SLAM十四讲笔记

Xin Li

Contents

Notations

Notation	Meaning
a	Scalar
$\mathbf{a}$	Vector
$\mathbf{A}$	Matrix
$(\cdot)^T$	Matrix transpose
$(\cdot)^{-1}$	Matrix inverse
$\mathcal{E}\{\cdot\}$	Expectation
$\|\mathbf{a}\|$	Euclidean norm of vector $\mathbf{a}$
$\|\mathbf{A}\|_F$	Frobenius norm of matrix $\mathbf{A}$
$\mathbb{R}$	Set of real numbers
$\mathbb{C}$	Set of complex numbers

Matrix Transformation

Coordinates & Basis

A point in the real Cartesian space $\mathbb{R}^3$ can be described by the basis $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$, as

$$a = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = a_1\mathbf{e}_1 + a_2\mathbf{e}_2 + a_3\mathbf{e}_3. \tag{1}$$

Inner/Outer Product

For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$, their inner product is defined as

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| \cos \langle \mathbf{a}, \mathbf{b} \rangle \tag{2}$$

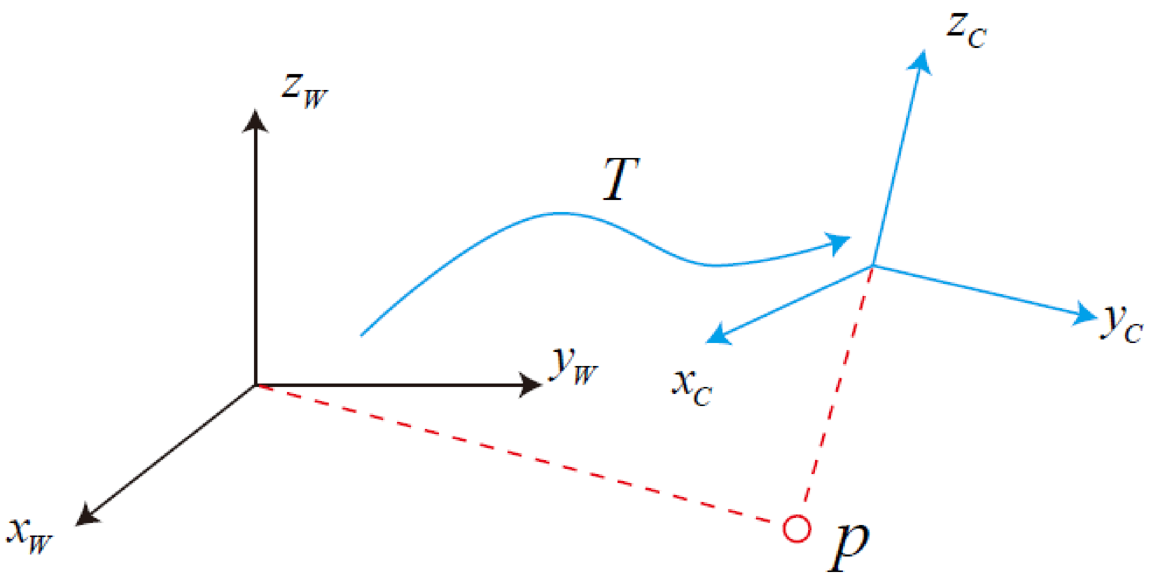
and their outer product is

$$\begin{aligned} \mathbf{a} \times \mathbf{b} &= \begin{bmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_2b_3 - a_3b_2 \\ a_3b_1 - a_1b_3 \\ a_1b_2 - a_2b_1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \mathbf{b} \\ &\triangleq \mathbf{a} \wedge \mathbf{b} \end{aligned} \tag{3}$$

which is orthogonal to the vectors $\mathbf{a}$ and $\mathbf{b}$. Here, $\mathbf{a}\wedge$ is a "Skew-symmetric" (or anti-symmetric) matrix.

Euclidean Transformation

Suppose the world coordinates (x_w, y_w, z_w) are stationary while the robot can be indicated by a moving coordinates (x_c, y_c, z_c). Consider a vector $\mathbf{p} \in \mathbb{R}^3$ in the figure below:



We can represent the vector using those two different coordinates. Assume the world coordinates are described by the basis $(\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3)$ and the robot coordinates are described by the basis $(\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3)$, the vector $\mathbf{p}$ will not change using the representations from those two bases, i.e.,

$$[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3] \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = [\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3] \begin{bmatrix} a'_1 \\ a'_2 \end{bmatrix}. \tag{4}$$

$$\begin{bmatrix} a_3 \end{bmatrix} \qquad \begin{bmatrix} a_3' \end{bmatrix}$$

Then, multiply the two sides with $[\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]^T$,

$$\begin{aligned} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} &= [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]^T [\mathbf{e}_1', \mathbf{e}_2', \mathbf{e}_3'] \begin{bmatrix} a_1' \\ a_2' \\ a_3' \end{bmatrix} \\ &\triangleq \mathbf{R}\mathbf{a}'. \end{aligned} \tag{5}$$

where $\mathbf{R}$ denotes the ***rotation matrix***, which contains certain special properties. We can define the set of the rotation matrix as

$$SO(n) = \{\mathbf{R} \in \mathbb{R}^{n \times n} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, \det(\mathbf{R}) = 1\} \tag{6}$$

where $SO(n)$ stands for "***Special Orthogonal Group***". Basically, the rotation matrix can describe the rotation of certain object. We can also perform the inverse operation (the reversed rotation) by $\mathbf{R}^{-1}\mathbf{a}$ (equivalent to $\mathbf{R}^T\mathbf{a}$ since $\mathbf{R}$ is symmetric) to obtain the vector $\mathbf{a}'$.

Moreover, the Euclidean transformation also includes the translation of the robot's location, and thus we also need the translation vector $\mathbf{t} \in \mathbb{R}^3$ to the original expression, i.e.,

$$\mathbf{a}' = \mathbf{R}\mathbf{a} + \mathbf{t}. \tag{7}$$

Overall Transform Matrix

We can represent above Euclidean transformation into a ***homogeneous coordinates*** form, as

$$\begin{bmatrix} \mathbf{a}' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix} \triangleq \mathbf{T} \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix} \tag{8}$$

which allows us to represent the overall transform in a linear form, and in the rest of the notes, I will implicitly represent the homogeneous coordinates $[\mathbf{a}, 1]^T$ by $\mathbf{a}$ for simplicity. In addition, the set of the transform matrix $\mathbf{T}$ can be defined as

$$SE(3) = \left\{ \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} | \mathbf{R} \in \mathbb{R}^{3 \times 3}, \mathbf{t} \in \mathbb{R}^3 \right\} \tag{9}$$

where $SE(3)$ denotes the ***special Euclidean group***.

Rotation Vector & Euler Angles

In $SE(3)$, we employ totally 16 entries to describe the Euclidean transform of the object, and in $SO(3)$ we employ 9 entries to describe the rotation with 3 DoFs, which can be redundant. A more compact representation of the rotation transform can be

- Rotation axis + rotation angle
- Euler angles

which will be introduced as follows.

Rotation Vector

Suppose the rotation axis is given by the vector $\mathbf{n}$, and the rotation angle is θ, the rotation vector can be represented by $\theta\mathbf{n}$, which relates the rotation matrix using the Rodrigues's Formula [1], as

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T + \sin\theta\mathbf{n}\wedge \tag{10}$$

where $\wedge$ is the operator that transform the vector to the anti-symmetric matrix. From this relation, we can also retrieve the rotation angle by

$$\begin{aligned}\operatorname{tr}(\mathbf{R}) &= \cos\theta \operatorname{tr}(\mathbf{I}) + (1-\cos\theta)\operatorname{tr}\left(\mathbf{n}\mathbf{n}^T\right) + \sin\theta \operatorname{tr}(\mathbf{n}\wedge) \\ &= 3\cos\theta + (1-\cos\theta) \\ &= 1 + 2\cos\theta \end{aligned} \tag{11}$$

and therefore we can obtain the rotation angle by

$$\theta = \cos^{-1}\left\{\frac{\operatorname{tr}(\mathbf{R}) - 1}{2}\right\} \tag{12}$$

In addition, if we rotate the rotation axis $\mathbf{n}$ by the matrix $\mathbf{R}$, the result are still $\mathbf{R}\mathbf{n} = \mathbf{n}$. Therefore, $\mathbf{n}$ is the eigenvector of the matrix $\mathbf{R}$ which corresponds to the eigenvalue 1. We can use eigen decomposition to find the rotation axis.

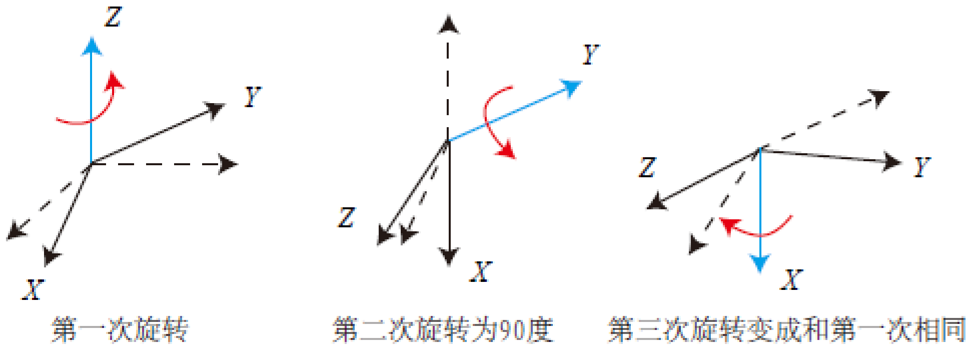
Euler Angles

Euler angles can be defined in many different ways. Here we employ the "yaw-pitch-roll" in the UAV area, which is equivalent to "Z-Y-X" rotation, i.e.,

- rotate about Z axis, we have **yaw angle**
- rotate about the Y axis after the first rotation, we have **pitch angle**
- rotate about the X axis after the second rotation, we have **roll angle**

and we can also use $[r, p, y]^T$ vector to describe the Euler angles.

However, the Euler angles suffer from the well-known ***Gimbal Lock*** problem:



As illustrated in the graph above, if we rotate in this way, the third rotation will cause ambiguity: That is, rotate about X axis will be equivalent to the case if we rotate Z axis at the first step. This will lead to a decrease in the DoFs of the system (from DoF=3 to DoF=2). Therefore, Euler angles are usually used for the visualization purpose.

Quaternions

In order to avoid ambiguity, we need to extend the number of variables to express the 3D rotation, which is called "quaternions".

A quaternion $\mathbf{q}$ contains 1 real part & 3 imaginary parts, i.e.,

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k, \tag{13}$$

where

$$i^2 = j^2 = k^2 = -1 \tag{14}$$
$$ij = k, ji = -k, \tag{15}$$
$$jk = i, kj = -i, \tag{16}$$
$$ki = j, ik = -j \tag{17}$$

and thus we can express the quaternions as a vector $\mathbf{q} = [s, \mathbf{v}^T]^T$, where $\mathbf{v} = [q_1, q_2, q_3]^T$ and $s = q_0$.

C++ Eigen 3 Implementation

```cpp
#include<iostream>
#include<Eigen/Eigen>
#include<Eigen/Core>
#include<Eigen/Geometry>
#include<cmath>
/**
 * @brief SLAM 14讲 第三章
 *
 * 复习eigen库有关知识: 向量变换
 *
 * @return int
 */

int main()
{
    // create a matrix
    Eigen::MatrixXd mat = Eigen::MatrixXd::Identity(5,5);
    Eigen::MatrixXd covMat = mat*mat.transpose();

    // Eigendecomposition
    Eigen::SelfAdjointEigenSolver<Eigen::MatrixXd> eigen_solver(covMat);

    //std::cout.precision(3);
    std::cout<<"Covariance Matrix = \n"<<covMat<<std::endl;
    std::cout<<"Eigenvalues = \n"<<eigen_solver.eigenvalues()<<std::endl;
    std::cout<<"Eigenvectors = \n"<<eigen_solver.eigenvectors()<<std::endl;

    /**
     * @brief
     * The code below is about the Eigen/Geometryw
     *
     */
    // For the rotation matrix, we directly apply the matrix3d (or matrixXd)
    Eigen::Matrix3d rotation_mat = Eigen::Matrix3d::Identity(3,3);
    // Angle axis rotation
    Eigen::AngleAxisd rot_vec(EIGEN_PI/4, Eigen::Vector3d(0,0,1));
    // Euler angles (from rotation matrix)
    Eigen::Vector3d eulerAngles = rotation_mat.eulerAngles(2,1,0); // ZYX order


    // To rotation matrix
    std::cout<<"From axis angle to the rotation matrix\n"<<rot_vec.toRotationMatrix()<<std::endl;
    std::cout<<"From the rotation matrix to the Euler angles\n"<<eulerAngles.transpose()<<std::endl;

    // Euclidean transform matrix
    Eigen::Isometry3d T = Eigen::Isometry3d::Identity(); // 4x4 matrix
    T.rotate(rot_vec);
    T.pretranslate(Eigen::Vector3d(1,3,4));// pretranslate = 提前位移, 坐标系依然是世界坐标系
    std::cout<<"Euclidean tranformation matrix\n"<<T.matrix()<<std::endl;

    //Quaternions
    Eigen::Quaterniond q1(1,0,0,0); // 直接赋值
    std::cout<<"Quaternion from the direct values\n"<<q1.coeffs()<<std::endl;
    Eigen::Quaterniond q2(rot_vec); // 从angleAxis赋值
    std::cout<<"Quaternion from the angleAxis\n"<<q2.coeffs()<<std::endl;

    //rotate other vector
    std::cout<<"(1,0,0) after rotation = "<<(q2*Eigen::Vector3d(1,0,0)).transpose()<<std::endl;

    return 0;
}
```

Lie Group & Lie Algebra

In the previous chapter, we introduced the concepts of the $SO(3)$ and $SE(3)$:

$$SO(n)=\{\mathbf{R}\in\mathbb{R}^{n\times n}|\mathbf{R}\mathbf{R}^T=\mathbf{I},\det(\mathbf{R})=1\} \tag{18}$$

$$SE(3)=\left\{\mathbf{T}=\begin{bmatrix}\mathbf{R} & \mathbf{t}\\ \mathbf{0}^T & 1\end{bmatrix}\in\mathbb{R}^{4\times 4}|\mathbf{R}\in\mathbb{R}^{3\times 3},\mathbf{t}\in\mathbb{R}^3\right\} \tag{19}$$

where we can find that for two arbitrary rotation matrices $\mathbf{R}_1$ amd $\mathbf{R}_2$:

$$\mathbf{R}_1+\mathbf{R}_2\notin SO(3) \tag{20}$$

$$\mathbf{R}_1\mathbf{R}_2\in SO(3) \tag{21}$$

which implies that their addition operation is not closed but the multiplication operation is closed. Actually, we call the set with only 1 type of operation as "group". Next we'll explain ***group*** in details.

Definition of Group

A group is an algebraic structure of one set plus one operator. If we denote a set $\mathbb{A}$ and an operator $(\cdot)$, the group can be defined as $G=(\mathbb{A},\cdot)$. We say G is a group if it satisfies the following conditions:

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

Lie Algebra

Consider an arbitrary rotation matrix $\mathbf{R}$, it satisfies

$$\mathbf{R}\mathbf{R}^T=\mathbf{I} \tag{22}$$

Now, assume we want to describe the rotation of a moving object, the rotation matrix should be a continuous-time function $\mathbf{R}(t)$, and thus we have

$$\mathbf{R}(t)\mathbf{R}^T(t)=\mathbf{I} \tag{23}$$

Take the derivative of this expression on both sides with respect to t, as

$$\dot{\mathbf{R}}(t)\mathbf{R}^T(t) + \mathbf{R}(t)\dot{\mathbf{R}}^T(t) = \mathbf{0} \tag{24}$$

which can be simplified as

$$\dot{\mathbf{R}}(t)\mathbf{R}^T(t) = -(\dot{\mathbf{R}}(t)\mathbf{R}^T(t))^T \tag{25}$$

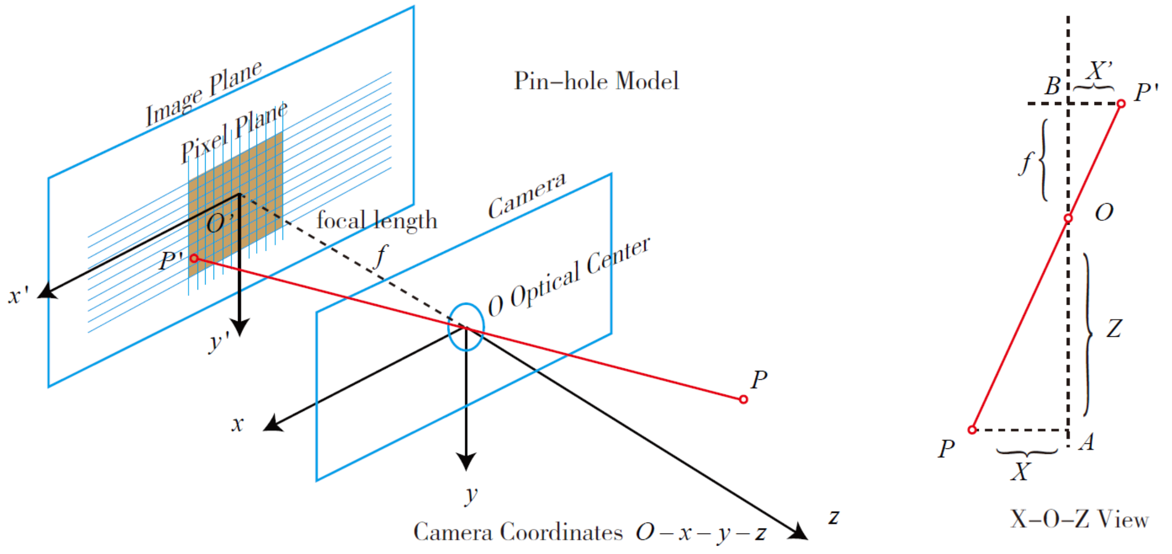
where we can find that the matrix $\dot{\mathbf{R}}(t)\mathbf{R}^T(t)$ is anti-symmetric. Recall that when we introduce the concepts of the inner product, an anti-symmetric matrix can be obtained by taking $(\cdot)^\wedge$ of a vector. Therefore, we denote the anti-symmetric matrix as

$$\phi(t)^\wedge \triangleq \dot{\mathbf{R}}(t)\mathbf{R}^T(t) \tag{26}$$

Camera & Images

In the previous chapters, we introduced how to express the robot's motion. This chapter will mainly focus on how the robot observe the outside world using the stereo camera, which is basically related to the image projection.

Pinhole Camera Models



Camera Calibration

Nonlinear Optimization

contents

Visual Odometry: Key Points

contents

Visual Odometry: Optical Flow

1. Refer to https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula ↩