T + \mathbf{R}(t)\dot{\mathbf{R}}(t)^T = 0.$$

Move the second term to right and commute the matrices by using the transposed relation:

$$\dot{\mathbf{R}}(t)\mathbf{R}(t)^T = -\left(\dot{\mathbf{R}}(t)\mathbf{R}(t)^T\right)^T. \tag{55}$$

It can be seen that $\dot{\mathbf{R}}(t)\mathbf{R}(t)^T$ is a *skew-symmetric* matrix. Recall that we introduced the $\wedge$ symbol in the cross product formula (**??**), which turns a vector into a skew-symmetric matrix. Similarly, for any skew-symmetric matrix, we can also find a unique vector corresponding to it. Let this operation be

represented by the symbol $^\vee$:

$$\mathbf{a}^\wedge = \mathbf{A} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}, \quad \mathbf{A}^\vee = \mathbf{a}. \tag{56}$$

So, since $\dot{\mathbf{R}}(t)\mathbf{R}(t)^T$ is a skew-symmetric matrix, we can find a three-dimensional vector $\boldsymbol{\phi}(t) \in \mathbb{R}^3$ corresponds to it:

$$\dot{\mathbf{R}}(t)\mathbf{R}(t)^T = \boldsymbol{\phi}(t)^\wedge.$$

Right multiply with $\mathbf{R}(t)$ on both sides. Since $\mathbf{R}$ is an orthogonal matrix, we have:

$$\dot{\mathbf{R}}(t) = \boldsymbol{\phi}(t)^\wedge \mathbf{R}(t) = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \mathbf{R}(t). \tag{57}$$

It can be seen that we can take the time derivative of a ration matrix just by multiplying a $\boldsymbol{\phi}^\wedge(t)$ matrix on the left. Consider at time $t_0 = 0$ that the rotation matrix is $\mathbf{R}(0) = \mathbf{I}$. According to the derivative definition, we use the first-order Taylor expansion around $t = t_0$ to write $\mathbf{R}(t)$ as:

$$\begin{aligned} \mathbf{R}(t) &\approx \mathbf{R}(t_0) + \dot{\mathbf{R}}(t_0)(t - t_0) \\ &= \mathbf{I} + \boldsymbol{\phi}(t_0)^\wedge (t). \end{aligned} \tag{58}$$

We see that $\boldsymbol{\phi}$ reflects the derivative of $\mathbf{R}$, so it is called the *tangent space* near the origin of SO(3). The above formula is a differential equation for $\mathbf{R}$, and with the initial value $\mathbf{R}(0) = \mathbf{I}$, we have solution like:

$$\mathbf{R}(t) = \exp(\boldsymbol{\phi}_0^\wedge t). \tag{59}$$

where we note $\boldsymbol{\phi}(t_0) = \boldsymbol{\phi}_0$.

The reader can verify that the above equation holds for both the differential equation and the initial value. This means that around $t = 0$, the rotation matrix can be calculated from $\exp(\boldsymbol{\phi}_0^\wedge t)$[1]. We see that the rotation matrix $\mathbf{R}$ is associated with another skew-symmetric matrix $\boldsymbol{\phi}_0^\wedge t$ through an exponential relationship. But what is the exponential of a matrix? Here we have two questions that need to be clarified:

1. Given $\mathbf{R}$ at a certain moment, we can find a $\boldsymbol{\phi}$ that describes the local derivative relationship of $\mathbf{R}$. How are they correlated with each other? We will say that $\boldsymbol{\phi}$ corresponds to the Lie algebra

[1] At this point we have not explained what this exp means and how it works. We will talk about its definition and calculation process right after this section.

$\mathfrak{so}(3)$ on SO(3);

2. Second, when a vector $\boldsymbol{\phi}$ is given, how is $\exp(\boldsymbol{\phi}^\wedge)$ calculated? Conversely, given $\mathbf{R}$, is there an opposite operation to calculate $\boldsymbol{\phi}$? In fact, this is the exponential/logarithmic mapping between Lie group and Lie algebra.

Let's solve these two problems below.

3.0.4 The Definition of Lie Algebra

Now let's give the strict definition of Lie Algebra. Each Lie group has a Lie algebra corresponding to it. Lie algebra describes the local structure of the Lie group around its origin point, or in other words, is the tangent space. The general definition of Lie algebra is listed as follows:

A Lie algebra consists of a set $\mathbb{V}$, a scalar field $\mathbb{F}$, and a binary operation $[,]$. If they satisfy the following properties, then $(\mathbb{V}, \mathbb{F}, [,])$ is a Lie algebra, denoted as $\mathfrak{g}$.

1. **Closure**: $\forall \mathbf{X}, \mathbf{Y} \in \mathbb{V}; [\mathbf{X}, \mathbf{Y}] \in \mathbb{V}$.
2. **Bilinear composition**: $\forall \mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{V}; a, b \in \mathbb{F}$, we have:

$$[a\mathbf{X} + b\mathbf{Y}, \mathbf{Z}] = a[\mathbf{X}, \mathbf{Z}] + b[\mathbf{Y}, \mathbf{Z}], \quad [\mathbf{Z}, a\mathbf{X} + b\mathbf{Y}] = a[\mathbf{Z}, \mathbf{X}] + b[\mathbf{Z}, \mathbf{Y}].$$

3. **Reflexive** [2]: $\forall \mathbf{X} \in \mathbb{V}; [\mathbf{X}, \mathbf{X}] = \mathbf{0}$.
4. **Jacobi identity**: $\forall \mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{V}; [\mathbf{X}, [\mathbf{Y}, \mathbf{Z}]] + [\mathbf{Z}, [\mathbf{X}, \mathbf{Y}]] + [\mathbf{Y}, [\mathbf{Z}, \mathbf{X}]] = \mathbf{0}$.

The binary operations $[,]$ are called *Lie brackets*. At first glance, we require a lot of properties about the Lie bracket. Compared to the simpler binary operations in the group, the Lie bracket expresses the difference between the two elements. It does not require a combination law but requires the element and itself to be zero after the brackets. For example, the cross product $\times$ defined on the 3D vector $\mathbb{R}^3$ is a kind of Lie bracket, so $\mathfrak{g} = (\mathbb{R}^3, \mathbb{R}, \times)$ constitutes a Lie algebra. Readers can try to substitute the cross product into the four properties to verify the above conclusion.

4 Chapter 5 Nonlinear Optimization

- batch state estimation problem의 형태와 least-square problem으로 푸는 방법
- Gauss-Newton + Levenburg-Marquardt method
- ceres 와 g2o(비선형 최적화 라이브러리) 의 사용법과 활용

[2] Reflexive means that an element operates with itself results in zero.

4.1 State Estimation

4.1.1 From Batch State Estimation to Least-square

이전 챕터에서, SLAM은 이산시간에서 motion equation과 observation equation을 푸는 것으로 표현하였다.

$$\begin{cases} \mathbf{x}_k = \mathbf{f}\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k \\ \mathbf{z}_{k,j} = \mathbf{h}\left(\mathbf{y}_j, \mathbf{x}_k\right) + \mathbf{v}_{k,j} \end{cases}. \tag{60}$$

이 때 pose variable $\mathbf{x}_k$ 는 $\mathbf{T}_k \in \mathrm{SE}(3)$ 로 표현되어야 합니다. 일반적인 카메라를 센서로 사용할 경우, 관측 데이터z 는 다음과 같이 표현됩니다.

$$s\mathbf{z}_{k,j} = \mathbf{K}(\mathbf{R}_k\mathbf{y}_j + \mathbf{t}_k), \tag{61}$$

$\mathbf{z}$ 는 이미지속 Landmark의 픽셀위치이며, $\mathbf{R}_k$ 와 $\mathbf{t}_k$ 는 $\mathbf{x}_k$ 로 부터 얻어지는 Transformation Matrix 입니다. 카메라의 intrinsic matrix $\mathbf{K}$ 까지 multiply 해주면, 우변은 pixel coordinate로 변환 됩니다. 이 때 양 변 모두 2d homogeneous coordinate인데 Fig. 3 을 보면 우변의 homogeneous term 이 1이 아닐수도 있습니다. 따라서 이를 보정해주기 위해 좌변의 s term을 곱해주게 됩니다.

지금까지 한 내용을 정리해 보자면 1. Motion + Observation eq 를 모델링 2. Camera 를 사용하여 Landmark를 관측할 때 Observation eq 을 formulation 했습니다. 하지만 우리는 아직 observation eq 에서 noise term $\mathbf{v}_{k,j}$ 를 고려하지 않았다. 따라서 다음으로는 noise term에 대해 알아보자.

motion + observatopm eq 에서 두 노이즈 항 $\mathbf{w}_k, \mathbf{v}_{k,j}$ 을 평균이 0인 가우시안 분포로 모델링 해보자. 식으로 표현하면 다음과 같다.

$$\mathbf{w}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{R}_k\right), \mathbf{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_{k,j}\right), \tag{62}$$

여기서 $\mathcal{N}$는 가우시안 분포, $\mathbf{0}$는 분포의 평균, $\mathbf{R}_k, \mathbf{Q}_{k,j}$는 공분산 행렬을 의미한다. 우리의 목표는 위의 noise 들이 포함된 noise data $\mathbf{z}$ 와 $\mathbf{u}$ 로 부터 pose $\mathbf{x}$ 와 map $\mathbf{y}$ 를 추정하는 것이다. (* motion and observation eq에서 추정해야 하는 값 $\mathbf{x}$, $\mathbf{y}$ 그리고 noise $\mathbf{w}$, $\mathbf{v}$ 을 제외하면 , input은 $\mathbf{z}$ 와 $\mathbf{u}$ 밖에 없다)

일반적으로 state estimation 의 솔루션은 두 종류가 존재 한다. 첫 번째 방법은 incremental method 또는 filtering이라 부른다. 일반적으로 SLAM은 시간이 지나면서 취득한 데이터가 많아지므로, 현재 데이터로만 state을 추정한 다음에, 나중의 새로운 데이터를 통해 업데이트를 하는 방법이다. 가장 대표적인 방법은 Extended Kalman filter (EKF)이다. 두 번째 방법은 batch estimation 이라 부른다. 이 방법은 취득한 데이터를 저장한 다음, 그 데이터들과 가장 적합한 trajectory와 map을 찾는 method이다. 즉 time 0 k 까지의 모든 control + observation data을 저장하고 이 데이터를 토대로 전체 trajectory와 map을 추정하는 방법이다.

일반적으로 incremental method는 current moment $\mathbf{x}_k$ 의 state estimation 에 집중하며, 이전 state는 크게 고려하지 않는다. 반대로 batch method는 오랜 시간동안 최적화된 trajectory 를 얻는데 집중한다. 따라서 현재는 batch method가 slam의 주된 approach이다. batch method만 사용한다면 SfM 처럼 로봇이 환경에 대한 데이터를 수집한 후 offline에서 mapping을 하는 방법도 가능하다. 하지만 이 경우 real-time

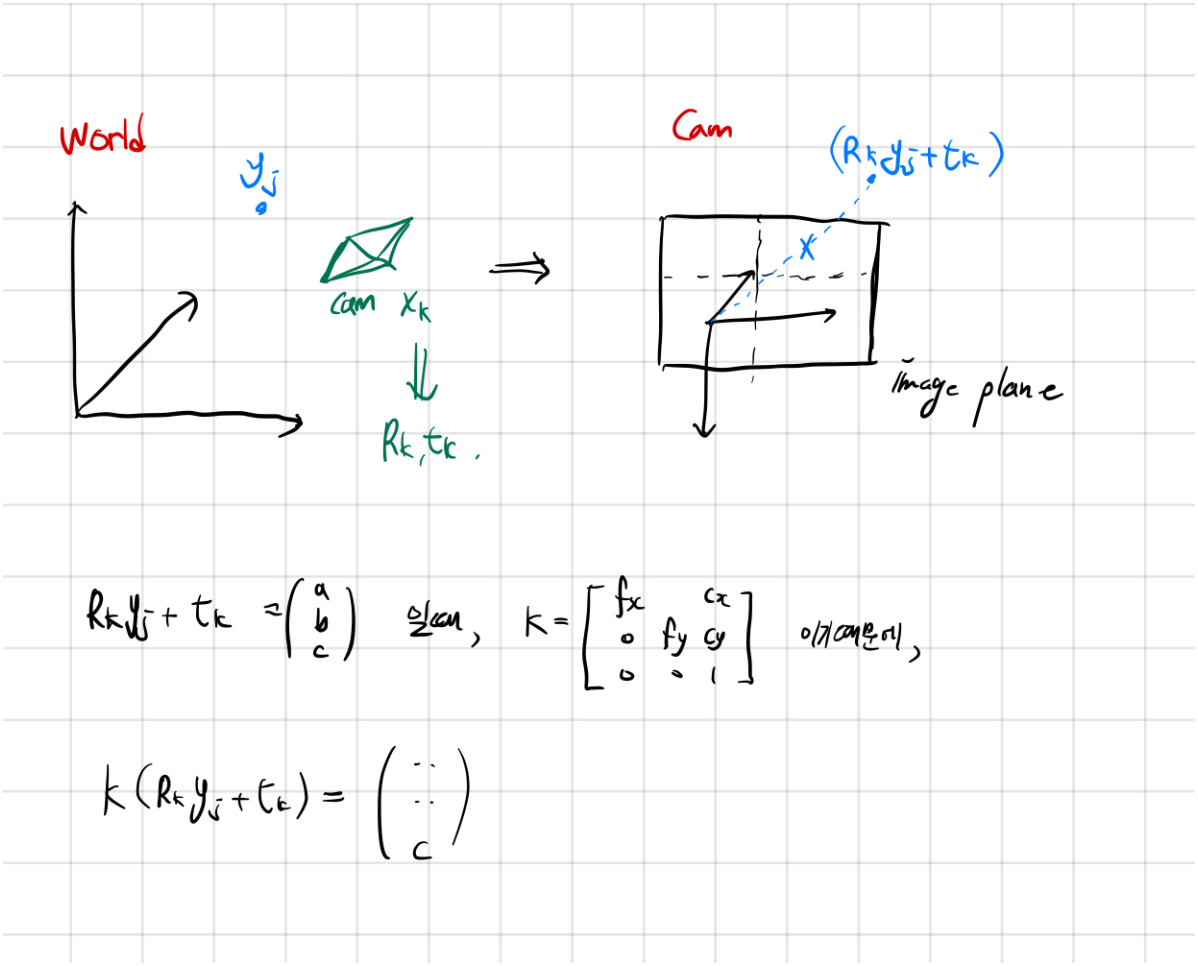

Figure 3: s term explain

으로 사용할 수 있는 방법은 아니다. 따라서 현재 SLAM을 로봇에 직접적으로 활용할때는 일종의 절충안을 사용한다. 예를 들어 과거의 trajectory는 일부만 수정하고, 최근에 추정한 일부 trajectory만 최적화하는 방법을 사용한다. 이러한 방법은 나중에 기술한 Sliding window estimation method와 연결된다.

이론적으로는 batch method가 이해하기 쉽기 때문에 이 section 에서는 non linear optimization 에 기반한 batch optimization method을 설명하고자 한다. filtering 기법(주로 Kalman filter)는 backend section 에서 다루고자 한다.

다시 motion and observation eq로 돌아와 보자. 우리는 state와 map을 stochastic model 로 모델링 하였으며, 각 noise는 가우시안 분포를 사용하기로 했다. camera senser를 사용할 경우 observation eq의 formulation도 기술하였으며, estimation은 batch method를 사용하기로 했다. batch method를 사용하기로 했으므로 time step 1 부터 N 까지의 모든 데이터를 고려하고, M 개의 landmark가 존재한다고 가정해보자. 수식으로 표현하면 다음과 같다.

$$\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}, \quad \mathbf{y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}.$$

마찬가지로 아랫첨자가 없는 $\mathbf{u}$ 와 $\mathbf{z}$ 는 time step 1 N 의 모든 control input 또는 observation data을 의미한다. 이러한 notation 하에서 로봇의 state estimation은 다음 조건부 확률 분포을 찾는 것과 동일하다.

$$P(\mathbf{x}, \mathbf{y}|\mathbf{z}, \mathbf{u}). \tag{63}$$

만약 control input을 모른다면 아래의 조건부 확률분포를 찾는 문제로 바뀌며, 이러한 케이스를 SfM이라 부른다.

$$P(\mathbf{x}, \mathbf{y}|\mathbf{z}) \tag{64}$$

conditional pdf을 추정하기 위해 Bayes' rule을 적용해보자.

$$P\left(\mathbf{x}, \mathbf{y}|\mathbf{z}, \mathbf{u}\right) = \frac{P\left(\mathbf{z}, \mathbf{u}|\mathbf{x}, \mathbf{y}\right) P\left(\mathbf{x}, \mathbf{y}\right)}{P\left(\mathbf{z}, \mathbf{u}\right)} \propto \underbrace{P\left(\mathbf{z}, \mathbf{u}|\mathbf{x}, \mathbf{y}\right)}_{\text{likehood}} \underbrace{P\left(\mathbf{x}, \mathbf{y}\right)}_{\text{prior}}. \tag{65}$$

Fig. 4 을 Bayes' rule이라 부른다. 좌변을 Posterior, 우변의 각 term을 likelihood , prior, Marginalization 이라 부른다. 비선형 시스템에서 posterior distribution을 직접적으로 찾는 것은 어렵다. 하지만, posterior을 최대화 하는 optimal point를 찾는 것은 가능하다. 이 때 optimal point을 MAP (Maximize a Posterior)라 부른다. 수식으로 표현하면 아래와 같다.

$$(\mathbf{x}, \mathbf{y})^*_{\text{MAP}} = \arg\max P\left(\mathbf{x}, \mathbf{y}|\mathbf{z}, \mathbf{u}\right) = \arg\max P(\mathbf{z}, \mathbf{u}|\mathbf{x}, \mathbf{y}) P(\mathbf{x}, \mathbf{y}). \tag{66}$$

Bayes' rule의 분모는 추정할 state와 관련 없이, input으로 들어오는 값이므로 무시해도 상관없다. 베이즈 룰을 통해 posterior를 최대화 하는 것과 (Likelihood 와 Prior 의 곱)을 최대화 하는 것이 동일하다는 것을 알았다. 또 로봇 pose나 map 에 대한 사전 정보가 없다면, prior가 없는 것과 동일 하다. 이 경우 MAP

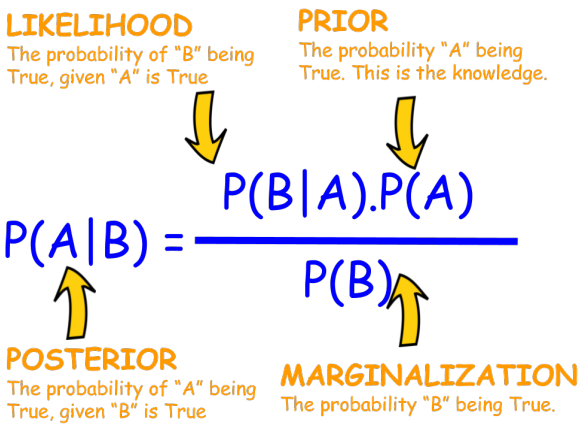

Figure 4: Bayes' rule

문제는 MLE(Maximize Likelihood Estimation)문제로 변하게 된다. MLE 문제를 수식화하면 다음과 같다.

$$(\mathbf{x}, \mathbf{y})^*{}_{\text{MLE}} = \arg\max P(\mathbf{z}, \mathbf{u}|\mathbf{x}, \mathbf{y}). \tag{67}$$

SLAM 문제에서 직관적으로 Likelihood는 "현재 포즈(ronot+randmark) 에서 어떤 관찰 데이터가 생성될 수 있는지"를 의미한다. 그리고 우리는 이미 observation data을 input으로 갖고 있기 때문에, MLE를 구하는 것은 "어떤 state가 observation data를 생성할 가능성이 가장 높은지"로 이해하면 된다.

4.1.2 Introduction to Least-squares

위 섹션에서 state estimation 문제를 MAP/MLE 문제로 formulation 했다. 이 섹션에서는 이를 least-square 방법으로 풀어보고자 한다. 위에서 이야기 했듯이, noise를 가우시안 분포로 가정해보자.

observation eq는 다음과 같았다.

$$\mathbf{z}_{k,j} = h\left(\mathbf{y}_j, \mathbf{x}_k\right) + \mathbf{v}_{k,j},$$

여기에 noise를 $\mathbf{v}_k \sim \mathcal{N}\left(\mathbf{0}, \mathbf{Q}_{k,j}\right)$ 으로 모델링 하면, observation eq는 다음과 같아진다.

$$P(\mathbf{z}_{j,k}|\mathbf{x}_k, \mathbf{y}_j) = N\left(h(\mathbf{y}_j, \mathbf{x}_k), \mathbf{Q}_{k,j}\right),$$

일반적인 Multivariate normal distribution $\mathbf{x} \sim \mathcal{N}(\mu, \mathbf{\Sigma})$ 은 다음과 같다.

$$P\left(\mathbf{x}\right) = \frac{1}{\sqrt{\left(2\pi\right)^N \det(\mathbf{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}\left(\mathbf{x}-\mu\right)\right). \tag{68}$$

양변에 negative logarithm을 적용하면 다음과 같다.

$$-\ln\left(P\left(\mathbf{x}\right)\right)=\underbrace{\frac{1}{2}\ln\left(\left(2\pi\right)^{N}\det\left(\mathbf{\Sigma}\right)\right)}_{\text{discarded}}+\frac{1}{2}(\mathbf{x}-\mu)^{T}\mathbf{\Sigma}^{-1}\left(\mathbf{x}-\mu\right). \tag{69}$$

log 함수는 단조 함수 이므로 maximize 문제는 minimize 문제로 바뀌게 된다. 그리고 첫 번째 항은 $\mathbf{x}$ 와 상관없으므로 minimize 문제에서 생략해도 상관 없다. 따라서 우리는 두 번재 항을 최소화 하는 $\mathbf{x}$ 을 찾으면, MLE 문제를 풀게 된다. 여기까지가 일반적인 MLE 문제였고, observation eq을 적용해보면 아래와 같다.

$$\begin{aligned}\left(\mathbf{x}_{k},\mathbf{y}_{j}\right)^{*}&=\arg\max\mathcal{N}(h(\mathbf{y}_{j},\mathbf{x}_{k}),\mathbf{Q}_{k,j})\\&=\arg\min\left(\left(\mathbf{z}_{k,j}-h\left(\mathbf{x}_{k},\mathbf{y}_{j}\right)\right)^{T}\mathbf{Q}_{k,j}^{-1}\left(\mathbf{z}_{k,j}-h\left(\mathbf{x}_{k},\mathbf{y}_{j}\right)\right)\right).\end{aligned} \tag{70}$$

argmin 내부의 항을 보면, noise $\mathbf{z}_{k,j}-h(\mathbf{x}_{k},\mathbf{y}_{j})$ 을 최소화 하는 quadratic form이라는 것을 알 수 있다. 이러한 quadratic form을 Mahalanobis distance라 부른다. 또는 $\mathbf{Q}_{k,j}^{-1}$에 의해 가중된 유클리드 거리($\mathcal{L}_2$-norm)로 간주할 수 있다. 여기서 $\mathbf{Q}_{k,j}^{-1}$는 information matrix라 부르며, 가우시안 공분산 행렬의 *inverse* 이다.

지금까지는 하나의 observation eq에 대해서만 다뤘으며, 이제 모든 observation eq을 고려해 보자. 먼저 control input과 observation이 독립이라고 가정하자. 그러면 다음과 같이 분포를 factorize할 수 있다. (* factorize 된 term의 prior가 왜 저렇게 표현되었는지 궁금하다면, 원래 motion and observation eq을 살펴보도록 하자)

$$P\left(\mathbf{z},\mathbf{u}|\mathbf{x},\mathbf{y}\right)=\prod_{k}P\left(\mathbf{u}_{k}|\mathbf{x}_{k-1},\mathbf{x}_{k}\right)\prod_{k,j}P\left(\mathbf{z}_{k,j}|\mathbf{x}_{k},\mathbf{y}_{j}\right), \tag{71}$$

모델과 실제 데이터 간의 오류를 아래와 같이 정의해보자.

$$\begin{aligned}\mathbf{e}_{u,k}&=\mathbf{x}_{k}-f\left(\mathbf{x}_{k-1},\mathbf{u}_{k}\right)\\\mathbf{e}_{z,j,k}&=\mathbf{z}_{k,j}-h\left(\mathbf{x}_{k},\mathbf{y}_{j}\right),\end{aligned} \tag{72}$$

우리는 motion and observation eq의 noise를 아래와 같이 모델링 했었다.

$$\mathbf{w}_{k}\sim\mathcal{N}\left(\mathbf{0},\mathbf{R}_{k}\right),\mathbf{v}_{k}\sim\mathcal{N}\left(\mathbf{0},\mathbf{Q}_{k,j}\right), \tag{73}$$

fractorize 된 $P\left(\mathbf{z},\mathbf{u}|\mathbf{x},\mathbf{y}\right)$ 방정식 양변에 negative logarithm 을 적용하면 argmax 문제는 다음과 같은 argmin 문제로 바뀐다.

$$\min J(\mathbf{x},\mathbf{y})=\sum_{k}\mathbf{e}_{u,k}^{T}\mathbf{R}_{k}^{-1}\mathbf{e}_{u,k}+\sum_{k}\sum_{j}\mathbf{e}_{z,k,j}^{T}\mathbf{Q}_{k,j}^{-1}\mathbf{e}_{z,k,j}. \tag{74}$$

위의 식을 통해 MLE 문제을 최적화를 사용한 least-square problem으로 풀 수 있다.

$\min J(\mathbf{x},\mathbf{y})$ 식을 보면 다음과 같은 특징을 발견할 수 있다.

- Objective function 은 여러 개의 error quadratic form의 summation 형태이다. 전체 state variable 은 high-D 이다. 하지만 각 error term은 단순하게 $\mathbf{x}_{k-1}, \mathbf{x}_k$ 또는 $\mathbf{x}_k, \mathbf{y}_j$ 에만 연관 되어 있다. 즉 sparse least-square problem이다. 자세한 내용은 Backend 챕터에서 다룬다.

- 일반적으로, 최적화를 위한 increment 는 unconstrained 이다. 하지만 pose의 경우 SE(3) 조건을 만족해야 한다.($\mathbf{R}^T\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$)(단순히 increment를 더해주는 방식을 사용하면 최적화 과정에서 pose 가 SE(3)을 만족하지 못하게 된다.)

- information matrix를 가중치로 사용하는 $\mathcal{L}_2-$norm을 Objective Function 으로 사용하였다. 만약 observation 이 매우 정확하다면, 공분산 $\mathbf{Q}_{k,j}$ 는 작아지고, information matrix $\mathbf{Q}_{k,j}^{-1}$ 는 크게 되어, 다른 term보다 높은 가중치를 가지게 된다.

예제를 하나 살펴본 뒤에, least-square problem을 비선형 최적화로 푸는 방법을 다루고자 한다.

4.2 Example: Batch State Estimation

다음과 같은 discrete time system을 생각해보자.

$$\begin{aligned} x_k &= x_{k-1} + u_k + w_k, & w_k &\sim \mathcal{N}(0, Q_k) \\ z_k &= x_k + n_k, & n_k &\sim \mathcal{N}(0, R_k) \end{aligned}, \tag{75}$$

state 는 1차원이라 가정하자. (x 축을 따라 앞뒤로 움직이는 자동차라 생각해도 좋다) batch에서 사용할 time step은 $k = 1, \ldots, 3$ 으로 설정하자.

초기 state x_0가 주어졌을 때, MLE을 통해 batch state estimation을 수행해보자. 다음과 같은 notation 을 생각해보자.

- batch state variable : $\mathbf{x} = [x_0, x_1, x_2, x_3]^T$
- batch observation : $\mathbf{z} = [z_1, z_2, z_3]^T$
- batch control : $\mathbf{u} = [u_1, u_2, u_3]^T$

이 때 MLE는 다음과 같다.

$$\begin{aligned} \mathbf{x}_{\text{map}}^* &= \arg\max P(\mathbf{x}|\mathbf{u}, \mathbf{z}) = \arg\max P(\mathbf{u}, \mathbf{z}|\mathbf{x}) \\ &= \prod_{k=1}^{3} P(u_k|x_{k-1}, x_k) \prod_{k=1}^{3} P(z_k|x_k), \end{aligned} \tag{76}$$

하나의 motion equation 에서 다음 분포를 얻을 수 있다.

$$P(u_k|x_{k-1}, x_k) = \mathcal{N}(x_k - x_{k-1}, Q_k). \tag{77}$$

observation equation도 유사하다.

$$P\left(z_k|x_k\right) = \mathcal{N}\left(x_k, R_k\right). \tag{78}$$

다음과 같이 error term을 정의하자.

$$e_{u,k} = x_k - x_{k-1} - u_k, \quad e_{z,k} = z_k - x_k, \tag{79}$$

이 때 negative logarithm을 적용한 objective function은 다음과 같다.

$$\min \sum_{k=1}^{3} e_{u,k}^T Q_k^{-1} e_{u,k} + \sum_{k=1}^{3} e_{z,k}^T R_k^{-1} e_{z,k}. \tag{80}$$

이 시스템을 하나의 quadratic term으로 표현해보자. $\mathbf{y} = [\mathbf{u}, \mathbf{z}]^T$ 로 정의하고, $\mathbf{H}$ 을 다음과 같이 정의하자.

$$\mathbf{H} = \left[\begin{array}{cccc} 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ \hline 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right], \tag{81}$$

이 때 에러는 다음과 같이 표현된다.

$$\mathbf{y} - \mathbf{H}\mathbf{x} = \mathbf{e} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}), \tag{82}$$

$$\mathbf{\Sigma} = \mathrm{diag}(Q_1, Q_2, Q_3, R_1, R_2, R_3). \tag{83}$$

MLE 문제는 다음과 같다.

$$\mathbf{x}_{\mathrm{map}}^* = \arg\min \mathbf{e}^T \mathbf{\Sigma}^{-1} \mathbf{e}, \tag{84}$$

이 때 solution은 다음과 같다.

$$\mathbf{x}_{\mathrm{map}}^* = (\mathbf{H}^T \mathbf{\Sigma}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{\Sigma}^{-1} \mathbf{y}, \tag{85}$$

4.3 Nonlinear Least-square Problem

가장 간단한 least-square Problem을 생각해보자.

$$\min_x F(x) = \frac{1}{2} ||f(x)||_2^2 \tag{86}$$

function f 가 smooth (미분가능한) 함수라면 다음을 만족하는 x 가 minimum이다

$$\frac{dF}{dx} = 0 \tag{87}$$

하지만 이 솔루션은 global min을 보장하지 못한다. local min이나 안장점의 최소점일 수 있다. 또 목적함수 F 가 미분할 수 없거나, 식 87 가 풀기 어려운 형태일 수도 있다. 이 경우 변수 x 를 반복적으로 업데이트 하여 최적 솔루션을 찾는 방법을 사용한다. 그 과정은 다음과 같다.

1. Give an initial value x_0.
2. For k-th iteration, we find an incremental value of Δx_k, such that the object function $||f(x_k + \Delta x_k)||_2^2$ reaches a smaller value.
3. If Δx_k is small enough, stop the algorithm.
4. Otherwise, let $x_{k+1} = x_k + \Delta x_k$ and return to step 2.

Δx_k 를 찾는 방법에 따라 Solution이 여러개로 나눠지는데, 이 책에서는 널리 사용되는 몇가지 방법만 소개하고자 한다.

4.3.1 The First and Second-order Method

Taylor expansion에서 first-order 와 second-order term만 고려하는 방식이다.

$x_k \in \mathbb{R}$ 일 때 second-order 까지 테일러 전개하면 다음과 같은 식이 얻어진다.

$$F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T \mathbf{H}(\mathbf{x}_k)\Delta\mathbf{x}_k, \tag{88}$$

이 때 $\mathbf{J}(\mathbf{x}_k)$ 는 $F(\mathbf{x})$ 의 $\mathbf{x}$ 에 대한 first derivative이며 (also called gradient, *Jacobi* Matrix, or *Jacobian*) (* 이 책에서는 $\mathbf{J}(\mathbf{x})$ 를 column vector로 작성하여 미소변화량 $\Delta\mathbf{x}$ 와 내적을 할경우 스칼라 값이 나오도록 설정하였습니다.) , $\mathbf{H}$ 는 second-order derivative (or *Hessian*) 입니다. 두 값 모두 $\mathbf{x} = \mathbf{x}_k$ 에서 얻어진 값입니다.

가장 간단하게 increment $\Delta\mathbf{x}_k$ 을 결정하는 방법은 first-order term만 사용하는 것이다. 예를 들어 gradient 방향의 반대로 increment를 잡으면 된다. 수식으로 표현하면 다음과 같다.

$$\Delta\mathbf{x}^* = -\mathbf{J}(\mathbf{x}_k). \tag{89}$$

일반적으로는 step length parameter λ 도 같이 사용한다. 이렇게 first order term만 사용하는 방법을 *steepest descent method* 라 부른다. 만약 increment 방향으로 이동하는 동안 1차 선형 근사가 유효하다면 Objective function은 감소할 것이다.

또 위와 같이 increment를 잡아서 최적화하는 과정이 실제로는 한번이 아니라 k번째 반복된다.

second-order term 도 고려하는 경우 increment는 다음 방정식을 사용해서 설정한다.

$$\Delta\mathbf{x}^* = \arg\min\left(F(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}\right). \tag{90}$$

$\arg\min$ 내부 항에서 $\Delta\mathbf{x}$ 의 도함수가 0이 되는 조건은 아래와 같다.

$$\mathbf{J} + \mathbf{H}\Delta\mathbf{x} = \mathbf{0} \Rightarrow \mathbf{H}\Delta\mathbf{x} = -\mathbf{J}. \tag{91}$$

위 선형 방정식을 풀어서 increment를 구할 수도 있다. 위 방법을 을 *Newton's method* 라 하나다. 근사지점 주변에서 Objective function이 1차 또는 2차 함수로 근사 가능하다면, 위 알고리즘은 유효할 것이다. 하지만 *steepest descent method* 은 지그재그 형태로 최적화 될 수 있으며, 이 경우 많은 step을 필요로 한다. *Newton's method* 의 경우 $\mathbf{H}$ 행렬을 계산하는데 시간이 오래 걸린다. 우리는 f의 테일러 확장을 계산하고 증분을 찾을 수 있는 한 1차 및 2차 방법 모두 매우 직관적이라는 것을 확인했습니다. 우리는 이 함수가 선형 또는 2차 함수처럼 보인다고 말합니다. 근사 함수의 최소값을 사용하여 원래 함수의 최소값을 추측할 수 있습니다. 원래의 목적 함수가 로컬에서 실제로 1차 또는 2차 함수처럼 보이는 한, 이 유형의 알고리즘은 항상 유효합니다(실제 대부분의 경우이기도 함). 그러나 이 두 가지 방법에도 단점이 있습니다. 최속강하법은 너무 욕심이 많고 지그재그 방식으로 이어지기 쉬우며 반복 횟수가 늘어난다. 그러나 Newton의 방법은 목적 함수의 $\mathbf{H}$ 행렬을 계산해야 하므로 문제가 큰 경우 시간이 많이 걸립니다. 일반적인 문제의 경우 quasi-Newton method가 더 좋은 성능을 내며, least-square probelm의 경우 *Gauss-Newton's method* 나 *Levernburg-Marquardt's method* 가 실용적이다.

4.4 The Gauss-Newton Method

최적화 목표가 다음과 같다고 하자.

$$\arg\min_{\mathbf{x}} \|f(\mathbf{x})\|^2 \tag{92}$$

$f(\mathbf{x})$ 에 대해 1차 Taylor expansion을 적용하자.

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\Delta\mathbf{x}. \tag{93}$$

increment는 $\|f(\mathbf{x} + \Delta\mathbf{x})\|^2$ 가 최소가 되도록 설정하는 것이 목표이다. 위의 근사를 사용하면, 다음 과 같이 increment가 설정된다.

$$\Delta\mathbf{x}^* = \arg\min_{\Delta\mathbf{x}} \frac{1}{2}\left\|f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T\Delta\mathbf{x}\right\|^2. \tag{94}$$

제곱항을 확장해보면 다음과 같다.

$$\frac{1}{2}\left\| f\left(\mathbf{x}\right)+\mathbf{J}\left(\mathbf{x}\right)^{T}\Delta\mathbf{x}\right\|^{2}=\frac{1}{2}\left(f\left(\mathbf{x}\right)+\mathbf{J}\left(\mathbf{x}\right)^{T}\Delta\mathbf{x}\right)^{T}\left(f\left(\mathbf{x}\right)+\mathbf{J}\left(\mathbf{x}\right)^{T}\Delta\mathbf{x}\right)$$
$$=\frac{1}{2}\left(\|f(\mathbf{x})\|_{2}^{2}+2f\left(\mathbf{x}\right)\mathbf{J}(\mathbf{x})^{T}\Delta\mathbf{x}+\Delta\mathbf{x}^{T}\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^{T}\Delta\mathbf{x}\right).$$

$\Delta\mathbf{x}$ 의 derivate 가 0이 되도록 설정하면 다음과 같다.

$$\mathbf{J}(\mathbf{x})f\left(\mathbf{x}\right)+\mathbf{J}(\mathbf{x})\mathbf{J}^{T}\left(\mathbf{x}\right)\Delta\mathbf{x}=\mathbf{0}.$$

그러면 다음 식이 얻어진다.

$$\underbrace{\mathbf{J}(\mathbf{x})\mathbf{J}^{T}}_{\mathbf{H}(\mathbf{x})}\left(\mathbf{x}\right)\Delta\mathbf{x}=\underbrace{-\mathbf{J}(\mathbf{x})f\left(\mathbf{x}\right)}_{\mathbf{g}(\mathbf{x})}. \tag{95}$$

이 식은 $\Delta\mathbf{x}$ 에 대한 linear equation이다. (*normal equation* or *Gauss-Newton equation*이라 불린다.) 좌변의 계수행렬을 $\mathbf{H}$, 우변의 항을 $\mathbf{g}$ 라 정의하면다음 식이 얻어진다.

$$\mathbf{H}\Delta\mathbf{x}=\mathbf{g}. \tag{96}$$

TODO

1. Set it initial value as $\mathbf{x}_0$.
2. For k-th iteration, calculate the Jacobian $\mathbf{J}(\mathbf{x}_k)$ and residual $f(\mathbf{x}_k)$.
3. Solve the normal equation: $\mathbf{H}\Delta\mathbf{x}_k=\mathbf{g}$.
4. If $\Delta\mathbf{x}_k$ is small enough, stop the algorithm. Otherwise let $\mathbf{x}_{k+1}=\mathbf{x}_k+\Delta\mathbf{x}_k$ and return to step 2.

TODO

4.5 The Levernberg-Marquatdt Method

Gauss-Newton 에서 사용하는 approximate second-order Taylor expansion 는 근사 지점 주변에서만 적절한 근사가 된다. 따라서 적절한 근사가 되는 $\Delta\mathbf{x}$에 *trust-region* 을 설정하는 method가 개발되었는데, 이러한 형태의 알고리즘을 *trust-region method* 라 부른다. 일반적으로 trust-region의 범위를 결정하는 방법은 approximate model과 실제 함수와의 차이를 사용하는 것이다. 차이가 크면 trust-region을 줄이고, 차이가 작다면 확장한다. 정량적으로 근사정도를 설명하기위해 indicator ρ 를 도입해보자.

$$\rho=\frac{f\left(\mathbf{x}+\Delta\mathbf{x}\right)-f\left(\mathbf{x}\right)}{\mathbf{J}\left(\mathbf{x}\right)^{T}\Delta\mathbf{x}}. \tag{97}$$

ρ의 분자는 objective function의 의 감소값이고, 분모는 approximate model의 감소값이다. ρ가 1에 가까우면 적절하게 근사되었음을 의미한다. 만약 ρ 가 작다면 trust-region 을 줄여야 하고, ρ가 상대적으로 크면 trust-region을 확장해도 괜찮다.

따라서 Gauss-Newton 방법보다 더 나은 효과를 낼 수 있는 개선된 버전의 비선형 최적화 프레임워크를 구축합니다.

ρ 을 최적화 프레임워크에 도입해보자.

1. Give the initial value $\mathbf{x}_0$ and initial trust-region radius μ.
2. For k-th iteration, we solve a linear problem based on Gauss-Newton's method added with a trust-region:
$$\min_{\Delta \mathbf{x}_k} \frac{1}{2} \left\| f(\mathbf{x}_k) + \mathbf{J}(\mathbf{x}_k)^T \Delta \mathbf{x}_k \right\|^2, \quad \text{s.t.} \quad \|\mathbf{D}\Delta \mathbf{x}_k\|^2 \leqslant \mu, \tag{98}$$
where μ is the radius and $\mathbf{D}$ is a coefficient matrix, which will be discussed in below.
3. Compute ρ
4. If $\rho > \frac{3}{4}$, set $\mu = 2\mu$.
5. Otherwise, if $\rho < \frac{1}{4}$, set $\mu = 0.5\mu$.
6. If ρ is larger than a given threshold, set $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$.
7. Go back to step 2 if not converged, otherwise return the result.

TODO

5 Visual Odometry: Part I

Visual Odometry는 feature을 추출할 경우 feature-base method, 추출하지 않을 경우 Direct-method 로 구분된다. 이 책에서는 feature -base method를 다루려 한다.

5.1 Feature Method

VO에는 다양한 feature들이 사용되는데 SIFT, SURF, ORB들이 대표적이다. 이러한 feature들은 hand-crated feature로 구분되는데 다음과 같은 특징을 갖고있다.

1. Repeatability: The same feature can be found in different images.
2. Distinctiveness: Different features have different expressions.
3. Efficiency: In the same image, the number of feature points should be far smaller than the number of pixels.

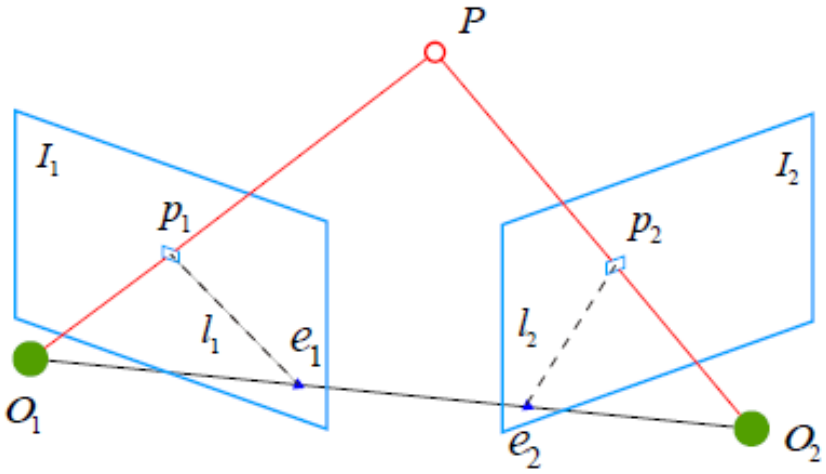

Figure 5: The epipolar constraints

4. Locality: The feature is only related to a small image area.

교재에서는 ORB feature 설계과정과 코드에 대한 설명이 있지만, 생략하도록 하겠다.

5.2 2D2D: Epipolar Geometry

5.2.1 Epipolar Constraints

Fig 5 와 같이, stereo camera 상황에서 한 쌍의 feature point 를 매칭한 상황을 가정해보자. 이러한 매칭이 충분할 경우 두 camera 사이의 motion을 복원 할 수 있다.

Fig. 5를 예로 들어 생각해보자. 우리의 목표는 두 프레임 I_1, I_2 사이의 relative motion을 구하는 것이다. I_1 에서 I_2 로의 motion을 $\mathbf{R}, \mathbf{t}$, 두 카메라의 center를 O_1, O_2 라 하자. 그리고 feature matching으로 얻은 pair p_1 과 p_2 가 있다고 하자.

이러한 상황에서 feature pair 또는 두 frame간의 관계는 epipolar geometry로 기술 될 수 있다. Epipolar geometry에서 사용하는 notation들을 먼저 알아보자.

1. epipolar plane : 세 점 O_1, O_2, P 가 결정하는 평면 ; Plane O_1O_2P
2. epipole : line O_1O_2 와 image plane I_1, I_2의 교차점 ; e_1, e_2
3. baseline : camera center을 잇는 line ; O_1O_2
4. epipolar line : epipolar plane과 image plane I_1, I_2 의 intersection ; l_1, l_2

첫 번째 이미지만 본다면 ray $\overrightarrow{O_1p_1}$ 는 Point P 가 위치 가능한 곳을 나타낸다. (점 P 가 ray 어느 곳에 위치해 있든, image plane 위의 점 p_1 으로 projection 되기 때문이다.).

두 번째 이미지의 epipolar line $\overrightarrow{e_2p_2}$ 는 point P 가 projection 될 수 있는 위치를 의미한다. (첫 번째 이미지만 본다면 , p_1 으로 Projection 될 수 있는 point P 는 ray $\overrightarrow{O_1p_1}$ 위에 존재해야 하고, ray $\overrightarrow{O_1p_1}$ 을 두 번째 이미지에 projection 하면 epipolar line $\overrightarrow{e_2p_2}$ 이기 때문이다)

이제 $\mathbf{p}_1, \mathbf{p}_2$ 간의 변환을 수식적으로 살펴보자.

첫 번째 프레임에서 P의 Position 이 다음과 같이 표현하자.

$$\mathbf{P} = [X, Y, Z]^T.$$

Pinhole 모델에 따르면 $\mathbf{p}_1, \mathbf{p}_2$ 두 픽셀의 픽셀 위치는 다음과 같다.

$$s_1\mathbf{p}_1 = \mathbf{KP}, \quad s_2\mathbf{p}_2 = \mathbf{K}\left(\mathbf{RP} + \mathbf{t}\right), \tag{99}$$

두 프레임이 동일한 카메라를 사용하였다고 가정하면, camera intrinsic matrix $\mathbf{K}$ 는 동일하다.그리고 $\mathbf{R}, \mathbf{t}$ 는 첫 번째 프레임에서 두 번째 프레임으로 좌표변환하는 rotation , translation matrix이다. 정확한 notion 은 $\mathbf{R}_{21}$, $\mathbf{t}_{21}$ 이며, 아랫첨자는 첫 번째 프레임에서 두 번째 프레임으로의 변환임을 의미한다.

homogeneous coordinate에서 동일함을 다음과 같이 표현하자. (up to a scale)

$$s\mathbf{p} \simeq \mathbf{p}. \tag{100}$$

그러면 Pinhole Projection 식은 다음과 같아진다.

$$\mathbf{p}_1 \simeq \mathbf{KP}, \quad \mathbf{p}_2 \simeq \mathbf{K}\left(\mathbf{RP} + \mathbf{t}\right). \tag{101}$$

pixel coordinate $\mathbf{p}_1, \mathbf{p}_2$ 가 아니라, normalized plane 위의 점 $\mathbf{x}_1, \mathbf{x}_2$ 으로도 표현 할수 있다.

$$\mathbf{x}_1 = \mathbf{K}^{-1}\mathbf{p}_1, \quad \mathbf{x}_2 = \mathbf{K}^{-1}\mathbf{p}_2. \tag{102}$$

이 때 두 point 간의 관계는 다음과 같다.

$$\mathbf{x}_2 \simeq \mathbf{R}\mathbf{x}_1 + \mathbf{t}. \tag{103}$$

양 변을 $\mathbf{t}$ cross pruduct 하자. matrix form으로는 $\mathbf{t}^\wedge$ 을 multiply 하는 것과 동일하다. 우변의 $\mathbf{t}$ 는 외적과정에서 사라진다.

$$\mathbf{t}^\wedge\mathbf{x}_2 \simeq \mathbf{t}^\wedge\mathbf{R}\mathbf{x}_1. \tag{104}$$

양 변에 $\mathbf{x}_2^T$ 을 multiply 하자.

$$\mathbf{x}_2^T\mathbf{t}^\wedge\mathbf{x}_2 \simeq \mathbf{x}_2^T\mathbf{t}^\wedge\mathbf{R}\mathbf{x}_1. \tag{105}$$

좌변의 $\mathbf{t}^\wedge\mathbf{x}_2$ 는 $\mathbf{t}$ 와 $\mathbf{x}_2$ 에 수직인 벡터이다. 따라서 $\mathbf{x}_2^T\mathbf{t}^\wedge\mathbf{x}_2 = 0$ 이다. 우변만 살리면 다음 식이 성립한다.

$$\mathbf{x}_2^T\mathbf{t}^\wedge\mathbf{R}\mathbf{x}_1 = 0. \tag{106}$$

다시 pixel coordinate $\mathbf{p}_1, \mathbf{p}_2$ 로 변환하면 다음과 같다.

$$\mathbf{p}_2^T\mathbf{K}^{-\mathrm{T}}\mathbf{t}^\wedge\mathbf{R}\mathbf{K}^{-1}\mathbf{p}_1 = 0. \tag{107}$$

두 방정식 모두 epipolar constraint 라 부르며, O_1, P, O_2 가 한 평면에 존재해야 함을 의미한다. quadratic form에서 중간에 위치한 term은 matching 결과와 무관하므로 따로 빼줄수 있다. 이 term을 pixel coordinate 에서는 Fundamental matrix $\mathbf{F}$, normalized plane 에서는 Essential Matrix $\mathbf{E}$ 라 부른다.

$$\mathbf{E} = \mathbf{t}^\wedge\mathbf{R}, \quad \mathbf{F} = \mathbf{K}^{-\mathrm{T}}\mathbf{E}\mathbf{K}^{-1}, \quad \mathbf{x}_2^T\mathbf{E}\mathbf{x}_1 = \mathbf{p}_2^T\mathbf{F}\mathbf{p}_1 = 0. \tag{108}$$

이제 두 frame 간의 relative motion을 추정하는 것은 다음 두 단계로 구성된다.

1. 매칭된 point를 사용하여 $\mathbf{E}$ or $\mathbf{F}$ 추정
2. $\mathbf{E}$ or $\mathbf{F}$ 을 사용해서 $\mathbf{R}, \mathbf{t}$ 추정

하나의 카메라만 사용하는 SLAM의 경우 $\mathbf{E}$ 와 $\mathbf{F}$ 는 큰 차이가 없기 때문에, 좀 더 간단한 형태인 essential matrix $\mathbf{E}$ 를 주로 사용한다. (* 서로 다른 카메라로 찍은 이미지를 주로 사용하는 SfM에서는 차이가 존재한다)

5.3 Essential Matrix

essential matrix $\mathbf{E} = \mathbf{t}^\wedge\mathbf{R}$ 는 3×3 행렬로 9개의 원소가 존재한다. 하지만 모든 3×3 matrix가 essential matrix가 될 수 있는 것은 아니고, 일종의 constraint가 존재한다.

- essential matrix는 epipolar constraint에서 얻어지는데, $\mathbf{x}_2^T\mathbf{E}\mathbf{x}_1 = 0$, 즉 *equal-to-zero* 방정식 이므로, $\mathbf{E}$에 0이 아닌 상수를 곱한 후에도 제약 조건은 만족된다. 이런 특성을 $\mathbf{E}$'s equivalence라 부른다.
- $\mathbf{E} = \mathbf{t}^\wedge\mathbf{R}$ 에 의하면, essential matrix $\mathbf{E}$ 의 singular value는 $[\sigma, \sigma, 0]^T$ 형태여야 한다. 이를 internal properties of essential matrix라 부른다.
- translation 과 rotation은 각각 3DOF 이다. 따라서 $\mathbf{t}^\wedge\mathbf{R}$ 는 6 DOF이다. 하지만 essential matrix의 equivalence 성질에 의해 실제로는 5 DOF 이다.

$\mathbf{E}$ 는 5 DOF 이기 때문에 5쌍의 점만 사용해서, $\mathbf{E}$ 를 구할 수도 있다. 하지만 internal property 가 비선형이기 때문에 선형대수적 지식만으로 추정하기는 어렵다. 따라서 equivalence만 고려하면 (9-1) DOF 가 되기 때문에 8쌍의 점을 사용하는 방법을 사용하기도 한다. 이를 *eight-point algorithm* 이라 부른다.

eight point algorihtm

normalized plane에서 한 쌍의 점을 생각해보자.

$$\mathbf{x}_1 = [u_1, v_1, 1]^T, \mathbf{x}_2 = [u_2, v_2, 1]^T \tag{109}$$

epipolar constraint 에 의해 다음 식이 얻어진다.

$$\Big(u_2, v_2, 1\Big) \begin{pmatrix} e_1 & e_2 & e_3 \\ e_4 & e_5 & e_6 \\ e_7 & e_8 & e_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = 0. \tag{110}$$

matrix $\mathbf{E}$ 를 벡터형태로 표현하면 다음과 같다. in the vector form:

$$\mathbf{e} = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9]^T,$$

$\mathbf{e}$ 를 사용해서 epipolar constraint를 표현하면 다음과 같아진다.

$$[u_2u_1, u_2v_1, u_2, v_2u_1, v_2v_1, v_2, u_1, v_1, 1] \cdot \mathbf{e} = 0. \tag{111}$$

8개의 점에 대해서 linear equation system을 만들면 다음과 같이 표현된다.

$$\begin{pmatrix} u_2^1u_1^1 & u_2^1v_1^1 & u_2^1 & v_2^1u_1^1 & v_2^1v_1^1 & v_2^1 & u_1^1 & v_1^1 & 1 \\ u_2^2u_1^2 & u_2^2v_1^2 & u_2^2 & v_2^2u_1^2 & v_2^2v_1^2 & v_2^2 & u_1^2 & v_1^2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ u_2^8u_1^8 & u_2^8v_1^8 & u_2^8 & v_2^8u_1^8 & v_2^8v_1^8 & v_2^8 & u_1^8 & v_1^8 & 1 \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \\ e_6 \\ e_7 \\ e_8 \\ e_9 \end{pmatrix} = 0. \tag{112}$$

좌변의 첫 번째 matrix는 8×9 matrix이며, element들은 매칭한 점들로 얻어지므로 주어진 행렬로이다. 이는 $\mathbf{Ax} = \mathbf{0}$ 에서 $\mathbf{x}$ 를 추정하는 것, 즉 $\mathbf{A}$ 의 nullspace을 구하는 문제로 회귀한다.

위의 linear equation system을 통해 essential matrix $\mathbf{E}$ 을 구했다면, 여기서 $\mathbf{R}, \mathbf{t}$ 를 복구해야 한다. 이과정에서 SVD를 사용한다. $\mathbf{E}$ 가 다음과 같이 분해해보자.

$$\mathbf{E} = \mathbf{U\Sigma V}^T, \tag{113}$$

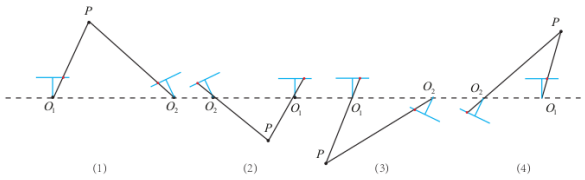

Figure 6: We get four solutions when decomposing the essential matrix.

여기서 $\mathbf{U}, \mathbf{V}$는 직교 행렬이고 $\mathbf{\Sigma}$는 특이값 행렬이다. 이 때 $\mathbf{E}$의 internal property 에 의해 $\mathbf{\Sigma} = \text{diag}(\sigma, \sigma, 0)$을 만족해야 한다. 이 경우 두 가지 후보가 존재한다.

$$\begin{aligned} \mathbf{t}_1^\wedge &= \mathbf{U}\mathbf{R}_Z(\tfrac{\pi}{2})\mathbf{\Sigma}\mathbf{U}^T, \quad \mathbf{R}_1 = \mathbf{U}\mathbf{R}_Z^T(\tfrac{\pi}{2})\mathbf{V}^T \\ \mathbf{t}_2^\wedge &= \mathbf{U}\mathbf{R}_Z(-\tfrac{\pi}{2})\mathbf{\Sigma}\mathbf{U}^T, \quad \mathbf{R}_2 = \mathbf{U}\mathbf{R}_Z^T(-\tfrac{\pi}{2})\mathbf{V}^T. \end{aligned} \tag{114}$$

$\mathbf{t}_1^\wedge, \mathbf{R}_1$ 에 대해서만 essential Matrix를 확인해보자.

$$\mathbf{t}_1^\wedge \mathbf{R}_1 = \mathbf{U}\mathbf{R}_Z(\frac{\pi}{2})\mathbf{\Sigma}\mathbf{U}^T\mathbf{U}\mathbf{R}_Z^T(\frac{\pi}{2})\mathbf{V}^T \tag{115}$$

$$= \mathbf{U}\mathbf{R}_Z(\frac{\pi}{2})\mathbf{\Sigma}\mathbf{R}_Z^T(\frac{\pi}{2})\mathbf{V}^T \tag{116}$$

$$= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{117}$$

여기서 $\mathbf{R}_Z\left(\frac{\pi}{2}\right)$ 는 Z 축을 따라 90°를 회전시키는 회전 행렬을 의미한다. epipolar constraint 에서 $-\mathbf{E}$는 $\mathbf{E}$와 같기 때문에 $\mathbf{t}$에 $(-)$ 를 붙여도 같은 결과가 나온다. 따라서 $\mathbf{E}$에서 $\mathbf{t}, \mathbf{R}$로 분해하는 방법에는 4가지 솔루션이 존재한다. Fig. 6은 essential matrix을 분해해서 가능한 4개의 솔루션을 보여준다. 우리는 얻어진 feature point가 양쪽 카메라에서 모두 positive depth을 얻어야 함을 알고 있다. 따라서 Fig. 6 에서 가능한 솔루션은 (1) 이다.

DLT로 얻은 Essential Matrix가 internal property 을 만족하지 못할 수도 있다. 예를 들어 singular matrix가 $\mathbf{\Sigma} = \text{diag}(\sigma_1, \sigma_2, \sigma_3)$ 및 $\sigma_1 \geqslant \sigma_2 \geqslant \sigma_3$ 가 될 수도 있다. 이 경우 singular value를 보정하여 새로운 essential matrix를 구한다.

$$\mathbf{E} = \mathbf{U}\text{diag}(\frac{\sigma_1 + \sigma_2}{2}, \frac{\sigma_1 + \sigma_2}{2}, 0)\mathbf{V}^T. \tag{118}$$

(* 위 솔루션이 기존 $\mathbf{E}$ 를 가장 가까운 essential matrix space에 projection 하는 것과 동일하다는 것이 밝혀져 있다.)

이외에도 matching이 올바르게 되었다면, σ_1 과 σ_2 는 유사한 값이 나오고, σ_3 는 0과 비슷한 값이 나올것이다. 이 경우 Singular matrix를 $\text{diag}(1, 1, 0)$ 으로 설정해도 무방하다.

(* equivalence 에 의해 $\text{diag}(1, 1, 0)$ 와 $\text{diag}(\sigma, \sigma, 0)$ 은 essential matrix에서 동등하다)

5.4 Homography

two-view geomtry에서는 앞에서 다뤘던 fundamental matrix 와 essential matrix 외에도 homography matrix $\mathbf{H}$ 를 자주 사용합니다. scene의 feature point가 모두 동일한 평면에 존재할 경우 homography matrix만 사용해서 motion을 추정할 수 있습니다. feature point을 모두 같은 평면에 존재하는 것으로 근사할 수 있는 드론 환경에서 유용하게 사용가능합니다.

앞에서 말했듯이 homography matrix는 feature point가 하나의 plane에만 존재하는 경우의 transformation 만 기술합니다. 에를 들어 이미지 I_1 및 I_2에서 point P가 projection 된 p_1 및 p_2 가 매칭되었다고 가정하자. 이 때 point는 어떤 평면 위에 존재해야 하므로 다음 식을 만족한다.

$$\mathbf{n}^T\mathbf{P} + d = 0. \tag{119}$$

d 를 우변으로 보내고, 양변을 $-d$ 로 나누면 다음과 같다.

$$-\frac{\mathbf{n}^T\mathbf{P}}{d} = 1. \tag{120}$$

point P 을 image I_2 에 projection 하는 eq을 생각해보면 다음이 유도된다.

$$\begin{aligned}
\mathbf{p}_2 &\simeq \mathbf{K}(\mathbf{R}\mathbf{P} + \mathbf{t}) \\
&\simeq \mathbf{K}\left(\mathbf{R}\mathbf{P} + \mathbf{t}\cdot(-\frac{\mathbf{n}^T\mathbf{P}}{d})\right) \\
&\simeq \mathbf{K}\left(\mathbf{R} - \frac{\mathbf{t}\mathbf{n}^T}{d}\right)\mathbf{P} \\
&\simeq \underbrace{\mathbf{K}\left(\mathbf{R} - \frac{\mathbf{t}\mathbf{n}^T}{d}\right)\mathbf{K}^{-1}}_{\mathbf{H}}\mathbf{p}_1.
\end{aligned}$$

따라서 $\mathbf{p}_1$ 에서 $\mathbf{p}_2$ 로 변환하는 과정은 $\mathbf{H}$ 를 사용하면 다음과 같아진다.

$$\mathbf{p}_2 \simeq \mathbf{H}\mathbf{p}_1. \tag{121}$$

유도과정에서의 $\mathbf{H}$ 는 Rotation 과 translation , 그리고 plane parameter $\mathbf{n}, d$ 로 구성되어 있다. matrix $\mathbf{H}$ 에서 motion 을 유도하는 방법은 Fundamental matrix와 유사하게, $\mathbf{H}$ 를 계산한 다음 factorize하여 translation 과 rotation을 찾으면 된다.

$$\begin{pmatrix} u_2 \\ v_2 \\ 1 \end{pmatrix} \simeq \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{pmatrix} \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix}. \tag{122}$$

homogeneous coordinate에서 진행되는 연산이기 때문에 $\mathbf{H}$ 에 상수를 곱해줘도 무방하다. 위 matrix eq을

연립방정식 형태로 만들면 다음과 같다.

$$u_2 = \frac{h_1 u_1 + h_2 v_1 + h_3}{h_7 u_1 + h_8 v_1 + h_9}$$
$$v_2 = \frac{h_4 u_1 + h_5 v_1 + h_6}{h_7 u_1 + h_8 v_1 + h_9}.$$

우변의 분모를 곱해주면 다음과 같다. 다만 homography 는 homogeneous coordinate의 특성 때문에 $\mathbf{h}_9 = 1$ 로 설정해도 무방하다.

$$h_1 u_1 + h_2 v_1 + h_3 - h_7 u_1 u_2 - h_8 v_1 u_2 = u_2$$
$$h_4 u_1 + h_5 v_1 + h_6 - h_7 u_1 v_2 - h_8 v_1 v_2 = v_2.$$

homography matrix는 8DOF 이다. 그리고 한 쌍의 점에서 두 개의 방정식이 얻어지므로, 이론적으로는 4개의 pair만 존재해도 homography matrix을 구할 수 있다. 4개의 pair에서 얻어진 방정식을 쌓으면 다음과 같은 형태이다.

$$\begin{pmatrix} u_1^1 & v_1^1 & 1 & 0 & 0 & 0 & -u_1^1 u_2^1 & -v_1^1 u_2^1 \\ 0 & 0 & 0 & u_1^1 & v_1^1 & 1 & -u_1^1 v_2^1 & -v_1^1 v_2^1 \\ u_1^2 & v_1^2 & 1 & 0 & 0 & 0 & -u_1^2 u_2^2 & -v_1^2 u_2^2 \\ 0 & 0 & 0 & u_1^2 & v_1^2 & 1 & -u_1^2 v_2^2 & -v_1^2 v_2^2 \\ u_1^3 & v_1^3 & 1 & 0 & 0 & 0 & -u_1^3 u_2^3 & -v_1^3 u_2^3 \\ 0 & 0 & 0 & u_1^3 & v_1^3 & 1 & -u_1^3 v_2^3 & -v_1^3 v_2^3 \\ u_1^4 & v_1^4 & 1 & 0 & 0 & 0 & -u_1^4 u_2^4 & -v_1^4 u_2^4 \\ 0 & 0 & 0 & u_1^4 & v_1^4 & 1 & -u_1^4 v_2^4 & -v_1^4 v_2^4 \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \end{pmatrix} = \begin{pmatrix} u_2^1 \\ v_2^1 \\ u_2^2 \\ v_2^2 \\ u_2^3 \\ v_2^3 \\ u_2^4 \\ v_2^4 \end{pmatrix}. \tag{123}$$

위 matrix form eq 처럼 matrix $\mathbf{H}$ 를 벡터로 취급하여 문제를 푸는 것을 Direct Linear Transform (DLT) 라 부른다.

만약 feature point들이 동일 평면에 존재하거나, motion을 구할려는 두 프레임이 rotation만 했다면, essential matrix에서 노이즈의 영향이 커지게 된다. 이런 경우에는 fundamental matrix $\mathbf{F}$와 homography matrix $\mathbf{H}$ 을 동시에 추정한 다음 reprojection error가 작은 것을 선택하여 relative motion을 얻는다.

5.5 Triangulation

이전 챕터에서는 epipolar constraint를 사용하여 Camera motion을 추정하는 방법을 다뤘다. 이 부분은 Slam에서 localization 에 해당하는 부분이다. 이번 세션에서는 카메라 모션을 활용해서 feature을 mapping 하는 방법에 대해 다루려 한다.

Triganulation 은 두 카메라 사이의 relative motion 과 2d point mathing이 주어졌을 때, 3D point의 위치를 추정하는 방법이다. Fig. 7 와 같이 이미지 I_1 , I_2 , 2d point matching p_1, p_2 , 이미지 I_1 에서 I_2 로의 변환 $\mathbf{T}$가 주여졌을 때, 3D point P 의 위치를 추정한다고 해보자. 이론적으로는 $O_1 p_1$, $O_2 p_2$ 가

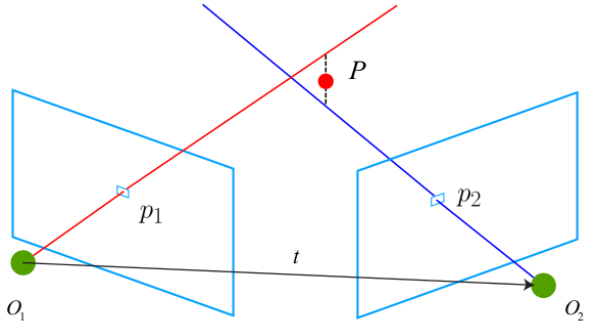

Figure 7: Use triangulation to estimate the depth.

한 점에서 교차해야 하고, 그 점이 P 가 되어야 한다. 하지만 노이즈로 인해 두 직선이 정확하게 교차하는 경우는 거의 없다. 따라서 least-square problem으로 가장 적절한 위치를 추정한다.

p_1, p_2 을 normalized coordinate 로 변환한 좌표를 $\mathbf{x}_1, \mathbf{x}_2$ 이라 하자. 이 때, 다음 식이 성립한다.

$$s_2\mathbf{x}_2 = s_1\mathbf{R}\mathbf{x}_1 + \mathbf{t}. \tag{124}$$

양변에 $\mathbf{x}_2^\wedge$ 을 multiply하면 다음과 같다.

$$s_2\mathbf{x}_2^\wedge\mathbf{x}_2 = 0 = s_1\mathbf{x}_2^\wedge\mathbf{R}\mathbf{x}_1 + \mathbf{x}_2^\wedge\mathbf{t}. \tag{125}$$

좌변은 0 이고 , 우변에서 $\mathbf{x}_1, \mathbf{x}_2, \mathbf{R}, \mathbf{t}$ 는 모두 알고 있는 값이다. 따라서 s_1 을 구할 수 있으며, s_1 은 O_1 기준에서 3d point P 의 depth이므로, 우리는 3d point P 의 position을 구하였다.

5.6 3D2D : PnP

PnP(Perspective-n-Point)는 3D point 와 2d point 간의 매칭이 존재할 때, 카메라의 pose를 추정하는 방법이다. 앞에서 설명한 2D-2D epipolar geometry의 경우 8-point algorithm과 같이 8개 이상의 matching pair가 필요하며, initialization, pure rotation, scale등의 문제가 있다. 하지만 3D-2D matching의 경우 3쌍의 매칭만 있어도 카메라 포즈를 복원할 수 있다. streo camera나 RGB-D cam을 사용할 경우 PnP을 통해 VO을 구현할 수 있으며, Monocular camera의 경우 PnP을 사용하기 전에 initialize가 필요하다.

PnP는 여러 종류의 구현이 존재한다. 예를 들어 P3P, DLT, EPnP(efficient PnP), UPnP 등이 존재한다. PnP에서도 least-square problem을 non linear optimization으로 푸는 방법을 사용한다. 이 과정을 bundle adjustment라 부른다. 이 책에서는 DLT 먼저 살펴본 뒤, bundle adjustment 방법을 소개하고자 한다.

5.6.1 Direct Linear Transformation

3D point와 2D point의 매칭이 주어졌을 때, 카메라 포즈를 구하는 문제를 생각해보자.

3D point P 는 homogeneous form으로 나타내면 다음과 같다.

$$\mathbf{P} = (X, Y, Z, 1)^T$$

3D point P 를 이미지 I_1 의 normalized plane에 projection 한 점을 $\mathbf{x}_1$ 이라 하자.

$$\mathbf{x}_1 = (u_1, v_1, 1)^T$$

Pinhole 카메라의 Projection 을 생각해보면, 다음 식이 성립한다.

$$s \begin{pmatrix} u_1 \\ v_1 \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} t_1 & t_2 & t_3 & t_4 \\ t_5 & t_6 & t_7 & t_8 \\ t_9 & t_{10} & t_{11} & t_{12} \end{pmatrix}}_{[\mathbf{R}|\mathbf{t}]} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}. \tag{126}$$

scale factor s 를 제거하면 다음 두 조건이 얻어진다.

$$u_1 = \frac{t_1 X + t_2 Y + t_3 Z + t_4}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}, \quad v_1 = \frac{t_5 X + t_6 Y + t_7 Z + t_8}{t_9 X + t_{10} Y + t_{11} Z + t_{12}}.$$

식을 간단히하기 위해 $\mathbf{T}$ 의 각 row vector를 다음과 같이 정의하자.

$$\mathbf{t}_1 = (t_1, t_2, t_3, t_4)^T, \mathbf{t}_2 = (t_5, t_6, t_7, t_8)^T, \mathbf{t}_3 = (t_9, t_{10}, t_{11}, t_{12})^T.$$

그러면 두 조건은 다음과 같이 바뀐다.

$$\mathbf{t}_1^T \mathbf{P} - \mathbf{t}_3^T \mathbf{P} u_1 = 0,$$

$$\mathbf{t}_2^T \mathbf{P} - \mathbf{t}_3^T \mathbf{P} v_1 = 0.$$

우리가 풀어야 할 문제는 3D-2D matching이 존재할 때 "카메라의 포즈"를 복원하는 것이다. 따라서 위 조건에서 $\mathbf{P}, u_1, v_1$ 은 주어진 값이며, $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3$ 가 구해야 할 값이다. 만약 N 개가 매칭되어 있는 상황이라면, 위 조건은 다음과 같은 linear eq을 푸는것과 동일해진다.

$$\begin{pmatrix} \mathbf{P}_1^T & 0 & -u_1 \mathbf{P}_1^T \\ 0 & \mathbf{P}_1^T & -v_1 \mathbf{P}_1^T \\ \vdots & \vdots & \vdots \\ \mathbf{P}_N^T & 0 & -u_N \mathbf{P}_N^T \\ 0 & \mathbf{P}_N^T & -v_N \mathbf{P}_N^T \end{pmatrix} \begin{pmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \mathbf{t}_3 \end{pmatrix} = 0. \tag{127}$$

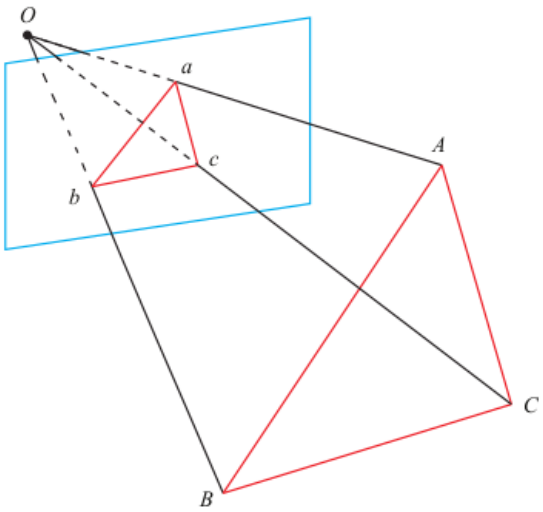

Figure 8: P3P probelm

t의 총 12차원이므로 **T** 는 적어도 6개의 매칭만 존재한다면 구할 수 있다. 위와 같이Lienar eq방식으로 푸는 방법을 Direct Linear Transform(DLT)라 부른다. 만약 6쌍보다 많이 매칭되었다면, SVD를 사용해서 least-sqaure solution을 찾으면 된다.

Essential Matrix에서 카메라 포즈를 복원할 때 생겼던 문제와 마찬가지로, matrix **T** 는 SE(3) 제약조건을 충족하지 않을 수도 있다. 따라서 우리는 matrix **T** 를 SE(3) 로 변환하는 과정을 필요로 한다. (* 정확히는 회전 행렬을 SO(3)을 만족하게 변환하면서, translation vector 정보도 유지해야 한다.) 이 때 QR decomposition을 사용하거나, 아래와 같은 변환을 사용한다.

$$\mathbf{R} \leftarrow \left(\mathbf{R}\mathbf{R}^T\right)^{-\frac{1}{2}}\mathbf{R}. \tag{128}$$

5.6.2 P3P

이론적으로는 3쌍의 매칭만 존재해도 camera pose를 복원할 수 있다. 이 때 사용하는 방법이 바로 P3P 이다. Fig. 8 와 같이, 3D point A, B, C 와 2D point a, b, c 가 매칭되었다고 해보자. 다음 삼각형 쌍들은 각각 같은 평면에 위치해 있다.

$$\Delta Oab - \Delta OAB, \quad \Delta Obc - \Delta OBC, \quad \Delta Oac - \Delta OAC. \tag{129}$$

각 평면에 있는 삼각형에 대해 제 2코사인 법칙을 적용하면 다음 식들이 성립한다.

$$\begin{aligned}
\overline{OA}^2 + \overline{OB}^2 - 2.\overline{OA}.\overline{OB}.\cos\langle a, b\rangle &= \overline{AB}^2 \\
\overline{OB}^2 + \overline{OC}^2 - 2.\overline{OB}.\overline{OC}.\cos\langle b, c\rangle &= \overline{BC}^2 \\
\overline{OA}^2 + \overline{OC}^2 - 2.\overline{OA}.\overline{OC}.\cos\langle a, c\rangle &= \overline{AC}^2.
\end{aligned} \tag{130}$$

각 방정식의 양변을 $\overline{OC}^2$ 로 나누고, $x = \overline{OA}/\overline{OC},\ y = \overline{OB}/\overline{OC}$ 으로 치환하면 다음과 같다.

$$\begin{aligned} x^2 + y^2 - 2.x.y.\cos\langle a,b\rangle &= \overline{AB}^2/\overline{OC}^2 \\ y^2 + 1^2 - 2.y.\cos\langle b,c\rangle &= \overline{BC}^2/\overline{OC}^2 \\ x^2 + 1^2 - 2.x.\cos\langle a,c\rangle &= \overline{AC}^2/\overline{OC}^2. \end{aligned} \tag{131}$$

$v = \overline{AB}^2/\overline{OC}^2,\ u.v = \overline{BC}^2/\overline{OC}^2,\ w.v = \overline{AC}^2/\overline{OC}^2,$

$$\begin{aligned} x^2 + y^2 - 2.x.y.\cos\langle a,b\rangle - v &= 0 \\ y^2 + 1^2 - 2.y.\cos\langle b,c\rangle - u.v &= 0 \\ x^2 + 1^2 - 2.x.\cos\langle a,c\rangle - w.v &= 0. \end{aligned} \tag{132}$$

첫 번째 방정식의 v 를 없애는 당향으로 식을 전개하면, 다음 두 방정식이 얻어진다.

$$\begin{aligned} (1-u)\,y^2 - ux^2 - 2\cos\langle b,c\rangle\, y + 2uxy\cos\langle a,b\rangle + 1 &= 0 \\ (1-w)\,x^2 - wy^2 - 2\cos\langle a,c\rangle\, x + 2wxy\cos\langle a,b\rangle + 1 &= 0. \end{aligned} \tag{133}$$

위의 연립 방정식에서 주어진 값과, 우리가 구해야 하는 값을 구분해보자. 먼저 이미지에서 2D point 위치를 알고 있으므로, $\cos\langle a,b\rangle$, $\cos\langle b,c\rangle$, $\cos\langle a,c\rangle$ 는 주어진 값이다. 3D point의 world coordinate 좌표는 알고 있으므로 $u = \overline{BC}^2/\overline{AB}^2$, $w = \overline{AC}^2/\overline{AB}^2$ 또한 주어진 값이다. x, y 는 아직 Camera pose를 확정하지 못했으므로 구해야 하는 값이다.

따라서 위 두 개의 방정식은 x, y 두 개의 미지수로 구성된 연립방정식이며, 이 연립방정식을 풀기위해서는 별도의 방법이 필요하다. 위 방정식을 풀 경우, 최대 4개의 Pose candidate을 얻을 수 있었다.

P3P는 세개의 점만 활용해서 Pose를 추정할수 있다는 장점이 있지만, 다음과 같은 단점도 존재한다.

1. 3 점의 정보만 활용하기 때문에, 보다 많은 매칭정보가 주어질 경우 유용한 정보를 사용하지 못 할 수도 있다.

2. point 위치가 noisy하거나 mismatch 될 경우 알고리즘이 유효하지 않다..

SLAM에서는 PnP만 활용하기 보다는, PnP로 카메라 포즈를 initilize하고, least-squre problem으로 추정된 값을 bundle adjustment한다.

5.6.3 Solve PnP by Minimizing the Reprojection Error

PnP 문제를 linear algebra 외에도 reprojection error를 사용한 non linear least-square problem으로 풀어보자.

n개의 3D point P 와 projection point p 가 주어졌을 때, 카메라 포즈 $\mathbf{R}, \mathbf{t}$ 를 추정하는 문제를 생각해보자. 3d point 좌표가 $\mathbf{P}_i = [X_i, Y_i, Z_i]^T$ 이고, projection point가 $\mathbf{u}_i = [u_i, v_i]^T$ 일 때, 다음 식이

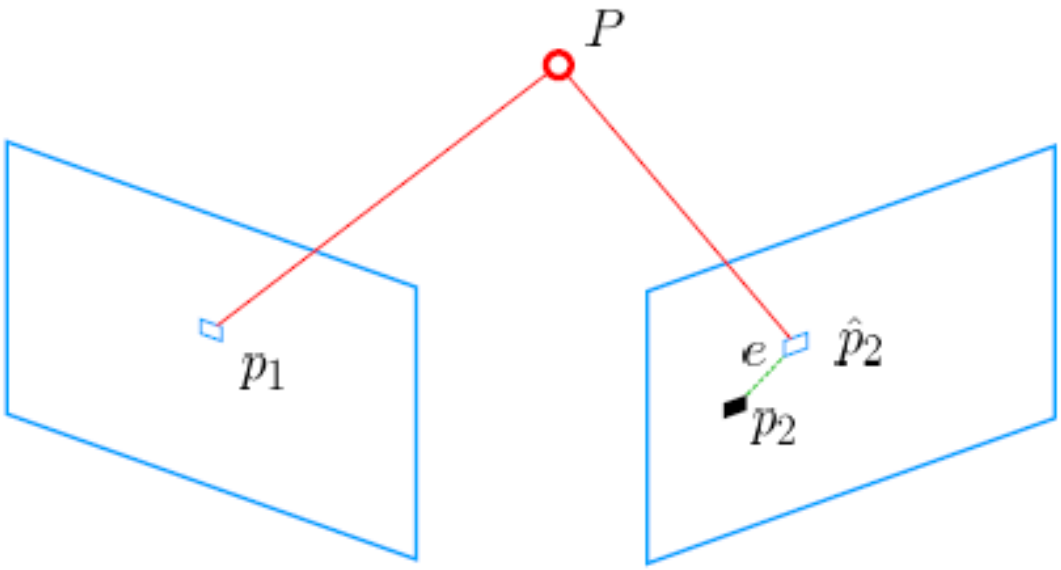

Figure 9: The reprojection error.

성립한다.

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{KT} \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix}. \tag{134}$$

matrix form은 다음과 같다.

$$s_i \mathbf{u}_i = \mathbf{KTP}_i.$$

P 를 projection했을 때의 pixel 좌표와 point p 사이의 에러를 최소화 하는 Transform matrix $\mathbf{T}$ 를 구하는 것이 목표이다.

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \frac{1}{2} \sum_{i=1}^{n} \left\| \mathbf{u}_i - \frac{1}{s_i} \mathbf{KTP}_i \right\|_2^2. \tag{135}$$

Gauss-Newton method나 Levenberg-Marquardt method를 적용하기 위해, derivative를 계산하자. 에러가 아래와 같이 근사해보자.

$$\mathbf{e}(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}(\mathbf{x}) + \mathbf{J}^T \Delta\mathbf{x}. \tag{136}$$

에러는 pixel coordinate error이므로 2d이다. $\mathbf{x}$ 는 camera pose이므로 6d이다. 따라서 $\mathbf{J}^T$ 는 2×6 행렬이다.

우리과 관심있는 카메라 좌표계(pose 를 구해야할 좌표계) 에서, 3d point P의 좌표를 다음과 같이 표현하자.

$$\mathbf{P}' = (\mathbf{TP})_{1:3} = [X', Y', Z']^T. \tag{137}$$

camera projection model을 적용하면 다음과 같다.

$$s\mathbf{u} = \mathbf{K}\mathbf{P}'. \tag{138}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix}. \tag{139}$$

depth 정보 s 를 제거하면 u, v 에 관한 두 식이 얻어진다.

$$u = f_x \frac{X'}{Z'} + c_x, \quad v = f_y \frac{Y'}{Z'} + c_y. \tag{140}$$

에러는 위 식으로 얻어진 u, v 와 matching 된 2d point 좌표간의 차이로 정의된다. 우리가 사용할 error approximate $\mathbf{e}(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}(\mathbf{x}) + \mathbf{J}^T\Delta\mathbf{x}$. 에서 자코비안은 pose SE(3)에 대한 미분값이다. 따라서 Lie group을 사용해서 자코비안을 구해보자.

$$\frac{\partial \mathbf{e}}{\partial \delta\boldsymbol{\xi}} = \lim_{\delta\boldsymbol{\xi}\to 0} \frac{\mathbf{e}\left(\delta\boldsymbol{\xi} \oplus \boldsymbol{\xi}\right) - \mathbf{e}(\boldsymbol{\xi})}{\delta\boldsymbol{\xi}} = \frac{\partial \mathbf{e}}{\partial \mathbf{P}'} \frac{\partial \mathbf{P}'}{\partial \delta\boldsymbol{\xi}}. \tag{141}$$

(* $\partial P'$ 을 chain rule에 넣은 이유에 대해서는 error term에 대해 생각해보자. $\sum_{i=1}^{n} \left\| \mathbf{u}_i - \frac{1}{s_i}\mathbf{K}\mathbf{T}\mathbf{P}_i \right\|_2^2$ 에서 $\mathbf{P}'$ 을 적용하면 $\mathbf{u}_i - \frac{1}{s_i}\mathbf{K}\mathbf{P}'_i$ 이다.)

chain rule의 첫 번째 term은 다음과 같다.

$$\frac{\partial \mathbf{e}}{\partial \mathbf{P}'} = -\begin{bmatrix} \frac{\partial u}{\partial X'} & \frac{\partial u}{\partial Y'} & \frac{\partial u}{\partial Z'} \\ \frac{\partial v}{\partial X'} & \frac{\partial v}{\partial Y'} & \frac{\partial v}{\partial Z'} \end{bmatrix} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix}. \tag{142}$$

$\mathbf{P}' = (\mathbf{T}\mathbf{P})_{1:3}$ 이므로, chain rule의 두 번째 term은 다음과 같다.

$$\frac{\partial (\mathbf{T}\mathbf{P})}{\partial \delta\boldsymbol{\xi}} = (\mathbf{T}\mathbf{P})^{\odot} = \begin{bmatrix} \mathbf{I} & -\mathbf{P}'^{\wedge} \\ \mathbf{0}^T & \mathbf{0}^T \end{bmatrix}. \tag{143}$$

$$\frac{\partial \mathbf{P}'}{\partial \delta\boldsymbol{\xi}} = \left[\mathbf{I}, -\mathbf{P}'^{\wedge}\right]. \tag{144}$$

따라서 자코비안은 다음과 같다.

$$\frac{\partial \mathbf{e}}{\partial \delta\boldsymbol{\xi}} = -\begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix}. \tag{145}$$

특징점의 3d poision도 최적화 해야 한다. camera pose (SE(3))와 달리 3차원 좌표이기 때문에 자코비

안을 구하는 것은 간단하다.

$$\frac{\partial \mathbf{e}}{\partial \mathbf{P}} = \frac{\partial \mathbf{e}}{\partial \mathbf{P}'} \frac{\partial \mathbf{P}'}{\partial \mathbf{P}}. \tag{146}$$

$$\mathbf{P}' = (\mathbf{TP})_{1:3} = \mathbf{RP} + \mathbf{t},$$

$$\frac{\partial \mathbf{e}}{\partial \mathbf{P}} = - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix} \mathbf{R}. \tag{147}$$

5.7 3D3D : Iterative Closest Point (ICP)

ICP 는 3D-point 간의 매칭이 존재할 때, pose estimation을 하는 알고리즘이다. 예를 들어 두 좌표계에서 다음과 같이 3d point matching이 발생하였다고 가정해보자.

$$\mathbf{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_n\}, \quad \mathbf{P}' = \{\mathbf{p}'_1, \cdots, \mathbf{p}'_n\},$$

우리의 목표는 두 좌표계 간의 transformation $\mathbf{R}, \mathbf{t}$ 을 구하는 것이다. 물론 $\mathbf{R}, \mathbf{t}$ 는 다음 식을 만족해야 한다.

$$\forall i, \mathbf{p}_i = \mathbf{R}\mathbf{p}'_i + \mathbf{t}.$$

ICP의 솔루션은 선형대수적으로(SVD) 푸는 방법과 non linear optimization(like bundle adjustment)로 푸는 방법으로 나눌수 있다. 먼저 SVD을 살펴본뒤, 다음으로 비선형 최적화 방법을 살펴보자.

5.8 Using Linear Algebra (SVD)

SVD를 사용해서 ICP을 풀어보자. 3d space 에서 error term음 다음과 같이 정의하자.

$$\mathbf{e}_i = \mathbf{p}_i - (\mathbf{R}\mathbf{p}'_i + \mathbf{t}). \tag{148}$$

least-square problem 형태로 표현하면 다음과 같다.

$$\min_{\mathbf{R},\mathbf{t}} \frac{1}{2} \sum_{i=1}^{n} \| (\mathbf{p}_i - (\mathbf{R}\mathbf{p}_i{}' + \mathbf{t})) \|_2^2. \tag{149}$$

문제해결에 앞서, 두 point set의 centroid를 다음과 같이 정의하자.

$$\mathbf{p} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{p}_i), \quad \mathbf{p}' = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{p}'_i). \tag{150}$$

centroid를 사용하면 Objective function은 아래와 같이 표현된다.

$$\begin{aligned}\frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{p}_i-(\mathbf{R}\mathbf{p}_i{}'+\mathbf{t})\right\|^2 &= \frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{p}_i-\mathbf{R}\mathbf{p}_i{}'-\mathbf{t}-\mathbf{p}+\mathbf{R}\mathbf{p}'+\mathbf{p}-\mathbf{R}\mathbf{p}'\right\|^2\\ &= \frac{1}{2}\sum_{i=1}^{n}\left\|(\mathbf{p}_i-\mathbf{p}-\mathbf{R}\left(\mathbf{p}_i{}'-\mathbf{p}'\right))+\left(\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right)\right\|^2\\ &= \frac{1}{2}\sum_{i=1}^{n}(\left\|\mathbf{p}_i-\mathbf{p}-\mathbf{R}\left(\mathbf{p}_i{}'-\mathbf{p}'\right)\right\|^2+\left\|\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right\|^2+\\ &\quad 2(\mathbf{p}_i-\mathbf{p}-\mathbf{R}\left(\mathbf{p}_i{}'-\mathbf{p}'\right))^T\left(\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right)).\end{aligned}$$

세 번째 항의 $(\mathbf{p}_i-\mathbf{p}-\mathbf{R}\left(\mathbf{p}_i{}'-\mathbf{p}'\right))$ 는 summation 과정에서 0이 되기 때문에, 최종 Objective function 은 다음과 같다.

$$\min_{\mathbf{R},\mathbf{t}} J=\frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{p}_i-\mathbf{p}-\mathbf{R}\left(\mathbf{p}_i{}'-\mathbf{p}'\right)\right\|^2+\left\|\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right\|^2. \tag{151}$$

첫 번째 항은 rotation matrix $\mathbf{R}$ 에만 depend한 반면, 두 번째 항은 $\mathbf{R}$, $\mathbf{t}$ 모두에 대해 depend 하며, centroid에만 관련이 있다. 즉 첫 번째 항을 최적화 해서 $\mathbf{R}$ 을 얻으면 두 번째 항을 0을 만드는 $\mathbf{t}$ 를 항상 구할 수 있다. (* $\frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right\|^2=\frac{n}{2}\left\|\mathbf{p}-\mathbf{R}\mathbf{p}'-\mathbf{t}\right\|^2$ 이므로,,, 하지만 첫 번째 항은 subscript i 에 의해 값이 계속 바뀐다.) 따라서 ICP는 다음 세 단계로 구성된다.

1. Calculate the centroids of the two groups of points $\mathbf{p},\mathbf{p}'$, and then calculate the **de-centroid coordinates** of each point:

$$\mathbf{q}_i=\mathbf{p}_i-\mathbf{p},\quad \mathbf{q}_i'=\mathbf{p}_i'-\mathbf{p}'.$$

2. The rotation matrix is calculated according to the following optimization problem:

$$\mathbf{R}^*=\arg\min_{\mathbf{R}}\frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{q}_i-\mathbf{R}\mathbf{q}_i'\right\|^2. \tag{152}$$

3. Calculate $\mathbf{t}$ according to $\mathbf{R}$ in step 2:

$$\mathbf{t}^*=\mathbf{p}-\mathbf{R}\mathbf{p}'. \tag{153}$$

$\mathbf{R}^*$ 을 구하는 두 번째 step을 자세히 살펴보자. $\mathbf{R}$에 대한 error term을 확장하면 아래와 같다.

$$\frac{1}{2}\sum_{i=1}^{n}\left\|\mathbf{q}_i-\mathbf{R}\mathbf{q}_i'\right\|^2=\frac{1}{2}\sum_{i=1}^{n}\underbrace{\mathbf{q}_i^T\mathbf{q}_i}_{\text{not relavant}}+\mathbf{q}_i'^{\mathrm{T}}\underbrace{\mathbf{R}^T\mathbf{R}}_{=\mathbf{I}}\mathbf{q}_i'-2\mathbf{q}_i^T\mathbf{R}\mathbf{q}_i'. \tag{154}$$

첫 번째 항은 $\mathbf{R}$ 과 관련이 없다. $\mathbf{R}^T\mathbf{R}=\mathbf{I}$이므로 두 번째 항도 $\mathbf{R}$ 과 관련이 없다 . 따라서 실제 Objective function는 다음과 같은 형태가 된다.

$$\sum_{i=1}^{n}-\mathbf{q}_i^T\mathbf{R}\mathbf{q}_i'=\sum_{i=1}^{n}-\mathrm{tr}\left(\mathbf{R}\mathbf{q}_i'\mathbf{q}_i^T\right)=-\mathrm{tr}\left(\mathbf{R}\sum_{i=1}^{n}\mathbf{q}_i'\mathbf{q}_i^T\right). \tag{155}$$

전개를 위해 $\mathbf{W}$ 을 아래와 같이 정의해보자.

$$\mathbf{W} = \sum_{i=1}^{n} \mathbf{q}_i \mathbf{q}_i'^{\mathrm{T}}. \tag{156}$$

$\mathbf{W}$ 는 3×3 행렬이다. $\mathbf{W}$ 에 SVD을 적용하면 다음과 같은 형태이다.

$$\mathbf{W} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{157}$$

$\mathbf{W}$ 가 full rank 일 때, $\mathbf{R}$ 은 아래와 같다.

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T. \tag{158}$$

.

5.9 Using non-linear optimization

ICP를 해결하는 또 다른 방법은 비선형 최적화를 사용하여 최적 값을 반복적으로 찾는 것이다. Lie algebra 를 사용해서 pose를 표현하면 Objective function은 아래와 같다.

$$\min_{\boldsymbol{\xi}} = \frac{1}{2} \sum_{i=1}^{n} \| (\mathbf{p}_i - \exp(\boldsymbol{\xi}^\wedge) \mathbf{p}_i') \|_2^2. \tag{159}$$

포즈에 대한 single error term의 derivative는 아래와 같다.

$$\frac{\partial \mathbf{e}}{\partial \delta \boldsymbol{\xi}} = -(\exp(\boldsymbol{\xi}^\wedge) \mathbf{p}_i')^\odot. \tag{160}$$

이 후에는 Gauss-Newton이나, Levenberg 방법을 도입하면 된다.

6 Visual Odometry: Part II

앞서 다루웠던 Feature method와 더불어 중요한 direct method를 이번 챕터에서 다룬다.

6.1 The Motivation of the Direct Method

이전 챕터에서는 feature 를 사용해서 camera motion을 추정했다. feature method로도 visual odometry를 구현할 수 있지만 몇 가지 단점들이 존재한다.

1. feature를 뽑는 과정 자체가 overhead
   (a) SIFT 는 CPU에서 real time으로 사용 불가

(b) ORB는 20ms 소요

(c) SLAM이 30 HZ를 요하는 것을 생각해보면 너무 과함

2. feature method에서는 모든 정보를 사용하지 않고, 일부 feature point 정보만 사용함. 즉 유용한 이미지 정보를 사용하지 못 할 수도 있음.

3. texture info가 없는 환경에서는 feature를 뽑기 힘듬. 이 경우 camera motion을 추정할 feature가 부족할 수도 있음 ($\leftrightarrow$ LoFTR)

이러한 단점을 극복하기 위해 몇가지 아이디어들이 제시되었다.

- feature point을 뽑돼, descriptor를 계산하지 않고, point의 움직임만 추적하는 방법. 이 경우 descriptor를 계산하는 시간을 아낄 수 있고, optical flow 를 계산하는 시간은 descriptor calcuation + matching 보다는 명백히 짧다
- 위와 유사하게, key point 만 뽑고 point 만 추적하는 방식

첫 번째 아이디어는 feature point 을 사용하기는 하지만, matching 대신 optical flow tracking 을 사용한다. tracking 정보를 기반으로 epipolar geometry, PnP 또는 ICP를 사용해서 camera motion을 추정한다. tracking을 위해서는 keypoint을 뽑아야 하므로, image corner point에 의존적이다. (* 즉 texture info가 없는 환경에서는 여전히 문제가 될 수 있다) 두 번째 아이디어 Direct method의 경우 pixel gray scale을 사용해서 카메라 모션과 point projection을 추정한다. 이 경우 corner point을 사용하지 않는다.

feature method에서는 reprojection error를 최소화 하는 방법을 사용해서 카메라 모션을 추정한다. direct method에서는 photometric error 을 최소화 하는 방법을 사용한다.

이번 챕터에서는 direct method을 소개 할 예정이다. direct method는 픽셀의 밝기 정보를 기반으로 카메라 움직임을 추정하기 때문에 feature method의 overhead을 겪지 않는다. (feature 계산시간, feature가 부족한 현상). texture가 없는 환경이더라도, 이미지의 밝기 변화가 존재하는 이상, direct method는 작동한다. direct mtehod는 사용하는 픽셀 수에 따라 sparse, semi-dense, dens로 분류된다. feature method는 항상 sparse map만 형성할수 있었던 반면에, direct method는 semi-dense 또는 dense map도 형성할 수 있다.

6.2 2D Optical Flow

direct-method 는 optical flow와 유사하다. 다만 optical flow는 pixel의 움직임을 추적한다면, direct-mthod는 카메라의 motion도 추적한다는 것이 차이점이다. 이 책에서는 이러한 유사성 때문에 optical flow를 먼저 설명한다.

Fig 10 은 하나의 pixel이 시간이 지남에 따라 움직이는 모습을 보여주고 있다. 예시에서는 흰 배경에서 pixel이 움직이고 있는 것을 묘사하고 있지만, 실제로는 real-world에서 같은 부분을 tracking 할 것이다. Optical Flow는 sparse optical flow 와 Dense optical Flow로 나뉘는데, 전자는 이미지 전체가 아닌 일부에

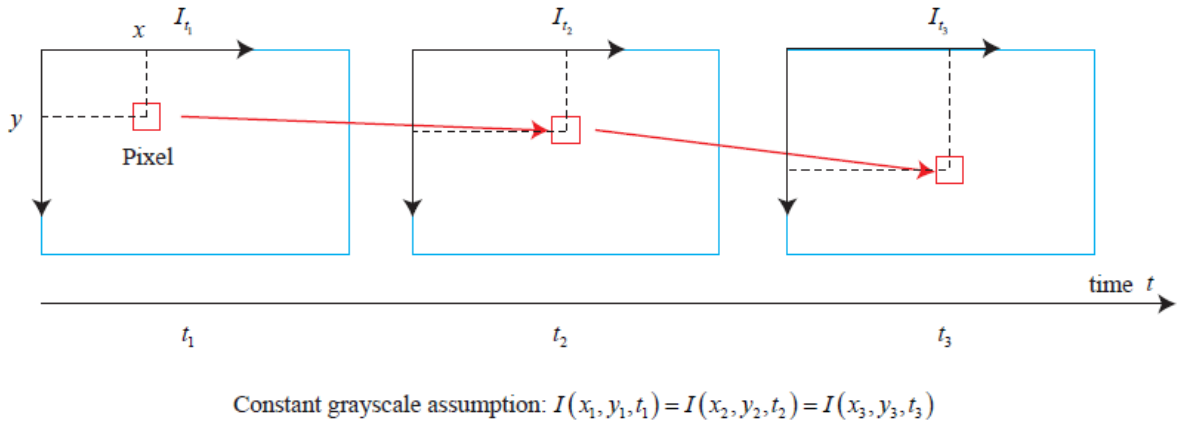

Figure 10: Optical flow of a single pixel.

대해서만 추적하는 경우를 의미하고, 후자는 이미지 전체에 적용하는 경우를 의미한다. Sparse의 경우는 Lucas-Kanade optical flow가 가장 유명하며, Dense의 경우 Horn-Schunck optical flow가 대중적이다. 이 책에서는 Lucas-kanade Optical Flow만 소개하고자 한다.

6.2.1 Lucas-Kanade Optical Flow

LK optical flow는 기본적으로 시간에 따라 이미지가 변화는 환경을 가정한다. 아래와 같은 Image function을 생각해보자. 그리고 간단하게 문제를 풀기 위해 grayscale , 즉 scalar function이라 하자.

$$\mathbf{I}(x, y, t).$$

우리의 목적은 시간 t 에서 (x, y) 위치에 있던 픽셀이 , 다른 시간에 어떤 위치에 있는지 추정하는 것이다. LK optical flow는 Constant grayscale assumption 을 사용한다. 같은 pixel은 시간이 변함에 따라 픽셀 위치는 변화하더라도, grayscale을 동일하다는 가정이다. 수식적으로는 아래와 같다.

$$\mathbf{I}(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t) = \mathbf{I}(x, y, t). \tag{161}$$

이 가정의 좌변에 Taylor expansion을 적용하면 아래와 같다.

$$\mathbf{I}\left(x + \mathrm{d}x, y + \mathrm{d}y, t + \mathrm{d}t\right) \approx \mathbf{I}\left(x, y, t\right) + \frac{\partial \mathbf{I}}{\partial x}\mathrm{d}x + \frac{\partial \mathbf{I}}{\partial y}\mathrm{d}y + \frac{\partial \mathbf{I}}{\partial t}\mathrm{d}t. \tag{162}$$

가정에 의해, 위 eq의 좌변과 우변의 첫 번째 항이 소거된다. 따라서,

$$\frac{\partial \mathbf{I}}{\partial x}\mathrm{d}x + \frac{\partial \mathbf{I}}{\partial y}\mathrm{d}y + \frac{\partial \mathbf{I}}{\partial t}\mathrm{d}t = 0. \tag{163}$$

양변을 $\mathrm{d}t$로 나누면,

$$\frac{\partial \mathbf{I}}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial \mathbf{I}}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} = -\frac{\partial \mathbf{I}}{\partial t}, \tag{164}$$

$\mathrm{d}x/\mathrm{d}t$, $\mathrm{d}y/\mathrm{d}t$ 를 u, v 로 표현하고, $\partial\mathbf{I}/\partial x$, $\partial\mathbf{I}/\partial y$,$\partial\mathbf{I}/\partial t$ 를 $\mathbf{I}_x, \mathbf{I}_y$, $\mathbf{I}_t$ 로 표현하면

$$\begin{bmatrix} \mathbf{I}_x & \mathbf{I}_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -\mathbf{I}_t. \tag{165}$$

우리의 목표는 u, v 를 구하는 것이다. 하지만 위 공식은 적어도 두개의 선형방정식이 있어야 풀 수있다. 따라서 Window의 모든 픽셀이 시간에 따라 동일하게 움직인다고 가정하고, u, v 을 구하는 방법을 사용한다. $w \times w$ 크기의 window 을 생각해보자. 그러면 w^2 개의 방정식을 얻게 된다.

$$\begin{bmatrix} \mathbf{I}_x & \mathbf{I}_y \end{bmatrix}_k \begin{bmatrix} u \\ v \end{bmatrix} = -\mathbf{I}_{tk}, \quad k = 1, \ldots, w^2. \tag{166}$$

방정식들을 쌓으면,

$$\mathbf{A} = \begin{bmatrix} [\mathbf{I}_x, \mathbf{I}_y]_1 \\ \vdots \\ [\mathbf{I}_x, \mathbf{I}_y]_k \end{bmatrix}, \mathbf{b} = \begin{bmatrix} \mathbf{I}_{t1} \\ \vdots \\ \mathbf{I}_{tk} \end{bmatrix}. \tag{167}$$

$$\mathbf{A} \begin{bmatrix} u \\ v \end{bmatrix} = -\mathbf{b}. \tag{168}$$

이 때, 최적의 u, v 는

$$\begin{bmatrix} u \\ v \end{bmatrix}^* = -\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{b}. \tag{169}$$

6.3 Direct Method

본격적으로 direct method에 대해 알아보자.

6.3.1 Derivation of the Direct Method

Fig. 11 와 같은 상황을 생각해보자. 3D point P 의 world 좌표가 $[X, Y, Z]$ 이고, 두 이미지에서 P 에 해당하는 픽셀좌표가 p_1, p_2 라 하자. 우리의 목표는 첫 번째 이미지에서 두 번째 이미지로의 relative motion 을 구하는 것이다.

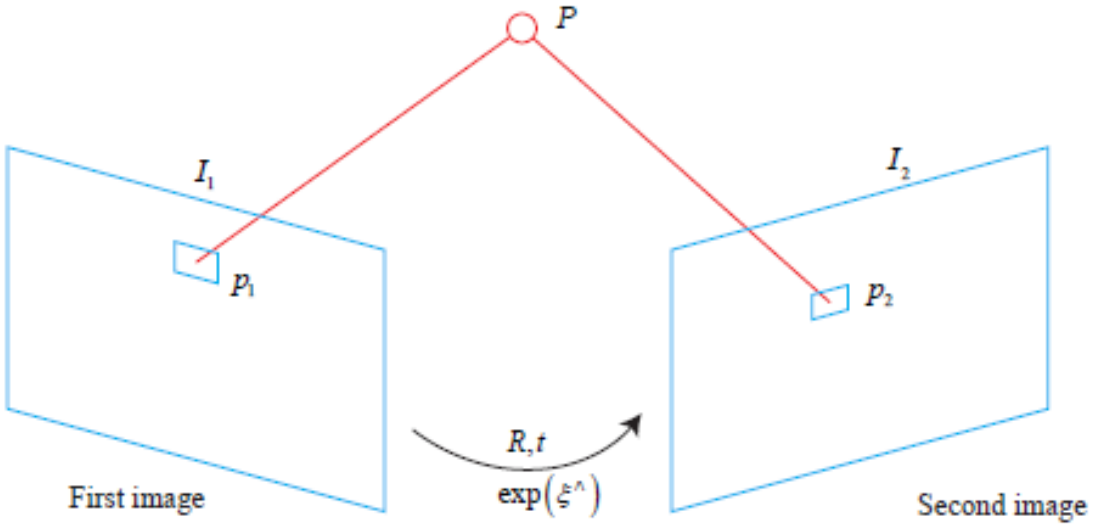

Figure 11: the direct method.

p_1, p_2 에 대한 projection 식은 다음과 같다.

$$\mathbf{p}_1 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_1 = \frac{1}{Z_1}\mathbf{KP},$$

$$\mathbf{p}_2 = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}_2 = \frac{1}{Z_2}\mathbf{K}\left(\mathbf{RP}+\mathbf{t}\right) = \frac{1}{Z_2}\mathbf{K}\left(\mathbf{TP}\right)_{1:3},$$

위 식에서 Z_1 , Z_2 는 각 카메라 좌표계에서 3d point P의 z 좌표, 즉 depth 를 의미한다.

- 이미지 1에서 P 좌표 : $\mathbf{P}_1 = [X_1, Y_1, Z_1]^T$
- 이미지 2에서 P 좌표 : $\mathbf{P}_2 = [X_2, Y_2, Z_2]^T$

feature method의 경우 descriptor를 사용해서 매칭을 진행하여 $\mathbf{p}_1, \mathbf{p}_2$ 의 픽셀 위치를 구하고 reprojection error을 계산하면 된다. 하지만 direct method의 경우 feature matching 을 하지 않기 때문에 어떤 $\mathbf{p}_2$와 $\mathbf{p}_1$이 같은 지점에 해당하는지 알 수 없다. 그렇기 때문에 direct method 에서는 우리가 구할려는 relative motion $\mathbf{T}$ 를 사용해 photometric error를 표현하고, error를 최소하는 $\mathbf{T}$ 를 구하는 것을 목표로 한다. 식을 통해 구체적으로 살펴 보자.

만약 3d point P 가 이미지 $\mathbf{I}_1, \mathbf{I}_2$ 어떤 점에 projection 되는지 안다면, photometric error는 다음과

같을 것이다.

$$e = \mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{p}_2). \tag{170}$$

하지만 direct method에서는 매칭을 하지 않기 때문에 $\mathbf{p}_2$ 을 다른 방식으로 표현할 필요가 있다.

$$\mathbf{q} = \mathbf{TP},$$
$$\mathbf{u} = \frac{1}{Z_2}\mathbf{Kq}.$$

$\mathbf{u}$ 를 사용해 $\mathbf{p}_2$ 를 치환하면 에러를 다음과 같이 정의된다.

$$e(\mathbf{T}) = \mathbf{I}_1(\mathbf{p}_1) - \mathbf{I}_2(\mathbf{u}), \tag{171}$$

이 때 최적화 문제는 다음과 같다.

$$\min_{\mathbf{T}} J(\mathbf{T}) = \|e\|^2. \tag{172}$$

SE(3) 인 $\mathbf{T}$ 가 최적화 변수이므로 Lie Algebra를 적용해야 한다. $\mathbf{T}$ 에 대한 Objective function 's derivative 는 다음과 같다.

$$\frac{\partial e}{\partial \mathbf{T}} = -\frac{\partial \mathbf{I}_2}{\partial \mathbf{u}}\frac{\partial \mathbf{u}}{\partial \mathbf{q}}\frac{\partial \mathbf{q}}{\partial \delta\boldsymbol{\xi}}, \tag{173}$$

Chain rule의 각 term은 다음 의미들을 갖고 있다.

1. $\partial \mathbf{I}_2/\partial \mathbf{u}$ is the grayscale gradient at pixel $\mathbf{u}$.
2. $\partial \mathbf{u}/\partial \mathbf{q}$ is the derivative of the projection equation with respect to the three-dimensional point in the camera frame. Let $\mathbf{q} = [X, Y, Z]^T$, , the derivative is:

$$\frac{\partial \mathbf{u}}{\partial \mathbf{q}} = \begin{bmatrix} \frac{\partial u}{\partial X} & \frac{\partial u}{\partial Y} & \frac{\partial u}{\partial Z} \\ \frac{\partial v}{\partial X} & \frac{\partial v}{\partial Y} & \frac{\partial v}{\partial Z} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} \end{bmatrix}. \tag{174}$$

3. $\partial \mathbf{q}/\partial \delta\boldsymbol{\xi}$ is the derivative of the transformed three-dimensional point with respect to the transformation

$$\frac{\partial \mathbf{q}}{\partial \delta\boldsymbol{\xi}} = [\mathbf{I}, -\mathbf{q}^\wedge]. \tag{175}$$

2, 3번째 term을 결합하면 다음과 같다.

$$\frac{\partial \mathbf{u}}{\partial \delta\boldsymbol{\xi}} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -\frac{f_x X}{Z^2} & -\frac{f_x XY}{Z^2} & f_x + \frac{f_x X^2}{Z^2} & -\frac{f_x Y}{Z} \\ 0 & \frac{f_y}{Z} & -\frac{f_y Y}{Z^2} & -f_y - \frac{f_y Y^2}{Z^2} & \frac{f_y XY}{Z^2} & \frac{f_y X}{Z} \end{bmatrix}. \tag{176}$$

따라서 objective funciton의 자코비안은 다음과 같다.

$$\mathbf{J} = -\frac{\partial \mathbf{I}_2}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \delta \boldsymbol{\xi}}. \tag{177}$$

자코비안을 구했으므로, Gauss-Newton 또는 Levenberg-Marquardt 을 사용해서 최적화를 진행하면 된다.

7 Filters and Optimization Approaches: Part I

GOAL

1. filter or least-square optimization 방법으로 slam backend을 formulation
2. bundle adjustment probelm에서 sparse structure을 사용하는 방법
3. g2o 와 ceres 라이브러리를 사용해서 Bundle Adjustment 문제 풀기

이전에는 Slam의 front end , 즉 Visual Odometry를 다뤘다. VO를 통해 두 frame 간의 relative transformation을 구할 수 있었다. 하지만 VO 만 사용할경우 오류가 누적되기 때문에, backend 에서 error drift 를 제거하는 과정이 필요하다. Backend에서는 Filter 와 optimization , 두 방법론이 존재하지만 최근에는 Optimization이 좀 더 많이 사용되는 추세이다. (* Andrew Davison 교수님의 "Visual Slam : Why Filter?"/ 여전히 IMU 센서와 퓨전하는 경우에는 EKF (filter approach)을 많이 사용한다.)

7.1 introduction

7.1.1 State Estimation from Probabilistic Perspective

Localization 과 Mapping을 동시에 하는 SLAM의 특성 상, Localization 과 Mapping의 성능은 서로의 영향을 받는다. 따라서 이전에 Mapping 한 landmark의 위치나 로봇의 trajectory가 이후의 정보에 의해 업데이트 될 수 도 있다. 이 때, 과거와 미래 정보를 함께 사용하여 업데이트 하는 방법을 batch 라 부른다. 반대로 현재 상태가 과거 정보에만 의존하여 결정되는 것을 incremental 이라 부른다.

$t = 0$ N time step만을 고려했을 때, $\mathbf{x}_0$ 의 $\mathbf{x}_N$까지의 로봇 position과 $\mathbf{y}_1, \cdots, \mathbf{y}_M$의 observation이 있다고 가정해보자. 이 때 motion 과 observation eq 은 다음과 같다.

$$\begin{cases} \mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k \\ \mathbf{z}_{k,j} = h\left(\mathbf{y}_j, \mathbf{x}_k\right) + \mathbf{v}_{k,j} \end{cases} \quad k = 1, \ldots, N,\ j = 1, \ldots, M. \tag{178}$$

이 eq을 풀기 위해서는 다음과 같은 사항을 고려해야 한다.

1. observation eq에서 $\mathbf{x}_k$가 $\mathbf{y}_j$ 를 볼 때에만 유효한 eq을 갖게 된다. 예를 들어 건물 전체에 걸쳐 Landmark가 형성되어 있다면, 카메라가 볼 수 있는 Landmark는 카메라 시야에 잡히는 극히 일부만

해당된다. 그럼에도 불구하고 motion eq 는 N 개에 불과한 반면, Landmark의 수는 훨씬 많기 때문에, 실제로 풀어야 할 방정식은 Observation eq 이 Motion eq보다 훨씬 많다.

2. motion 을 측정할 센서가 없을 경우, motion eq을 계산할 수 없다. 이 경우 다음과 같은 방법들을 고려한다.

   - 실제로 운동 방정식이 없다고 가정
   - 카메라가 움직이지 않는다고 가정
   - 카메라가 일정한 속도로 움직인다고 가정

   위 방법들 모두 사용 가능하며, 만약 모션 방정식이 없는 경우 최적화는 observation eq만 사용하여 진행된다. 이는 SfM과 매우 유사한 방법이다. 다만 SfM은 global feature와 같은 방법을 사용하는 반면, SLAM은 이미지의 order을 사용하는 것이 차이점이다.

위 motion and observation eq에서 고려했듯이, 우리는 센서을 통한 observation과 motion이 모두 noise의 영향을 받는 것으로 가정했다. 따라서 pose $\mathbf{x}$ 와 landmark $\mathbf{y}$ 모두 확률 변수로 고려할 것이다. 이제 우리가 풀어야 할 문제는 다음과 같다.

1. motion data $\mathbf{u}$ 와 observation data $\mathbf{z}$ 가 주어졌을 때, state $\mathbf{x}$ 와 landmark $\mathbf{y}$ 를 확률변수로 모델링 하는 방법
2. 새로운 데이터가 취득되었을 때, estimation을 업데이트 하는 방법

대부분의 통계적 모델링 처럼, state와 noise 모두 가우시안 분포를 따른다고 가정하자. 이 경우 평균과 공분산 값만 저장하면 된다. 평균은 state 의 optimal value를 나타내며, 공분산 행렬은은 uncertainty를 대표한다.

다시 원래 문제로 돌아와서, motion and observation 이 주어졌을 때, 가우시안 분포를 어떻게 추정하는지에 대해 다뤄보자.

Nonliear Optimization 세션에서 우리는 batch method를 MLE로 바꾸고 이를 least-square problem으로 풀 수 있음을 보였다.

카메라 포즈와 랜드마크의 위치 모두 최적화 변수이다. 표기의 편의성을 위해 이 섹션에서만 $\mathbf{x}_k$ 에 랜드마크 위치까지 포함시키자.

$$\mathbf{x}_k \triangleq \{\mathbf{x}_k, \mathbf{y}_1, \ldots, \mathbf{y}_m\}. \tag{179}$$

이 notation을 사용할경우 observation eq이 상대적으로 간단해진다.

$$\begin{cases} \mathbf{x}_k = f\left(\mathbf{x}_{k-1}, \mathbf{u}_k\right) + \mathbf{w}_k \\ \mathbf{z}_k = h\left(\mathbf{x}_k\right) + \mathbf{v}_k \end{cases} \qquad k = 1, \ldots, N. \tag{180}$$

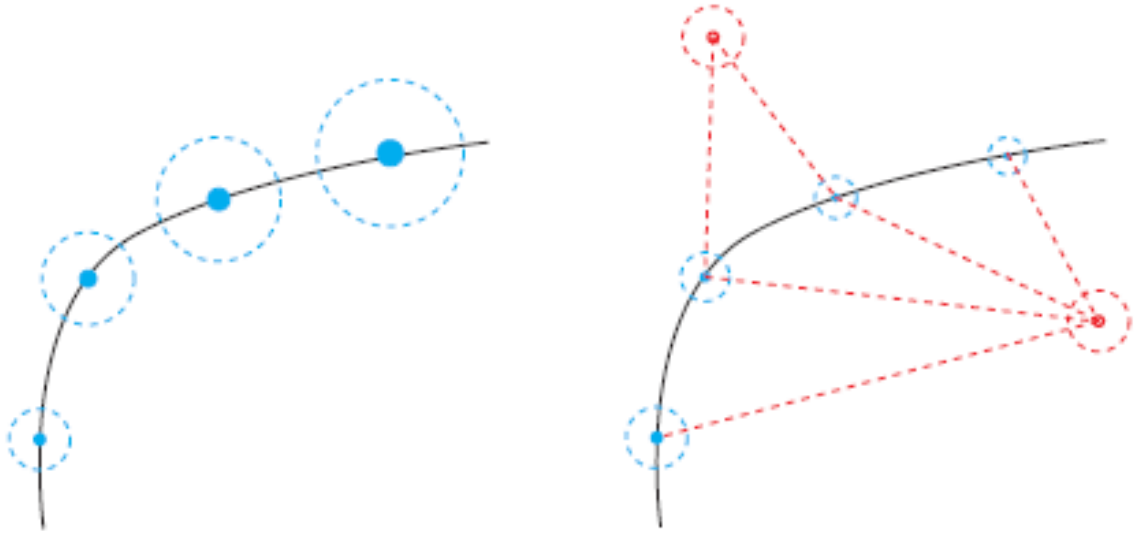

Figure 12: An intuitive description of uncertainty. Left : motion eq만 주어졌을 때, pose는 noise가 중첩되어 uncertainty가 점점 커진다. Right: road sign과 같은 landmark가 주어졌을 경우, 이를 활용하여 uncertainty를 줄일 수 있다. // 책에서는 left를 눈을 감고 걷는 것으로, right는 눈을 뜨고 외부 풍경을 보면서 걷는 것으로 비유하고 있다. 즉 observation을 통해 현재 위치를 보다 정확하게 추정할 수 있다는 의미

현재 state $\mathbf{x}_k$ 의 conditional distribution은 다음과 같이 표현된다.

$$P(\mathbf{x}_k|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}). \tag{181}$$

$\mathbf{x}_k$ 는 $\mathbf{x}_{k-1}$, $\mathbf{x}_{k-2}$ 와 연관이 있지만, $\mathbf{x}_0$ 만 conditin으로 넣어줘도 충분하다. 위 notation에서 $\mathbf{z}_k$ 가 그 시점의 모든 관측 데이터를 표현하고 있음을 주의해야 한다.

Bayes' rule을 적용하면 다음과 같다.

$$P\left(\mathbf{x}_k|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k}\right) \propto \underbrace{P\left(\mathbf{z}_k|\mathbf{x}_k\right)}_{\text{likelihood}} \underbrace{P\left(\mathbf{x}_k|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right)}_{\text{prior}}. \tag{182}$$

$\mathbf{x}_{k-1}$ 의 affect를 고려해보자. (* 다음 식이 이해가 되지 않는다면, 조건부 확률에 대해 다시 공부하는 것을 추천드립니다)

$$P\left(\mathbf{x}_k|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) = \int P\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) P\left(\mathbf{x}_{k-1}|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) \mathrm{d}\mathbf{x}_{k-1}. \tag{183}$$

7.1.2 Linear Systems and the Kalman Filter

Markov property를 가정해보자. 이 경우 k 번째 state는 $k-1$ state에만 depend 해지며, integral 내부 첫 번째 항이 간단해진다.

$$P\left(\mathbf{x}_k|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) = \int P\left(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) P\left(\mathbf{x}_{k-1}|\mathbf{x}_0, \mathbf{u}_{1:k}, \mathbf{z}_{1:k-1}\right) \mathrm{d}\mathbf{x}_{k-1}. \tag{184}$$

$$P\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)=P\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_k\right), \tag{185}$$

두 번째 term의 경우 state $\mathbf{x}_{k-1}$ 는 time k 에서의 control $\mathbf{u}_k$ 의 영향을 받지 않으므로, 다음 식이 성립한다.

$$P\left(\mathbf{x}_{k-1}|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)=P\left(\mathbf{x}_{k-1}|\mathbf{x}_0,\mathbf{u}_{1:k-1},\mathbf{z}_{1:k-1}\right), \tag{186}$$

Kalman filter는 motion and observation eq가 linear Gaussian System을 따른다고 가정한다. 이 때 motion and observation eq는 다음과 같다.

$$\begin{cases}\mathbf{x}_k=\mathbf{A}_k\mathbf{x}_{k-1}+\mathbf{u}_k+\mathbf{w}_k\\ \mathbf{z}_k=\mathbf{C}_k\mathbf{x}_k+\mathbf{v}_k\end{cases}\qquad k=1,\ldots,N, \tag{187}$$

$$\mathbf{w}_k\sim N(\mathbf{0},\mathbf{R}).\quad \mathbf{v}_k\sim N(\mathbf{0},\mathbf{Q}), \tag{188}$$

Markov property를 가정했을 때, 우리의 목표는 time $k-1$ 에서의 posterior state estimation $\hat{\mathbf{x}}_{k-1}$ 과 covariance $\hat{\mathbf{P}}_{k-1}$ 을 알고 있을 때, control과 observation 값을 사용해서 posterior state estimation $\hat{\mathbf{x}}_k$ 과 covariance $\hat{\mathbf{P}}_k$ 을 추정하는 것이다. 식 전개에서 posterior는 $\hat{\mathbf{x}}_k$, prior는 $\check{\mathbf{x}}_k$ 로 표현할 예정이다.

Prediction

이전 time step의 state을 알고 있을 때, motion eq을 사용해서 다음 state을 추정하는 과정이다. 수식적으로는 다음을 추정하는 단계이다.

$$P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)=\int P\left(\mathbf{x}_k|\mathbf{x}_{k-1},\mathbf{u}_k\right)P\left(\mathbf{x}_{k-1}|\mathbf{x}_0,\mathbf{u}_{1:k-1},\mathbf{z}_{1:k-1}\right)$$

다만 우리는 이전 time step의 posterior state ($\hat{\mathbf{x}}_{k-1}$ 과 covariance $\hat{\mathbf{P}}_{k-1}$)을 알고 있으므로 적분의 두 번째 항은 관심사가 아니다. 그리고 첫 번째 항은 motion eq에 의해 분포가 결정됨을 알고 있다. 따라서 prediction을 다음과 같다.

$$P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)=N\left(\mathbf{A}_k\hat{\mathbf{x}}_{k-1}+\mathbf{u}_k,\mathbf{A}_k\hat{\mathbf{P}}_{k-1}\mathbf{A}_k^T+\mathbf{R}\right). \tag{189}$$

observation 까지 고려할 때, 위 식은 prior 역할을 한다. 따라서 $\check{\mathbf{x}}_k,\check{\mathbf{P}}_k$ 는 다음과 같다.

$$P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k}\right)\propto\underbrace{P\left(\mathbf{z}_k|\mathbf{x}_k\right)}_{\text{likelihood}}\underbrace{P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)}_{\text{prior}}.$$

$$\check{\mathbf{x}}_k=\mathbf{A}_k\hat{\mathbf{x}}_{k-1}+\mathbf{u}_k,\quad \check{\mathbf{P}}_k=\mathbf{A}_k\hat{\mathbf{P}}_{k-1}\mathbf{A}_k^T+\mathbf{R}, \tag{190}$$

measurement

$$P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k}\right) \propto \underbrace{P\left(\mathbf{z}_k|\mathbf{x}_k\right)}_{\text{likelihood}} \underbrace{P\left(\mathbf{x}_k|\mathbf{x}_0,\mathbf{u}_{1:k},\mathbf{z}_{1:k-1}\right)}_{\text{prior}}.$$

위 식에서 likelihood 부분을 추정하는 파트이다. observation eq 에 의해 다음이 얻어진다.

$$P\left(\mathbf{z}_k|\mathbf{x}_k\right) = N\left(\mathbf{C}_k\mathbf{x}_k,\mathbf{Q}\right), \tag{191}$$

Update

time k 에서 Posterior를 $\mathbf{x}_k \sim N(\hat{\mathbf{x}}_k,\hat{\mathbf{P}}_k)$ 라 할 때, prediction + measurement 결과에 의해 다음이 성립한다.

$$N(\hat{\mathbf{x}}_k,\hat{\mathbf{P}}_k) = \eta N\left(\mathbf{C}_k\mathbf{x}_k,\mathbf{Q}\right)\cdot N(\check{\mathbf{x}}_k,\check{\mathbf{P}}_k), \tag{192}$$

여기서 η는 분포의 적분을 1로 만들기 위한 정규화 수이다.

양 변 모두 가우시안 분포이므로, 양 쪽의 exponential part만 비교하자. $\mathbf{x} \sim \mathcal{N}(\mu,\mathbf{\Sigma})$ 일 때, distribution 은 다음과 같다.

$$P\left(\mathbf{x}\right) = \frac{1}{\sqrt{(2\pi)^N \det(\mathbf{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}\left(\mathbf{x}-\mu\right)\right). \tag{193}$$

따라서 지수 부분에 대해 다음 방정식이 성립한다.

$$(\mathbf{x}_k-\hat{\mathbf{x}}_k)^T\hat{\mathbf{P}}_k^{-1}\left(\mathbf{x}_k-\hat{\mathbf{x}}_k\right) = \left(\mathbf{z}_k-\mathbf{C}_k\mathbf{x}_k\right)^T\mathbf{Q}^{-1}\left(\mathbf{z}_k-\mathbf{C}_k\mathbf{x}_k\right) + (\mathbf{x}_k-\check{\mathbf{x}}_k)^T\check{\mathbf{P}}_k^{-1}\left(\mathbf{x}_k-\check{\mathbf{x}}_k\right). \tag{194}$$

$\mathbf{x}_k$ 의 second order coefficient 을 비교해보자.

$$\hat{\mathbf{P}}_k^{-1} = \mathbf{C}_k^T\mathbf{Q}^{-1}\mathbf{C}_k + \check{\mathbf{P}}_k^{-1}, \tag{195}$$

전개의 편의를 위해 다음과 같이 $\mathbf{K}$ 를 정의하자.

$$\mathbf{K} = \hat{\mathbf{P}}_k\mathbf{C}_k^T\mathbf{Q}^{-1}. \tag{196}$$

$\mathbf{x}_k$ 의 second order coefficient 식 양변에 $\hat{\mathbf{P}}_k$ 을 곱하면 다음과 같다.

$$\mathbf{I} = \hat{\mathbf{P}}_k\mathbf{C}_k^T\mathbf{Q}^{-1}\mathbf{C}_k + \hat{\mathbf{P}}_k\check{\mathbf{P}}_k^{-1} = \mathbf{K}\mathbf{C}_k + \hat{\mathbf{P}}_k\check{\mathbf{P}}_k^{-1}. \tag{197}$$

따라서 posterior의 covariance $\hat{\mathbf{P}}_k$ 는 다음과 같다.

$$\hat{\mathbf{P}}_k = (\mathbf{I}-\mathbf{K}\mathbf{C}_k)\check{\mathbf{P}}_k. \tag{198}$$

$\mathbf{x}_k$ 의 first order coefficient 을 비교해보자.

$$-2\hat{\mathbf{x}}_k^T\hat{\mathbf{P}}_k^{-1}\mathbf{x}_k = -2\mathbf{z}_k^T\mathbf{Q}^{-1}\mathbf{C}_k\mathbf{x}_k - 2\check{\mathbf{x}}_k^T\check{\mathbf{P}}_k^{-1}\mathbf{x}_k. \tag{199}$$

정리하면 다음과 같다.(양변의 transpose 하고, Covariance Matrix 가 Symmetric임을 사용하자)

$$\hat{\mathbf{P}}_k^{-1}\hat{\mathbf{x}}_k = \mathbf{C}_k^T\mathbf{Q}^{-1}\mathbf{z}_k + \check{\mathbf{P}}_k^{-1}\check{\mathbf{x}}_k. \tag{200}$$

양변에 $\hat{\mathbf{P}}_k$ 을 곱하고, $\mathbf{K}$ 를 사용해서 표현하자.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{P}}_k\mathbf{C}_k^T\mathbf{Q}^{-1}\mathbf{z}_k + \hat{\mathbf{P}}_k\check{\mathbf{P}}_k^{-1}\check{\mathbf{x}}_k \tag{201}$$

$$= \mathbf{K}\mathbf{z}_k + (\mathbf{I} - \mathbf{K}\mathbf{C}_k)\,\check{\mathbf{x}}_k = \check{\mathbf{x}}_k + \mathbf{K}\left(\mathbf{z}_k - \mathbf{C}_k\check{\mathbf{x}}_k\right). \tag{202}$$

원래 목표였던, posterior $\hat{\mathbf{x}}_k, \hat{\mathbf{P}}_k$ 를 모두 구했다.

위 과정을 정리하면 다음과 같다.

1. *Predict*:

$$\check{\mathbf{x}}_k = \mathbf{A}_k\hat{\mathbf{x}}_{k-1} + \mathbf{u}_k, \quad \check{\mathbf{P}}_k = \mathbf{A}_k\hat{\mathbf{P}}_{k-1}\mathbf{A}_k^T + \mathbf{R}. \tag{203}$$

2. *Update*: Calculate $\mathbf{K}$, which is the Kalman gain:

$$\mathbf{K} = \check{\mathbf{P}}_k\mathbf{C}_k^T\left(\mathbf{C}_k\check{\mathbf{P}}_k\mathbf{C}_k^T + \mathbf{Q}_k\right)^{-1}, \tag{204}$$

and the posterior:

$$\begin{aligned}\hat{\mathbf{x}}_k &= \check{\mathbf{x}}_k + \mathbf{K}\left(\mathbf{z}_k - \mathbf{C}_k\check{\mathbf{x}}_k\right)\\ \hat{\mathbf{P}}_k &= \left(\mathbf{I} - \mathbf{K}\mathbf{C}_k\right)\check{\mathbf{P}}_k.\end{aligned}. \tag{205}$$

EKF todo

7.2 Bundle Adjustment and Graph Optimization

일반적으로 Bundle Adjustment는 카메라 파라미터 (intrinsic and extrinsic)과 3D randmark로 이미지를 최적화 하는 작업을 의미한다.

7.2.1 The Projection Model and Cost Function

시작에 앞서 Projection process에 대해 검토해보자. extrinsic, distortion, instrinsic을 차례대로 고려하면 다음 과정을 거친다.

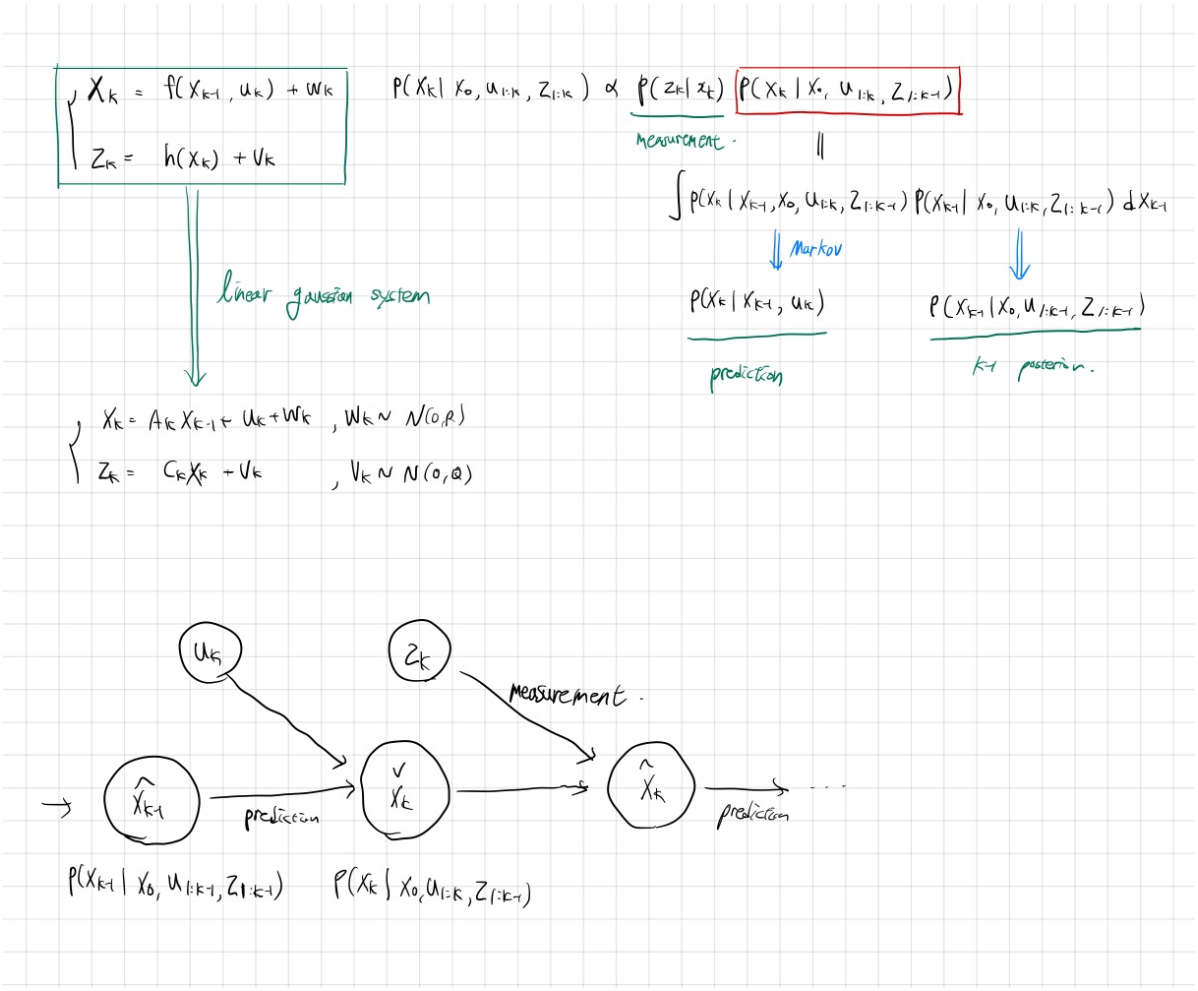

Figure 13: Kalman filter

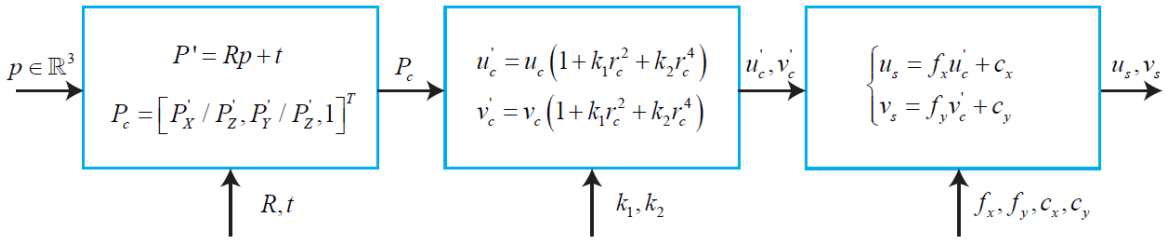

Figure 14: Projection Process

1. Extrinsic Parameter$(\mathbf{R}, \mathbf{t})$ 적용:

$$\mathbf{P}' = \mathbf{R}\mathbf{p} + \mathbf{t} = [X', Y', Z']^T. \tag{206}$$

2. normalized plane으로 projection:

$$\mathbf{P}_c = [u_c, v_c, 1]^T = [X'/Z', Y'/Z', 1]^T. \tag{207}$$

3. Distortion 모델 적용 (radical distortion 만 고려하자):

$$\begin{cases} u_c' = u_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \\ v_c' = v_c\left(1 + k_1 r_c^2 + k_2 r_c^4\right) \end{cases}. \tag{208}$$

4. intrinsic parameter 적용 $\mathbf{K}$:

$$\begin{cases} u_s = f_x u_c' + c_x \\ v_s = f_y v_c' + c_y \end{cases}. \tag{209}$$

Fig. 14 을 보면 이해가 더 쉽다.

위 과정을 잘 살펴보면 Observation eq와 동일함을 확인 할 수 있다. camera pose $\mathbf{x}$, landmark position $\mathbf{y}$ 을 사용하여 표현한 observation 은 다음과 같았다.

$$\mathbf{z} = h(\mathbf{x}, \mathbf{y}). \tag{210}$$

$\mathbf{T}$ 는 Camera pose를 의미하고 $\mathbf{p}$ 는 word coordinate에서 landmark 위치를 의미하므로 $h(\mathbf{T}, \mathbf{p})$ 는 observation eq와 동일하다.

다만 stochastic model임을 고려하면, 실제 픽셀 좌표가 $\mathbf{z} \triangleq [u_s, v_s]^T$ 일 때 error term은 다음과 같다.

$$\mathbf{e} = \mathbf{z} - h(\mathbf{T}, \mathbf{p}). \tag{211}$$

Observation eq 전체에 대해 error를 고려하면 overall cost function 은 다음과 같다. (z_{ij} 는 pose $\mathbf{T_i}$에서 landmark $\mathbf{p}_j$ 을 관측할 때의 data이다)

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\mathbf{e}_{ij}\|^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\mathbf{z}_{ij} - h(\mathbf{T}_i, \mathbf{p}_j)\|^2. \tag{212}$$

cost function 에서, $\mathbf{T}_i, \mathbf{p}_j$ 모두 최적화 변수이므로, 해당 최적화는 Pose와 Landmark 를 조정한다. 이를 BA(Bundle Adjustment)라 부르며, non linear optimization 을 통해 이 문제를 풀어보자.

7.2.2 Solving Bundle Adjustment

$h(\mathbf{T}, \mathbf{p})$ 가 비선형 함수 이기 때문에, 최적화를 위해서는 increment $\Delta\mathbf{x}$ 와 자코비안을 구해야 할 필요가 있다. 최적화 변수들을 모두 $\mathbf{x}$ 에 담아보자 .

$$\mathbf{x} = [\mathbf{T}_1, \ldots, \mathbf{T}_m, \mathbf{p}_1, \ldots, \mathbf{p}_n]^T. \tag{213}$$

이 경우 $\Delta\mathbf{x}$ 만 결정하면, 모든 최적화 변수의 increment을 구하는 것과 동일해 진다. 1차 Taylor expansion 이 아래 꼴임을 생각해보면,

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \mathbf{J}(\mathbf{x})^T \Delta\mathbf{x}.$$

Objective function은 다음과 같이 근사 된다. (아래식 에서는 $\mathbf{J}(\mathbf{x})^T \Delta\mathbf{x}$ 을 $\mathbf{T}_i$ 와 $\mathbf{p}_j$ 와 관련된 term으로 나누었다.)

$$\frac{1}{2}\|f(\mathbf{x} + \Delta\mathbf{x})\|^2 \approx \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}\|\mathbf{e}_{ij} + \mathbf{F}_{ij}\Delta\xi_i + \mathbf{E}_{ij}\Delta\mathbf{p}_j\|^2, \tag{214}$$

여기서 $\mathbf{F}_{ij}$는 i 번째 포즈에 대한 overall cost function 의 편도함수이고 $\mathbf{E}_{ij}$는 j 번째 landmark에 대한 overall cost function 의 편도함수이다.
만약 다음과 같이 $\mathbf{x} = [\mathbf{x}_c, \mathbf{x}_p]$ 로 분해 한다면,

$$\mathbf{x}_c = [\xi_1, \xi_2, \ldots, \xi_m]^T \in \mathbb{R}^{6m}, \tag{215}$$

$$\mathbf{x}_p = [\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n]^T \in \mathbb{R}^{3n}, \tag{216}$$

overal cost function 은 다음 같이 표현도 가능하다.

$$\frac{1}{2}\|f(\mathbf{x} + \Delta\mathbf{x})\|^2 = \frac{1}{2}\|\mathbf{e} + \mathbf{F}\Delta\mathbf{x}_c + \mathbf{E}\Delta\mathbf{x}_p\|^2. \tag{217}$$

Gauss-Newton, Levernberg-Marquatdt 상관없이, increment을 결정하기 위해서는 다음 increment eq

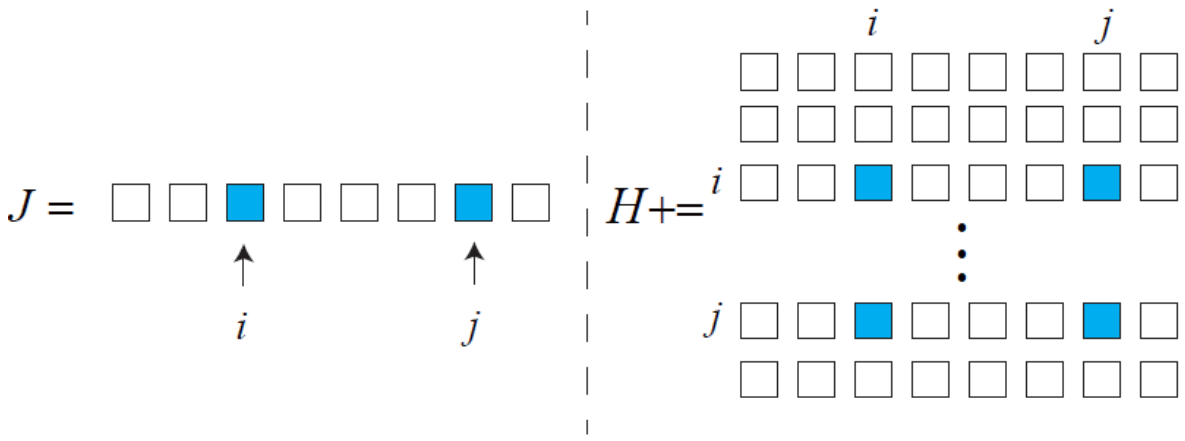

Figure 15: When an error term has sparsity in its Jacobian $\mathbf{J}$, it will also contribute to the sparsity of the Hessian $\mathbf{H}$.

을 풀어야 한다. (Gauss-Newton은 $\mathbf{H}$ 를 $\mathbf{J}^T\mathbf{J}$ 사용하고, Leverneberg는 $\mathbf{J}^T\mathbf{J} + \lambda\mathbf{I}$ 를 사용한다)

$$\mathbf{H}\Delta\mathbf{x} = \mathbf{g}. \tag{218}$$

좌변을 정리하기 위해 $\mathbf{J}^T\mathbf{J}$ 을 구해야 한다. 이를 위에서 표현한 partial derivative $\mathbf{F}$ $\mathbf{E}$ 으로 표현해보자.

$$\mathbf{J} = [\mathbf{F}\ \mathbf{E}]. \tag{219}$$

$$\mathbf{H} = \mathbf{J}^T\mathbf{J} = \begin{bmatrix} \mathbf{F}^T\mathbf{F} & \mathbf{F}^T\mathbf{E} \\ \mathbf{E}^T\mathbf{F} & \mathbf{E}^T\mathbf{E} \end{bmatrix}. \tag{220}$$

increment eq을 풀 때 한가지 문제점은 헤시안 행렬의 크기가 매우 크다는 것이다. (* motion and observation eq의 전체 개수 만큼의 사이즈를 가지고 있다.) 특히 increment 을 구하기 위해서는 헤시안 행렬의 역해렬을 양변에 곱해줘야 하는데 , 역 행렬을 구하는 과정은 $O(n^3)$ 복잡도를 가진다. 이 문제를 해결하기 위해 헤시안 행렬이 매우 Sparsity 하다는 성질을 이용한다.

7.2.3 Sparsity

error term $\mathbf{e}_{ij}$ 만 생각해보자. 이 에러는 $\mathbf{T}_i$ 와 $\mathbf{p}_j$ 에 의해 발생하는 에러이다. 따라서 이 error term의 자코비안은 다음과 같다. (* 자코비안은 $\mathbf{J} = [\mathbf{F}\ \mathbf{E}]$ 을 잊지 말자)

$$\mathbf{J}_{ij}(\mathbf{x}) = \left(\mathbf{0}_{2\times6}, ...\mathbf{0}_{2\times6}, \frac{\partial\mathbf{e}_{ij}}{\partial\mathbf{T}_i}, \mathbf{0}_{2\times6}, ...\mathbf{0}_{2\times3}, ...\mathbf{0}_{2\times3}, \frac{\partial\mathbf{e}_{ij}}{\partial\mathbf{p}_j}, \mathbf{0}_{2\times3}, ...\mathbf{0}_{2\times3}\right), \tag{221}$$

놀랍게도 이 자코비안은 $\mathbf{T}_i$ 및 $\mathbf{p}_j$ 에만 관련이 있고, 나머지 포즈나 landmark와는 상관이 없다. Fig 15 을 한 번 살펴보자. $\mathbf{J}_{ij}$는 i, j에서만 0이 아니다. 해당 자코비안으로 헤시안 행렬을 만들 경우 $(i,i), (i,j), (j,i), (j,j)$

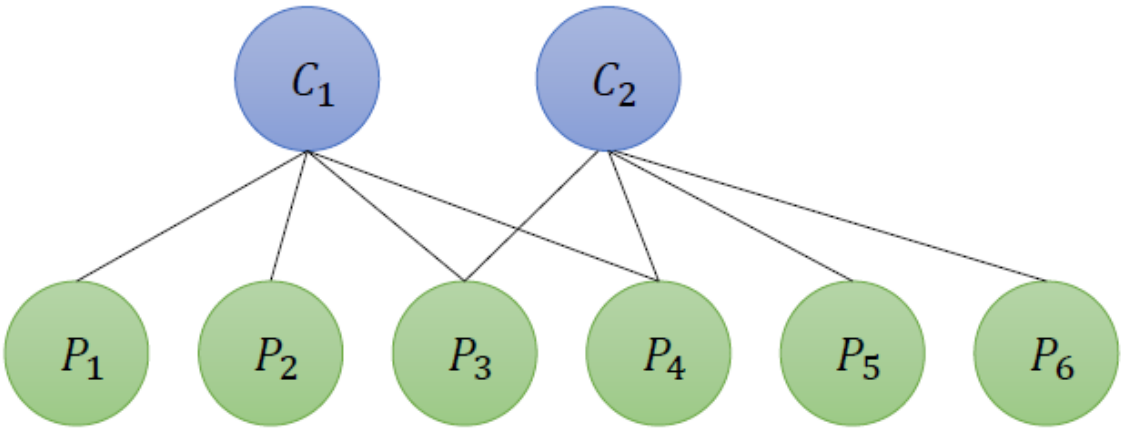

Figure 16: A minimal example of bundle adjustment.

에서만 0이 아니다. 헤시안 행렬 0이 아닌 블록만 있기 때문에 전체 Hessian 행렬 $\mathbf{H}$에 4개의 0이 아닌 블록을 추가합니다. 0이 아닌 블록은 $(i,i),(i,j),(j,i),(j,j)$에 있습니다.

Total Hessian matrix를 한 번 살펴보자. 각 error term에서 유도된 $\mathbf{J}_{ij}$ 을 사용하면 다음과 같다.

$$\mathbf{H} = \sum_{i,j} \mathbf{J}_{ij}^T \mathbf{J}_{ij}, \tag{222}$$

카메라 포즈와 landmark와 연관된 부분을 기준으로 4개의 블록으로 나눌수 있다.

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix}. \tag{223}$$

여기서 $\mathbf{H}_{11}$는 카메라 포즈에만 관련되고 $\mathbf{H}_{22}$는 랜드마크에만 관련된다.

1. i,j가 어떻게 변하든 $\mathbf{H}_{11}$는 항상 Block -diagonal matrix이다. $\mathbf{H}_{i,i}$에 0이 아닌 블록만 있다.
2. $\mathbf{H}_{22}$는 항상 Block -diagonal matrix이다. $\mathbf{H}_{j,j}$에 0이 아닌 블록만 있다.
3. $\mathbf{H}_{12}$ 및 $\mathbf{H}_{21}$의 경우 경우에 따라 희소하거나 조밀할 수 있다.

지금까지는 헤시안 행렬의 Sparsity에 대해 다뤘다. 다음 챕터에서는 Sparsity를 사용해서 incremental eq을 풀어보자.

7.2.4 Minimal Example of BA

Fig. 16 와 같이, 2개의 카메라 포즈(C_1, C_2)와 6개의 랜드마크 $(P_1, P_2, P_3, P_4, P_5, P_6)$ 가 있다고 하자. 이러한 카메라와 landmark 에 해당하는 변수는 $\mathbf{T}_i, i = 1, 2$ 와 $\mathbf{p}_j, j = 1, \cdots, 6$ 이다. 카메라 C_1는 랜드마크

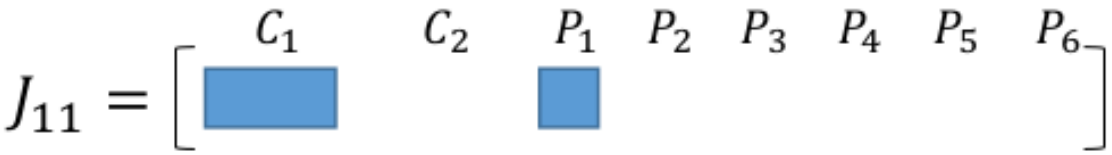

Figure 17: The distribution of non-zero blocks of the $\mathbf{J}_{11}$ matrix. The upper mark indicates the variable corresponding to that column of the matrix. Since the pose dimension is larger than the landmark dimension, the matrix block corresponding to C_1 is wider than the matrix block corresponding to P_1.

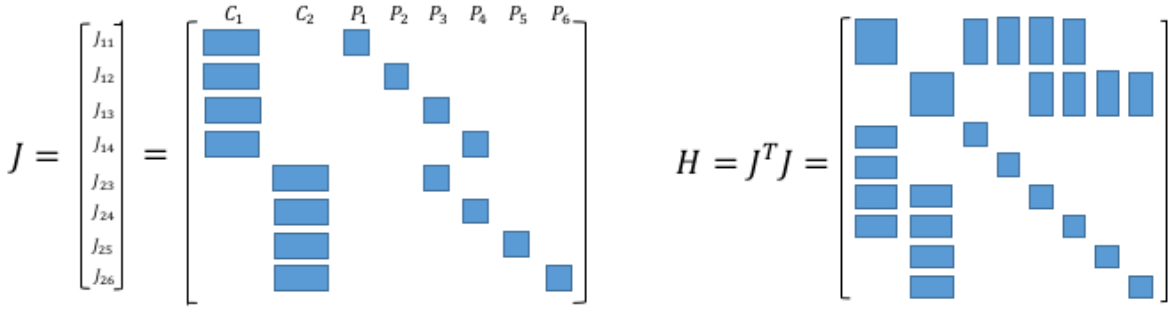

Figure 18: The sparsity of the Jacobian matrix (left) and the sparsity of the $\mathbf{H}$ matrix (right). The colored squares indicate that the matrix has a non-zero value in the corresponding matrix block, and the rest of the uncolored parts indicate that the value of is always 0.

P_1, P_2, P_3, P_4를 관찰하고 카메라 C_2는 P_3, P_4, P_5, P_6을 관찰한다.

이 때, overall cost function은 다음과 같다.

$$\frac{1}{2}\left(\|\mathbf{e}_{11}\|^2 + \|\mathbf{e}_{12}\|^2 + \|\mathbf{e}_{13}\|^2 + \|\mathbf{e}_{14}\|^2 + \|\mathbf{e}_{23}\|^2 + \|\mathbf{e}_{24}\|^2 + \|\mathbf{e}_{25}\|^2 + \|\mathbf{e}_{26}\|^2\right). \tag{224}$$

구조를 확인하기 위해, $\mathbf{e}_{11}$ 만 살펴보자. $\mathbf{J}_{11}$를 $\mathbf{e}_{11}$에 해당하는 Jacobian 행렬이라고 하면, $\mathbf{x} = (\xi_1, \xi_2, \mathbf{p}_1, \cdots, \mathbf{p}_2)^T$ 이므로, $\mathbf{J}_{11}$ 는 다음과 같다.

$$\mathbf{J}_{11} = \frac{\partial \mathbf{e}_{11}}{\partial \mathbf{x}} = \left(\frac{\partial \mathbf{e}_{11}}{\partial \xi_1}, \mathbf{0}_{2\times 6}, \frac{\partial \mathbf{e}_{11}}{\partial \mathbf{p}_1}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3}\right). \tag{225}$$

그림으로 표현하면, $\mathbf{J}_{11}$ 은 FIg. 17 로 그려진다.

전체 자코비안과 헤시안은 Fig. 18 와 같은 행렬 구조로 표현된다.

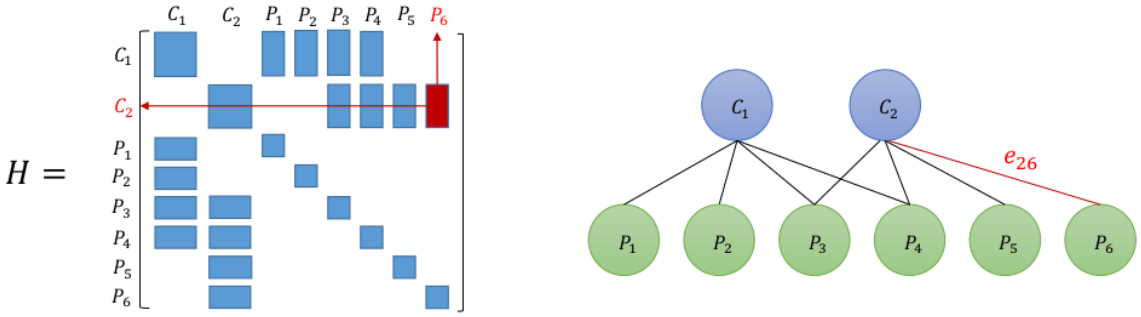

Figure 19: The corresponding relationship between the non-zero matrix blocks in the **H** matrix and the edges in the graph. For example, the red matrix block on the right in the **H** matrix in the left picture indicates that there is an edge between the corresponding variables C_2 and P_6 in the right picture $\mathbf{e}_{26}$.

7.2.5 Schur Trick

실제로 **H**의 희소성을 사용하여 $\mathbf{H}\Delta\mathbf{x} = \mathbf{g}$ 의 해를 구해보자. 헤시안 행렬은 다음과 같은 4개의 블록으로 나눌 수 있다.

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} \\ \mathbf{H}_{21} & \mathbf{H}_{22} \end{bmatrix}.$$

편의를 위해 Fig. 20 와 같이 4개의 블록을 $\mathbf{B}, \mathbf{E}, \mathbf{E}^T, \mathbf{C}$ 라 하자. 이 때 incremental eq $\mathbf{H}\boldsymbol{\Delta}\mathbf{x} = \mathbf{g}$ 는 다음과 같이 표현된다.

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E^T} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix}, \tag{226}$$

$\mathbf{C}$ 는 각 블록이 3×3 인 block diagonal matrix인데, $\mathbf{C}$ 는 역행렬을 구하기는 것이 상대적으로 쉽다. 블록 하나씩 개별적으로 역행렬을 구하면 되기 때문이다. 그렇다면 $\mathbf{C}^{-1}$ 이 포함된 행렬을 곱하는 것에 부담갖지 않아도 된다.

$$\begin{bmatrix} \mathbf{I} & -\mathbf{EC^{-1}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E^T} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{EC^{-1}} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix}. \tag{227}$$

$$\begin{bmatrix} \mathbf{B} - \mathbf{EC}^{-1}\mathbf{E}^T & \mathbf{0} \\ \mathbf{E}^T & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{v} - \mathbf{EC}^{-1}\mathbf{w} \\ \mathbf{w} \end{bmatrix}. \tag{228}$$

그러면 우리는 $\Delta\mathbf{x}_c, \Delta\mathbf{x}_p$ 각각에 linear eq을 얻게 된다.

$$\left[\mathbf{B} - \mathbf{EC}^{-1}\mathbf{E}^T\right] \Delta\mathbf{x}_c = \mathbf{v} - \mathbf{EC}^{-1}\mathbf{w}. \tag{229}$$

$$\Delta\mathbf{x}_p = \mathbf{C}^{-1}(\mathbf{w} - \mathbf{E}^T\Delta\mathbf{x}_c) \tag{230}$$

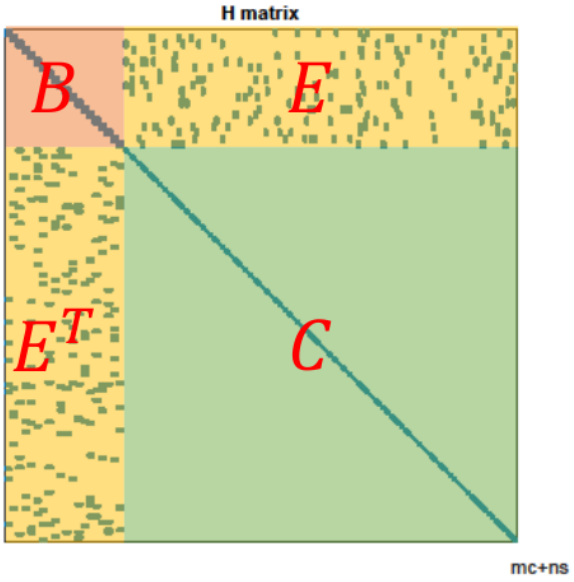

Figure 20: Blocks in H matrix.

위와 같이 sparsity한 특성을 확용하여 incremental을 결정하는 것을 marginalization 또는 *Schur Elimination* (Schur trick) 이라 부른다. 아쉽게도 $\Delta \mathbf{x}_c$ 에 관련된 linear eq을 푸는 trick에 대해서는 밝혀져 있지 않다.

7.2.6 Robust Kernels

앞서 다룬 BA 문제에서는 objective function에서 error term 의 $\mathcal{L}_2$ norm을 최소화 하였다. 하지만 매칭이 잘못 되었거나, 이외의 여러가지 이유로 특정 error term이 커지게 된다면, 모든 변수가 잘못된 방향으로 최적화 될것이다. $\mathcal{L}_2$ norm이 커질수록 , error가 훨씬 커지는 특성에서 기인한다. 그래서 커널 함수가 제안되었다. 가장 일반적인 Huber 커널을 살펴보면 다음과 같다.

$$H(e) = \begin{cases} \frac{1}{2}e^2 & \text{when} |e| \leqslant \delta, \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{231}$$

8 Filters and Optimization Approaches: Part II

이전 챕터에서는 BA기반의 graph optimization을 집중적으로 다뤘다. 하지만 랜드마크의 수가 많아질수록 계산 효율은 급격하게 떨어지는 단점이 존재한다. 따라서 이번 챕터에서는 그 단점을 개선한, Pose graph 방법을 소개하고자 한다.

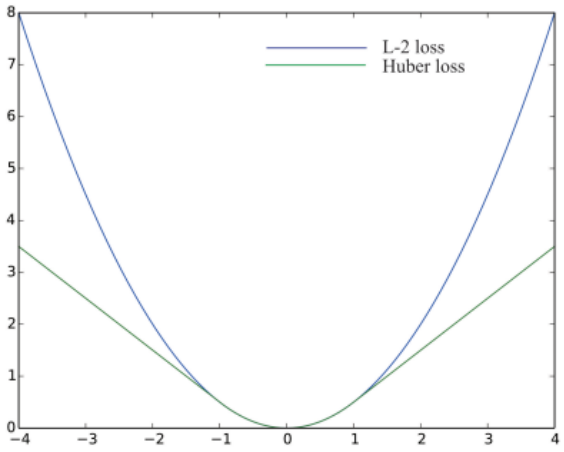

Figure 21: The Huber kernel.

9 Sliding Window Filter and Optimization

9.1 Controlling the Structure of BA

continuous video 환경에서 computing scale을 조절하기 위해, 여러가지 방법들이 도입되었다. keyframe과 landmark에 대해서만 BA을 계산하고, keyframe이 아닐 경우 localization 에만 사용하고 mapping에는 활용하지 않는 방식이 대표적이다. 하지만 keyframe만 BA에 활용함에도 불구하고, 시간이 지나면 keyframe의 개수가 계속해서 늘어나기 때문에 Computing scale이 증가하게 된다.

이를 발지하는 가장 간단한 방법은 현재와 가장 가까운 N 개의 keyframe만 BA에서 유지하고, 이전 keyframe의 제거하는 바식이다. 이러한 방법을 Sliding Window method라 부른다. 알고리즘에 따라 N 개의 keyframe을 선택하는 방법이 달라지는데, 예를 들어 시공간적으로 가까운 keyframe을 유지하거나, ORB-SLAM2 와 같이 co-visibility graph을 사용하여 선택하기도 한다.

위에서 설명한 어떤 방법을 선택하든, 결국은 모든 frame을 한 번에 BA 작업에 사용할 수 없기 때문에 생긴 절충안이다. 하지만 일부 frame만 BA에 활용하면서 생긴 새로운 문제가 있다. 바로 BA에서 사용하지 않는 외부변수(e.g. BA에 사용하지 않는 Frame의 pose)을 다루는 방법이다.

9.2 Sliding Window

먼저 Sliding Window 방법에 대해 생각해보자. BA에 N 개의 keyframe을 사용할 때, 각 keyframe의 pose가 다음과 같다고 하자.

$$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N,$$

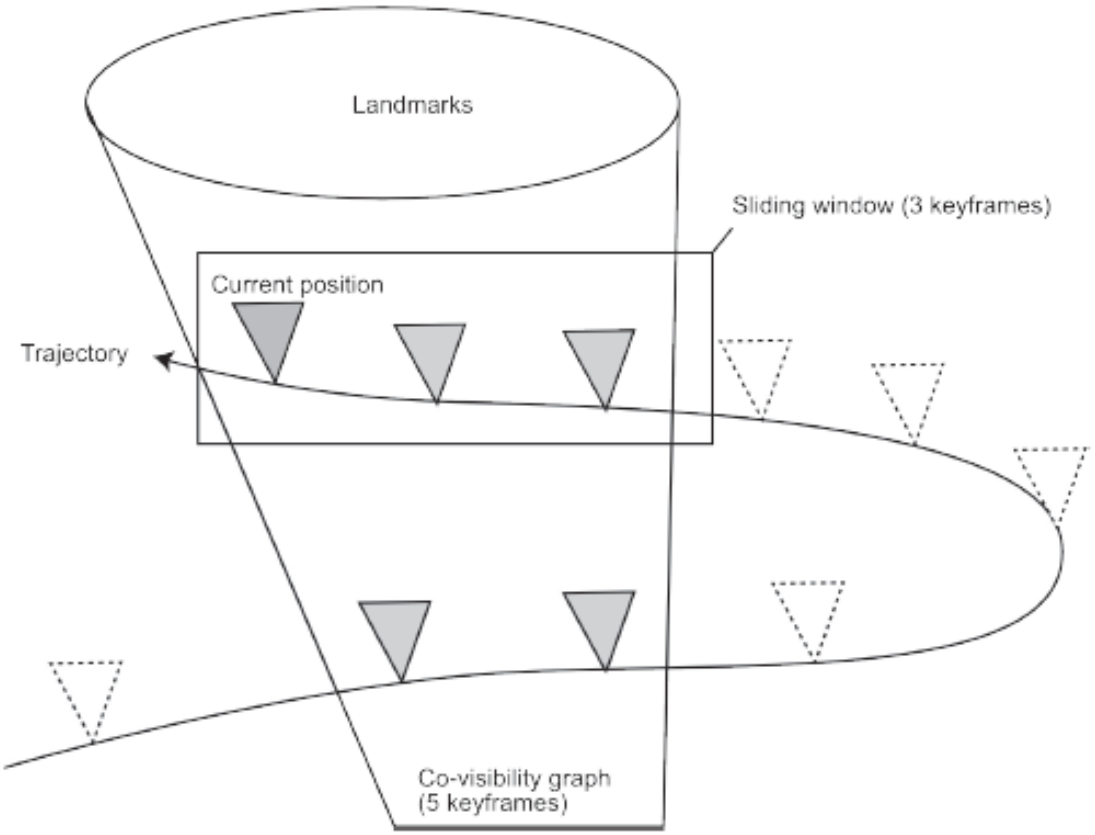

Figure 22: Co-visibility graph in sliding window.

우리가 궁금한 것은 각 keyframe의 위치와 uncertainty이다. 즉 state을 가우시안 분포로 가정한다면, 평균과 공분산에 해당한다. 마찬가지로 BA에서 M 개의 landmark point을 사용한다고 가정하자.

$$\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N$$

그렇다면 우리과 관심있는 것은 다음과 같다.

$$[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N | \boldsymbol{y}_1, \ldots, \boldsymbol{y}_M] \sim N([\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_N]^T, \boldsymbol{\Sigma}).$$

여기서 관심사는 "윈도우가 변경될 때(즉 BA내의 keyframe이 하나 삭제되고, 새로운 keyframe이 추가될때) state variable이 어떻게 바뀌야 하는지" 이다.

1. We want to add a new keyframe into the window as well as its corresponding landmarks.
2. We need to delete an old keyframe in the window, and may also delete the landmarks it observes.

TODO