《十四讲》实践习题 U3-4

创建时间:	2020/3/26 13:55	**更新时间:**	2020/3/30 22:20
作者:	293311923@qq.com		
URL:	https://github.com/AceCoooool/slambook/blob/master/code/ch3/README.md		

第三单元

Eigen安装

① 方式1（直接apt-get安装）

`sudo apt-get install libeigen3-dev`

② 方式2（官网下载安装）

1. 从Eigen官网下载最新的版本
2. 在下载的目录下面执行下述目录

```
# 解压到/usr/include中
sudo tar -xzvf eigen-eigen-xxxxx.tar.gz -C /usr/include
# 更改命名
sudo mv /usr/include/eigen-eigen-xxxx /usr/include/eigen3
# 将Eigen及其子目录移动到include下面（个人觉得没必要）
# sudo cp -r /usr/include/eigen3/Eigen /usr/include
```

> 注意：opencv_contrib安装对eigen的版本有需求，比如opencv-3.4.5采用3.3.7的Eigen会报错！

检测版本的demo:

```
#include <iostream>
#include "Eigen/Dense"
using namespace std;

int main() {
    cout << "Eigen versions: " << endl;
    cout << EIGEN_WORLD_VERSION << "." << EIGEN_MAJOR_VERSION << "." << EIGEN_MINOR_VERSION << endl;
    return 0;
}
```

Pangolin安装

1. 从github上面下载：

   ```
   git clone https://github.com/stevenlovegrove/Pangolin.git
   ```

2. 安装依赖环境：

   ```
   sudo apt-get install libglew-dev
   ```

3. 编译：

```
mkdir build
cd build
cmake ..
make
```

> 这里需要注意的是，如果当前系统环境采用的是`Anaconda`，那么可能会出现错误（一般是因为你可能`conda`安装过`libpng`，所以导致环境调用的是Anaconda下面的`libpng`）。最简单的方式是就是切换到`system`后再重新编译安装

验证旋转矩阵的正交矩阵

根据式子3.5我们知道旋转矩阵可以表示成：

$$R = \begin{bmatrix} e_1^T e_1^{'} & e_1^T e_2^{'} & e_1^T e_3^{'} \\ e_2^T e_1^{'} & e_2^T e_2^{'} & e_2^T e_3^{'} \\ e_3^T e_1^{'} & e_3^T e_2^{'} & e_3^T e_3^{'} \end{bmatrix}$$

要验证R为正交矩阵，等价于验证$RR^T = I$。那我们不妨先写出$A = RR^T$的形式（为了简便，只写出A_{ij}）：

$$A_{ij} = e_i^T e_1^{'} e_1^{'T} e_j^{'} + e_i^T e_2^{'} e_2^{'T} e_j^{'} + e_i^T e_3^{'} e_3^{'T} e_j^{'} = (e_1^{'} + e_2^{'} + e_3^{'})^T (e_i e_j^T)(e_1^{'} + e_2^{'} + e_3^{'})$$

① 当$i = j$时：

$$A_{ii} = e_i^T e_1^{'} e_1^{'T} e_i^{'} + e_i^T e_2^{'} e_2^{'T} e_i^{'} + e_i^T e_3^{'} e_3^{'T} e_i^{'} = (e_1^{'} + e_2^{'} + e_3^{'})^T (e_i e_i^T)(e_1^{'} + e_2^{'} + e_3^{'})$$

由于$e_i e_i^T$只有第i行第i列为1，其余均为0；所以相当于$(e_1^{'} + e_2^{'} + e_3^{'})$第i行的平方；而又因为$e_1^{'}, e_3^{'}, e_3^{'}$为单位正交坐标系，所有其每行之和均为1。所以有$A_{ii} = 1$

② 当$i != j$时：

$$A_{ij} = (e_1^{'} + e_2^{'} + e_3^{'})^T (e_i e_j^T)(e_1^{'} + e_2^{'} + e_3^{'})$$

因为$i != j$，所以$e_i e_j^T = 0$；所以$A_{ij} = 0$

所以得证！！！

> 注：e_1, e_2, e_3只要为单位正交坐标系即可（不必$[1,0,0],[0,1,0],[0,0,1]$），但上述为了简化，采用的e_1, e_2, e_3为$[1,0,0],[0,1,0],[0,0,1]$

寻找罗德里格斯公式的推导过程并加以理解

> 来自：[罗德里格斯公式 理解、推导](), [wiki]()

罗德里格斯公式（Rodriguez formula公式为：

$$R = I + sin(\theta)K + (1 - cos(\theta))K^2$$

① 由于要描述的都是旋转问题，那么先来看下什么是旋转

说白了就是坐标系进行了刚性旋转（坐标系B到坐标系C，不妨假设坐标系B为$[1,0,0;0,1,0;0,0,1]$），那么对应的坐标转换矩阵R存在下述关系：

$$C = RB = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix} [b_x \quad b_y \quad b_z]$$

> 其中的b_x等均为了列向量

根据线性代数的定义，旋转矩阵R就是从基向量矩阵B到基向量矩阵C的过渡矩阵。由于旋转矩阵R是标准3阶正交矩阵，故旋转矩阵R的自由度为3，这说明最少可以用三个变量来表示旋转矩阵R，这就是**罗德里格斯公式（Rodriguez formula）**存在的基础。

罗德里格斯公式（Rodriguez formula首先要确定一个三维的单位向量$k = [k_x, k_y, k_z]^T$（两个自由度---因为单位向量）和一个标量θ（一个自由度）

② 证明罗德里斯公式

先考虑对一个向量作旋转，其中 v 是原向量，三维的单位向量 $k = [k_x, k_y, k_z]^T$是旋转轴， θ是旋转角度，v_{rot}是旋转后的向量。

1. 将v分解成$v_{//}$(相对k)和$v_\perp$：（顺便定义与k，v垂直的向量）

$$v_{//} = (v\cdot k)k v_\perp = v - v_{//} = -k\times(k\times v) w = k\times v$$

2. 不难看出来$v_{//}, v_\perp, w$三者互相正交。从而不难通过图上的关系知：

$$v_{rot} = v_{//} + cos(\theta)v_\perp + sin(\theta)w$$

3. 将1中$v_{//}$改写成×乘的形式$v_{//} = v + k\times(k\times v)$（其实就是1中第二条式子）。在代入2不难得到：

$$v_{rot} = v + k\times(k\times v) - cos(\theta)k\times(k\times v) + sin(\theta)k\times v$$

4. 正如式子3.3引入的$\wedge$符号，我们也可以将k表示成$K = k\wedge$：（满足$k\times v = Kv$）

$$\begin{bmatrix} 0 & -k_z & k_y \\ k_z & 0 & -k_x \\ -k_y & k_x & 0 \end{bmatrix}$$

5. 将4代入3中，就可以“改写”成：

$$v_{rot} = v + (1 - cos(\theta))K^2 v + sin(\theta)Kv = (I + (1 - cos(\theta))K^2 + sin(\theta)K)v$$

> 关于$k\times(k\times v) = K^2 v$你可以自己带进去验证一下

所以就可以得到：$R = I + sin(\theta)K + (1 - cos(\theta))K^2$

> 但显然上面的式子和课本上的3.14形式上并不一样。下面我们可以稍微变化一下。
>
> 对于2，我们可以写成下述等价形式：
>
> $$v_{rot} = v_{//} + cos(\theta)v_\perp + sin(\theta)w = v_{//} + cos(\theta)(v - v_{//}) + sin(\theta)k\times v = cos(\theta)v + (1 - cos(\theta))v_{//} + sin(\theta)k\times v$$
>
> 其中$v_{//}$是v在k上的投影，所有可以写成$kk^T v/(k^T k) = kk^T/v$，所以代入就可以得到：
>
> $$v_{rot} = (cos(\theta) + (1 - cos(\theta))kk^T + sin(\theta)k\wedge)v$$
>
> 所以有$R = cos(\theta) + (1 - cos(\theta))kk^T + sin(\theta)k\wedge$

验证四元数旋转某个点后，结果是一个虚四元数（实部为零），所以仍然对应到一个三维空间点（式3.34）

其实题目的意思就是证明$p' = qpq^{-1}$的实部等于0。再证明之前，我们先给出各个四元数：

$$q = [cos\frac{\theta}{2}, cos\frac{\theta}{2}, cos\frac{\theta}{2}, cos\frac{\theta}{2}] p = [0, x, y, z] q^{-1} = [cos\frac{\theta}{2}, -cos\frac{\theta}{2}, -cos\frac{\theta}{2}, -cos\frac{\theta}{2}]/(4(cos\frac{\theta}{2})^2)$$

> 关于q^{-1}请看式子3.29

下面我们将四元数的乘法（式子3.23写成矩阵形式）：

$$q_a q_b = [s_a, x_a, y_a, z_a]\begin{bmatrix} s_b & x_b & y_b & z_b \\ -x_b & s_b & -z_b & y_b \\ -y_b & z_b & s_b & -x_b \\ -z_b & -y_b & x_b & s_b \end{bmatrix}$$

那么现在写出qpq^{-1}的式子：

$$qpq^{-1} = cos\frac{\theta}{2}[1, 1, 1, 1]\begin{bmatrix} 0 & x & y & z \\ -x & 0 & -z & y \\ -y & z & 0 & -x \\ -z & -y & x & 0 \end{bmatrix} cos\frac{\theta}{2}/(4(cos\frac{\theta}{2})^2)\begin{bmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 \\ 1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

很显然可以知道实部的结果为：

$$[-x - y - z, z + y - x, x + y - z, z + y - x][1, 1, 1, 1]^T = 0$$

所以得证。

假设有一个大的Eigen矩阵，想把它的左上角3×3的块取出来，然后赋值为$I_{3\times3}$。请编程实现

```cpp
//导入安装好的Eigen库
#include <iostream>
#include "Eigen/Core"

using namespace std;

int main() {
    const int N = 10;
    Eigen::MatrixXf mat(N, N); //eigen类外定义矩阵，调用mat
    Eigen::MatrixXf ide = Eigen::Matrix<float, 3, 3>::Identity();
    mat.block<3, 3>(0, 0) = ide;    // 这句是关键，通过
    cout << mat << endl;
}
```

6. 一般解线性方程$Ax=b$有哪几种做法？ 你能在Eigen中实现吗？

① 求逆法：$Ax=b \to x=A^{-1}b$

```cpp
// 1. inverse method
MatrixXd method1(const MatrixXd &A, const MatrixXd &b) {
    return A.inverse() * b;
}
```

> 必须是方阵，A满秩

② QR分解法：适合非方阵和方阵 当方程组有解时的出的是真解，若方程组无解得出的是近似解

```cpp
// 2. QR
MatrixXd method2(const MatrixXd &A, const MatrixXd &b) {
    return A.colPivHouseholderQr().solve(b);
}
```

③ 最小二乘法：适合非方阵和方阵，方程组有解时得出真解，否则是最小二乘解(在求解过程中可以用QR分解 分解最小二成的系数矩阵)

```cpp
// 3. least-square
MatrixXd method3(const MatrixXd &A, const MatrixXd &b) {
    return (A.transpose() * A).inverse() * (A.transpose() * b);
}
```

④ LU分解：只能为方阵（满足分解的条件才行）

```cpp
// 4. LU
MatrixXd method4(const MatrixXd &A, const MatrixXd &b) {
    return A.lu().solve(b);
}
```

⑤ Cholesky分解：只能为方阵 (结果与其他的方法差好多)

```cpp
// 5. Cholesky
MatrixXd method5(const MatrixXd &A, const MatrixXd &b) {
    return A.llt().solve(b);
}
```

第四单元

Sophus安装

1. 从github上面下载

```
git clone https://github.com/strasdat/Sophus.git
```

2. 安装

```
cd Sophus
git checkout a621ff  # 切换到非模板分支
mkdir build
cd build
cmake ..
make
```

3. 使用

```
cmake_minimum_required( VERSION 2.8 )
project( useSophus )

# 为使用 sophus，您需要使用find_package命令找到它
find_package( Sophus REQUIRED )
include_directories( ${Sophus_INCLUDE_DIRS} )

add_executable( useSophus useSophus.cpp )
target_link_libraries( useSophus ${Sophus_LIBRARIES} )
```

验证$SO(3), SE(3)$和$Sim(3)$关于乘法成群

对于集合和运算，群$G=(A,\cdot)$满足下述四个条件：

1. 封闭性：$\forall a_1,a_2\in A\ a_1\cdot a_2\in A$
2. 结合律：$\forall a_1,a_2,a_3\in A$，$(a_1\cdot a_2)\cdot a_3=a_1\cdot(a_2\cdot a_3)$
3. 幺元：\exist a_0\in A，$s.t.\forall a\in A$，$a_0\cdot a=a\cdot a_0=a$
4. 逆：$\forall a\in A$，\exist a^{-1}\in A，$s.t.\ a\cdot a^{-1}=a_0$

① 验证$G=(SO(3),\cdot)$

1. 封闭性：$(R_1R_2)(R_1R_2)^T=R_1R_2R_2^TR_1^T=I$，$det(R_1R_2)=det(R_1)det(R_2)=1$。所以$R_1R_2\in SO(3)$
2. 结合律：矩阵乘法的性质
3. 幺元：单位矩阵 $I=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$
4. 逆：R的逆就是R^T

② 验证$G(SE(3),\cdot)$

1. 封闭性：

$$\begin{bmatrix}R_1&t_1\\0^T&1\end{bmatrix}\begin{bmatrix}R_2&t_2\\0^T&1\end{bmatrix}=\begin{bmatrix}R_1R_2&R_1t_2+t_1\\0^T&1\end{bmatrix}$$

   显然$R_1R_2\in SO(3), R_1t_2+t_1\in R^3$

2. 结合律：矩阵乘法的性质
3. 幺元：单位矩阵 $I=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}$
4. 逆：等于证明T存在逆，其实非常容易知道每列线性独立，所以存在逆（因为R存在逆，所以前面3列线性独立，又因为最后一列无法由前面三列线性表示，所以4列均线性独立）

③ 验证$G(Sim(3),\cdot)$

1. 封闭性：

$$\begin{bmatrix} sR_1 & t_1 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} sR_2 & t_2 \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} s^2 R_1 R_2 & sR_1 t_2 + t_1 \\ 0^T & 1 \end{bmatrix}$$

   显然$R_1 R_2 \in SO(3), sR_1 t_2 + t_1 \in R^3$

2. 结合律：矩阵乘法的性质
3. 幺元：单位矩阵 $I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
4. 逆：等于证明T存在逆，其实非常容易知道每列线性独立，所以存在逆（因为R存在逆，所以前面3列线性独立，又因为最后一列无法由前面三列线性表示，所以4列均线性独立）

验证$(R^3, R, \times)$构成李代数

李代数由一个集合V，一个数域F和一个二元运算符$[,]$组成。如果它们满足以下几条性质，则称$(V, F, [,])$为一个李代数，记做$\mathfrak{g}$：

1. 封闭性：$\forall X, Y \in V, [X, Y] \in V$
2. 双线性：$\forall X, Y, Z \in V$，$a, b \in F$，有：

$$[aX + bY, Z] = a[X, Z] + b[Y, Z], \quad [Z, aX + bY] = a[Z, X] + b[Z, Y]$$

3. 自反性：$\forall X \in V, [X, X] = 0$
4. 雅克比等价：$\forall X, Y, Z \in V, [X, [Y, Z]] + [Z, [X, Y]] + [Y, [Z, X]] = 0$

下面令$V=R^3, F=R, [,]=\times $：（请先了解叉积的性质）

1. 封闭性很明显，因为叉积与另两个向量均垂直，但维数还是三维
2. 双线性，叉积本身性质
3. 自反性：叉积本身性质
4. 雅克比等价：$D = X \times (Y \times Z) + Z \times (X \times Y) + Y \times (Z \times X)$，这条式子的含义是什么呢？其实非常简单，就是$D$落在$Y, Z$构成的平面，也落在$X, Y$构成的平面，也落在$Z, X$够成的平面。而对于任意的$X, Y, Z$它们两两构成的平面的交集也就只有0这个向量了。（可以参考：叉乘不满足结合律！！！）

验证$\mathfrak{so}(3)$和$\mathfrak{se}(3)$满足李代数要求的性质

① 验证$\mathfrak{so}(3)$

令 $V = R^3, F = R, [\phi_1, \phi_2] = (\Phi_1\Phi_2 - \Phi_2\Phi_1)^\vee$ ：

1. 封闭性：（假设$\phi_1 = [a_1, a_2, a_3]$，$\phi_2 = [b_1, b_2, b_3]$）

   $(\Phi_1\Phi_2 - \Phi_2\Phi_1) = [-a_3b_3 - a_2b_2 \quad a_2b_1 \quad a_3b_1a_1b_2 \quad -a_3b_3 - a_1b_1 \quad a_3b_2a_1b_3 \quad a_2b_3 \quad -a_2b_2 - a_1b_1] - [-a_3b_3 - a_2b_2 \quad a_1b_2 \quad a_1b_3a_2b_1 \quad -a_3b_3 - a_1b_1$

   也为反对称矩阵，因此 $(\Phi_1\Phi_2 - \Phi_2\Phi_1)^\vee \in R^3$

2. 双线性：

   $[a\phi_1 + b\phi_2, \phi_3]^\wedge = (a\Phi_1 + b\Phi_2)\Phi_3 - \Phi_3(a\Phi_1 + b\Phi_2) = a(\Phi_1\Phi_3 - \Phi_3\Phi_1) + b(\Phi_2\Phi_3 - \Phi_3\Phi_2) = a[\phi_1, \phi_3]^\wedge + b[\phi_2, \phi_3]^\wedge [\phi_3, a\phi_1 + b\phi_2]^\wedge = \Phi_3(a\Phi_1 +$

   所以得证

3. 雅克比等价：
   首先我们知道：$[\phi_1, [\phi_2, \phi_3]] = (\Phi_1(\Phi_2\Phi_3 - \Phi_3\Phi_2) - (\Phi_2\Phi_3 - \Phi_3\Phi_2)\Phi_1)^\vee$
   所以存在下述关系：

   $([\phi_1, [\phi_2, \phi_3]] + [\phi_3, [\phi_1, \phi_2]] + [\phi_2, [\phi_3, \phi_1]])^\wedge = [\phi_1, [\phi_2, \phi_3]]^\wedge + [\phi_3, [\phi_1, \phi_2]]^\wedge + [\phi_2, [\phi_3, \phi_1]]^\wedge = \Phi_1(\Phi_2\Phi_3 - \Phi_3\Phi_2) - (\Phi_2\Phi_3 - \Phi_3\Phi_2)\Phi_1 + \Phi_3(\Phi_1$

   所以得证

② 验证$\mathfrak{se}(3)$

令 $V = R^6, F = R, [\xi_1, \xi_2] = (\xi_1^\wedge \xi_2^\wedge - \xi_2^\wedge \xi_1^\wedge)^\vee$：

1. 封闭性：

   $$(\xi_1^\wedge\ xi_2^\wedge -\ xi_2^\wedge\ xi_1^\wedge) = [\phi_1^\wedge \quad \rho_1 0^T \quad 0]\,[\phi_2^\wedge \quad \rho_2 0^T \quad 0] - [\phi_2^\wedge \quad \rho_2 0^T \quad 0]\,[\phi_1^\wedge \quad \rho_1 0^T \quad 0] = [\phi_1^\wedge \phi_2^\wedge - \phi_2^\wedge \phi_1^\wedge \quad \phi_1^\wedge \rho_2 - \phi_2^\wedge \rho_1 0^T \quad 0]$$

   因为$(\phi_1^\wedge \phi_2^\wedge - \phi_2^\wedge \phi_1^\wedge)^\vee \in so(3)$，所以得证

2. 双线性：（令 $Xi = \xi^\wedge$）

   $$[a\,xi_1 + b\xi_2, \xi_3]^\wedge = (a\,Xi_1 + b\,Xi_2)\,Xi_3 -\ Xi_3(a\,Xi_1 + b\,Xi_2) = a(\ Xi_1\,Xi_3 -\ Xi_3\,Xi_1) + b(\ Xi_2\,Xi_3 -\ Xi_3\,Xi_2) = a[\ xi_1,\ xi_3]^\wedge + b[\ xi_2,\ xi_3]^\wedge[\ xi_3, a\,x$$

   所以得证

3. 雅克比等价：
   首先我们知道：$[\ xi_1, [\ xi_2,\ xi_3]] = (\ Xi_1(\ Xi_2\,Xi_3 -\ Xi_3\,Xi_2) - (\ Xi_2\,Xi_3 -\ Xi_3\,Xi_2)\,Xi_1)^\vee$
   所以存在下述关系：

   $$([\xi_1, [\xi_2, \xi_3]] + [\xi_3, [\xi_1, \xi_2]] + [\xi_2, [\xi_3, \xi_1]])^\wedge = [\xi_1, [\xi_2, \xi_3]]^\wedge + [\xi_3, [\xi_1, \xi_2]]^\wedge + [\xi_2, [\xi_3, \xi_1]]^\wedge =\ Xi_1(\ Xi_2\,Xi_3 -\ Xi_3\,Xi_2) - (\ Xi_2\,Xi_3 -\ Xi_3\,Xi_2)\Xi_1 +\ X$$

   所以得证

4. 验证性质（4.20）和（4.21）

① 性质（4.20）

不妨假设$a = [x, y, z]^T$，且$x^2 + y^2 + z^2 = 1$。那么：

$$aa^T = [xyz]\,[x \quad y \quad z] = [x^2 \quad xy \quad xzxy \quad y^2 \quad yzxz \quad yz \quad z^2]\,a^\wedge a^\wedge = [0 \quad -z \quad yz \quad 0 \quad -x-y \quad x \quad 0]\,[0 \quad -z \quad yz \quad 0 \quad -x-y \quad x \quad 0] = [x^2 - 1 \quad x$$

所以得证。

② 性质（4.21）

$$a^\wedge a^\wedge a^\wedge = [x^2 - 1 \quad xy \quad xzxy \quad y^2 - 1 \quad yzxz \quad yz \quad z^2 - 1]\,[0 \quad -z \quad yz \quad 0 \quad -x-y \quad x \quad 0] = [0 \quad z \quad -y-z \quad 0 \quad xy \quad -x \quad 0] = -a^\wedge$$

所以得证。

证明：$Rp^\wedge R^T = (Rp)^\wedge$

① 最直接的方式就是两边都采用具体的形式展开，比较是不是即可。（写起来很长，就略了。）

② 左边很容易证明为反对称矩阵（所以我们只需证明几个位置即可）：

$$(Rp^\wedge R^T)^T = R(p^\wedge)^T R = -Rp^\wedge R$$

下面我们先给出具体的形式：

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = [R_1 \quad R_2 \quad R_3] \qquad p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

1. 左边：

   $$[R_1 \quad R_2 \quad R_3]\,[0 \quad -z \quad yz \quad 0 \quad -x-y \quad x \quad 0]\begin{bmatrix} R_1^T \\ R_2^T \\ R_3^T \end{bmatrix} = z(R_2R_1^T - R_1R_2^T) + x(R_3R_2^T - R_2R_3^T) + y(R_1R_3^T - R_3R_1^T)$$

2. 右边：

   $$Rp = xR_1 + yR_2 + zR_3$$

   因为我们已经知道左边为反对称矩阵，因此我们只需检查3个位置的值即可（下标为12, 13, 23位置）

3. 下边以$i = 1, j = 2$为例（其他两个位置同样的方式）
   左边矩阵在$i = 1, j = 2$的值：

   $$z(r_{11}r_{22} - r_{21}r_{12}) + x(r_{13}r_{22} - r_{23}r_{12}) + y(r_{11}r_{23} - r_{21}r_{13}) = -zr_{33} - xr_{31} - yr_{32}$$

   > 上述用了正交矩阵的性质：$r_{ij} = det(R_{ij})$

右边矩阵在$i=1, j=2$的值：

$$-zr_{33}-xr_{31}-yr_{32}$$

所以左边=右边，得证。

证明：$Rexp(p^{\wedge})R^T = exp((Rp)^{\wedge})$

> 该式称为$SO(3)$上的伴随性质，同样地，在$SE(3)$上亦有伴随性质：
>
> $$Texp(\xi^{\wedge})T^{-1} = exp((Ad(T)\xi)^{\wedge})$$
>
> 其中：
>
> $$Ad(T) = \begin{bmatrix} R & t^{\wedge}R \\ 0 & R \end{bmatrix}$$

> 参考[Derivation of Adjoint for SO(3)](), [Ethan Eade]()

1. 先对两边同时取log操作：
   左边：

   $$\log(R\exp(p^{\wedge})R^T) = Rp^{\wedge}R^T$$

   > 这条式子个人不知道怎么推出来，[logarithm]()应该需要对角矩阵才行吧？

   右边：

   $$\log(\exp((Rp)^{\wedge})) = (Rp)^{\wedge}$$

2. 那么假设$Rp^{\wedge}R^T = (Rp)^{\wedge}$，它代表的含义其实也是蛮明确的：反旋转+叉积+旋转 = 旋转后再叉积。下面我们不妨将上述操作作用在一个向量v上面：
   右边乘v：

   $$(Rp)^{\wedge}v = (Rp)\times v = Rp\times(RR^Tv) = R[p\times(R^Tv)] = Rp^{\wedge}R^Tv$$

   等于左边。所以得证。

仿照左扰动的推导，推导$SO(3)$和$SE(3)$在右扰动下的导数

① $SO(3)$

$$\frac{\partial(Rp)}{\partial\varphi} = \lim_{\varphi\to 0}\frac{\exp(\phi^{\wedge})\exp(\varphi^{\wedge})p-\exp(\phi^{\wedge})p}{\varphi} \approx \lim_{\varphi\to 0}\frac{\exp(\phi^{\wedge})(I+\varphi^{\wedge})p-\exp(\phi^{\wedge})p}{\varphi} = \lim_{\varphi\to 0}\frac{\exp(\phi^{\wedge})\varphi^{\wedge}p}{\varphi} = \lim_{\varphi\to 0}\frac{R\varphi^{\wedge}p}{\varphi} = \lim_{\varphi\to 0}\frac{-Rp^{\wedge}\varphi}{\varphi} = -Rp^{\wedge}$$

② $SE(3)$

$$\frac{\partial(Tp)}{\partial\delta\xi} = \lim_{\delta\xi\to 0}\frac{\exp(\xi^{\wedge})\exp(\delta\xi^{\wedge})p-\exp(\xi^{\wedge})p}{\delta\xi} \approx \lim_{\delta\xi\to 0}\frac{\exp(\xi^{\wedge})\exp(I+\delta\xi^{\wedge})p-\exp(\xi^{\wedge})p}{\delta\xi} = \lim_{\delta\xi\to 0}\frac{\exp(\xi^{\wedge})\delta\xi^{\wedge}p}{\delta\xi} = \lim_{\delta\xi\to 0}\frac{\begin{bmatrix} R & t \\ 0^T & 1\end{bmatrix}[\delta\phi^{\wedge} \quad \delta\rho 0^T \quad 1]}{\delta\xi}$$

再进一步推导之前，先说一下：① $Tp \in R^{4\times 1}, \delta\xi \in R^{6\times 1}$（因为有6个变量，所以$\delta\xi$才为6维）。所以结果应该为$4\times 6$ ② 下述认为前三维为位移，后三维为旋转（即前面三维与ρ相关，后面三维与ϕ相关）

进一步推导：

$$\frac{\partial(Tp)}{\partial\delta\xi} = \lim_{\delta\xi\to 0}\frac{[R\delta\phi^{\wedge}p+R\delta\rho 0^T]}{\delta\xi} = [-Rp^{\wedge} \quad W0^T \quad 0^T]$$

其中的W为矩阵R按列和的转置。

搜索cmake的find_package指令是如何运作的。它有那些可选的参数？为了让cmake找到某个库，需要哪些先决条件？

FIND_PACKAGE参数

FIND_PACKAGE(<name> [version] [EXACT] [QUIET] [NO_MODULE] [[REQUIRED | COMPONENTS] [componets...]])：用来调用预定义在 CMAKE_MODULE_PATH 下的Find<name>.cmake 模块（也可以自己定义 Find<name> 模块，将其放入工程的某个目录中，通过 SET(CMAKE_MODULE_PATH dir) 设置查找路径，供工程FIND_PACKAGE 使用）

> 这条命令执行后，CMake 会到变量 CMAKE_MODULE_PATH 指示的目录中查找文件Findname.cmake 并执行。

下面来说明下其中各个参数的含义（作用）：

- version 参数：它是正在查找的包应该兼容的版本号（格式是major[.minor[.patch[.tweak]]]）（例如OpenCV采用3.x版本，采用的是 find_package(OpenCV 3 REQUIRED) --- 这里的3就是版本号）
- EXACT选项：要求版本号必须精确匹配。如果在find-module内部对该命令的递归调用没有给定[version]参数，那么[version]和EXACT选项会自动地从外部调用前向继承。对版本的支持目前只存在于包和包之间
- QUIET 参数：会禁掉包没有被发现时的警告信息。对应于Find<name>.cmake 模块中的 NAME_FIND_QUIETLY
- REQUIRED 参数：其含义是指是否是工程必须的，表示如果报没有找到的话，cmake的过程会终止，并输出警告信息。对应于Find<name>.cmake 模块中的NAME_FIND_REQUIRED 变量。
- COMPONENTS参数：在REQUIRED选项之后，或者如果没有指定REQUIRED选项但是指定了COMPONENTS选项，在它们的后面可以列出一些与包相关（依赖）的部件清单（components list）

> 注：比较常用的是version，COMPONENTS和REQUIRED

FIND_PACKAGE如何查找

1. find_package() 命令会在模块路径中寻找 Find<name>.cmake ，这是查找库的一个典型方式。首先CMake 查看${CMAKE_MODULE_PATH} 中的所有目录，然后再查看它自己的模块目录 <CMAKE_ROOT>/share/cmake-x.y/Modules/ 。
2. 如果没找到这样的文件，会寻找 <Name>Config.cmake 或者 <lower-case-name>-config.cmake ，它们是假定库会安装的文件（但是目前还没有多少库会安装它们）。不做检查，直接包含安装的库的固定值。

第1种方式称为模块模式，第2种方式称为配置模式。配置模式的文件的编写见 这里的文档 。可能还会用到 importing and exporting targets 这篇文档。下述主要介绍更常见的第1种方式

不管使用哪一种模式，只要找到包，就会定义下面这些变量（这些我们在CMakeLists.txt里面要用到）：

```
<NAME>_FOUND
<NAME>_INCLUDE_DIRS or <NAME>_INCLUDES
<NAME>_LIBRARIES or <NAME>_LIBRARIES or <NAME>_LIBS
<NAME>_DEFINITIONS
```

这些都在 Find<name>.cmake 文件中。

现在，在你的代码（要使用库 <name> 的代码）的顶层目录中的 CMakeLists.txt 文件中，我们检查变量<NAME>_FOUND 来确定包是否被找到。大部分包的这些变量中的包名是全大写的，如 LIBFOO_FOUND ，有些包则使用包的实际大小写，如 LibFoo_FOUND 。如果找到这个包，我们用<NAME>_INCLUDE_DIRS 调用include_directories() 命令，用 <NAME>_LIBRARIES 调用 target_link_libraries() 命令。

cmake自带查找模块的外部库

为了能支持各种常见的库和包，CMake自带了很多模块。可以通过命令 cmake --help-module-list （输入cmake --help，然后双击Tab会有命令提示）得到你的CMake支持的模块的列表。