

SLAM14

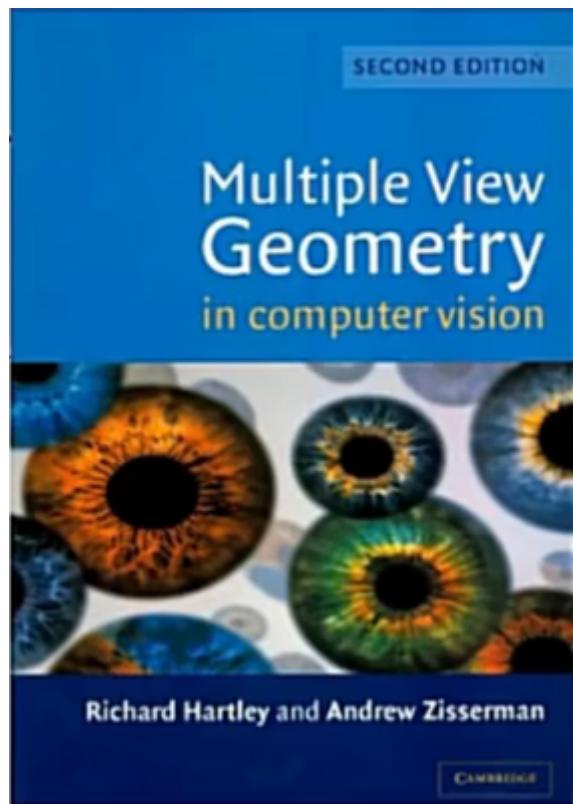
SLAM : Simultaneous Localization and Mapping

同时定位与地图构建

Chapter1

引言

- 困难之处
 - 三维空间的运动
 - 受噪声影响
 - 数据来源只有图像
- 从学习角度来看
 - 牵涉到理论太广
 - 从理论到实现困难
 - 资料缺乏
- 人类看到的是图像，计算机看到的是数值矩阵
- Make computer see?



- 本书的特点

- 基础、必要的理论知识
 - 大量的编程内容
 - 重视工程实践
-
- 观念：只有亲自动手实现了算法，才能谈得上理解
 - 旅行愉快！

- 内容

- 第一部分：数学基础
 - 概述 Linux
 - 三维空间刚体运动表述 Eigen
 - 李群与李代数 Sophus
 - 相机模型与图像 OpenCV
 - 非线性优化 Ceres, g2o
- 第二部分：视觉SLAM
 - 视觉里程计 OpenCV
 - 后端优化 Ceres, g2o, gtsam

- 第二部分：视觉SLAM

- 回环检测 DBoW3
 - 地图构建 PCL, Octomap
- 其他
 - 历史回顾和未来

代码：<https://github.com/gaoxiang12/slambook>



- 预备知识

- 数学：高等数学、线性代数（矩阵论）、概率论
- 编程：C++、Linux，了解语法和基本命令即可
- 不提供windows环境下的方案！
- 阅读第一章习题作为自测

- 例如：

- 类是什么？STL是什么？
- 你是否用过Ubuntu？如何在Ubuntu中安装某个软件？

第一部分：数学基础

第二部分:视觉SLAM

第三部分:其他

Chapter2

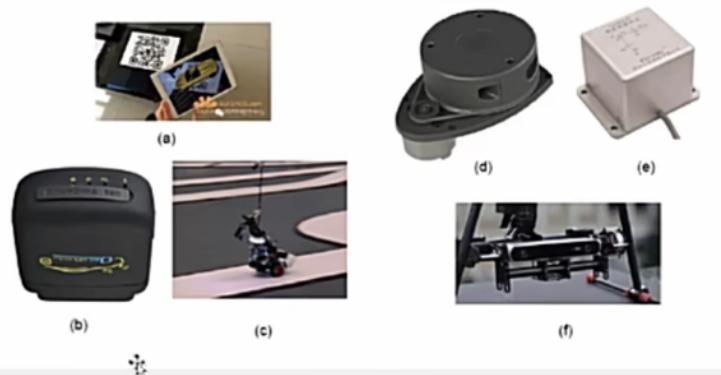
内外兼修

- Question 自主运动两大基本问题

- 我在什么地方? ——定位
- 周围长什么样子? ——建图
- 机器人的“内外兼修”: 定位侧重对自身的了解, 建图侧重对外在的了解
- 相互耦合的两个问题
- 准确的定位需要精确的地图
- 精确的地图来自准确的定位

传感器的配置

- How to do SLAM? — Sensors



两类传感器

- 安装于环境中的:
 - 二维码 Marker
 - GPS
 - 导轨、磁条
- 携带于机器人本体上的:
 - IMU
 - 激光
 - 相机

环境中配置传感器

- 环境中的传感器限制了应用环境
 - 需要环境允许使用GPS、允许贴marker
 - 而SLAM强调未知环境
 - 更重视携带式传感器

1. Def. of SLAM

Sensors

The methods and difficulty of SLAM depend heavily on the equipped sensors.

- Lasers
 - Accurate
 - Fast
 - Long history in research
 - Heavy
 - Expensive
- Examples: SICK, Velodyne, Rplidar

- Cameras
 - Cheap
 - Light-weight
 - Rich information
 - High computation cost
 - Work under assumptions
- Categories: monocular, stereo, RGBD



相机分类

- 相机
 - 以一定速率采集图像，形成视频
- 分类
 - 单目 Monocular
 - 双目 Stereo
 - 深度 RGBD
 - 其他 鱼眼 全景 Event Camera, etc.

相纸之间的差距

- 相机的本质
 - 以二维投影形式记录了三维世界的信息
 - 此过程丢掉了一个维度：距离
- 各类相机主要区别：有没有深度信息
 - 单目：没有深度，必须通过移动相机产生深度 Moving View Stereo
 - 双目：通过视差计算深度 Stereo
 - RGBD：通过物理方法测量深度

单目相机的缺点



图 2-4 单目视觉中的尴尬：不知道深度时，手掌上的人是真人还是模型？

仅有一个图像时：

- 可能是很近但很小的物体
- 可能是很远但很大的物体

它们成像相同

必须在移动相机后才能得知相机的运动和场景的结构

单目相机

- 当相机运动起来时
 - 场景和成像有几何关系
 - 近处物体的像运动快
 - 远处物体的像运动慢
 - 可以推断距离



双面相机

- 双目相机：左右眼的微小差异判断远近
- 同样，远处物体变化小，近处物体变化大——推算距离 计算量非常大



图 2-5 双目相机的数据：左眼图像，右眼图像。通过左右眼的差异，能够判断场景中物体离相机距离。

深度相机

- 深度相机
 - 物理手段测量深度
 - 结构光 ToF
 - 主动测量，功耗大
 - 深度值较准确
 - 量程较小，易受干扰

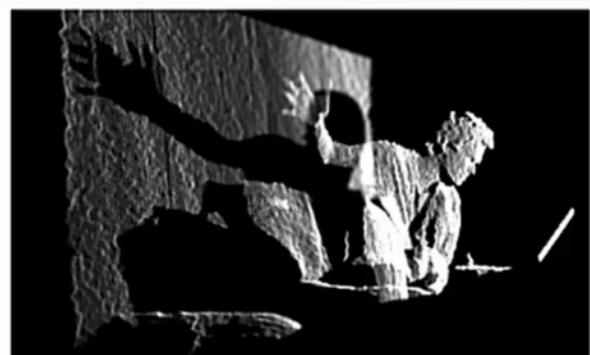


图 2-6 RGBD 数据：深度相机可以直接测量物体的图像和距离，从而恢复三维结构。

共同点

- 共同点
 - 利用图像和场景的几何关系，计算相机运动和场景结构 Motion & Structure
 - 三维空间的运动和结构
 - 图像来自连续的视频

视觉SLAM的做法

- 视觉SLAM框架
- 前端：VO
- 后端：Optimization
- 回环检测 Loop Closing
- 建图 Mapping



图 2-7 整体视觉 SLAM 流程图。

- 视觉里程计 Visual Odometry
 - 相邻图像估计相机运动
 - 基本形式：通过两张图像计算运动和结构
 - 不可避免地有漂移
- 方法
 - 特征点法 第七讲
 - 直接法 第八讲



图 2-8 相机拍摄到的图片与人眼反应的运动方向。

新图像进来
↓
前端（视觉里程计） → 快速算出当前位姿 + 新3D点
↓
如果这帧是关键帧 → 交给后端局部BA优化
↓
同时把这帧放进“词袋数据库”
↓
回环检测模块不断检索：有没有历史帧和当前帧很像？
↓ 是 → 加上回环约束 → 触发全局优化（后端）
↓ 否 → 继续往前走
↓
最终所有优化好的关键帧位姿 + 地图点 → 输出给建图模块

• 后端优化

- 从带有噪声的数据中优化轨迹和地图 状态估计问题
- 最大后验概率估计 MAP
- 前期以EKF为代表，现在以图优化为代表 第十、十一讲

- 回环检测
 - 检测机器人是否回到早先位置
 - 识别到达过的场景
 - 计算图像间的相似性
- 方法：词袋模型 第十二讲

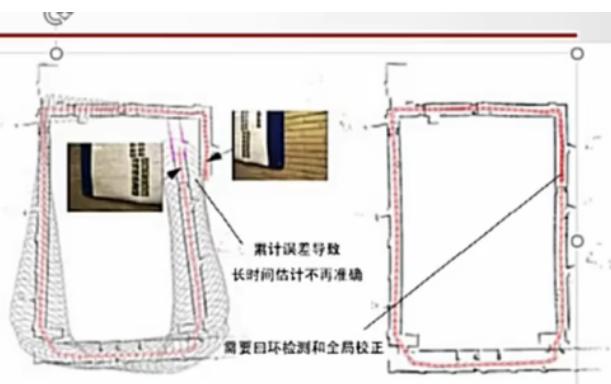


图 2-9 累计误差与回环检测的校正结果 [10]。

- 建图 第十三讲
 - 用于导航、规划、通讯、可视化、交互等
 - 度量地图 vs 拓扑地图
 - 稀疏地图 vs 稠密地图

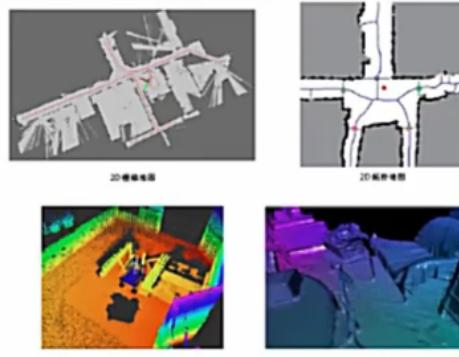


图 2-10 形形色色的地图：2D 棋格地图、拓扑地图以及 3D 点云地图和网格地图 [12]。

SLAM数学问题

- SLAM问题的数学描述
 - 离散时间: $t = 1, 2, \dots, k$
 - 小萝卜的位置: x_1, x_2, \dots, x_k
 - 小萝卜是从上一个时刻运动到下一个时刻的
 - 运动方程:

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

上一时刻
输入
噪声

• SLAM问题的数学描述

- 路标（三维空间点）： y_1, y_2, \dots, y_n
- 传感器在位置 x_k 处，探测到了路标 y_j
- 观测方程:

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

- 两个基本方程:

运动方程	$\begin{cases} \dot{x}_k = f(x_{k-1}, u_k, w_k) \\ z_{k,j} = h(y_j, x_k, v_{k,j}) \end{cases}$
观测量方程	

- Question:

- 位置是三维的, 如何表述? ——第三、四讲
- 观测是相机中的像素点, 如何表述? ——第五讲
- 已知 u, z 时, 如何推断 x, y ? ——第六讲

cmake

```
## 1. 基本结构模板

```cmake
cmake_minimum_required(VERSION 3.10)
project(MyProject)
set(CMAKE_CXX_STANDARD 17)

set(SRC main.cpp src/a.cpp src/b.cpp)
add_executable(my_app ${SRC})
target_link_libraries(my_app PRIVATE pthread)
```

---

## 2. 添加头文件路径

```cmake
include_directories(include)
或现代写法
target_include_directories(my_app PUBLIC include)
```

---

## 3. 添加源文件

```cmake
aux_source_directory(src SRC_LIST)
add_executable(my_app ${SRC_LIST})
```

---

## 4. 链接库

```cmake
• 系统库

```

```

```cmake
target_link_libraries(my_app PRIVATE pthread dl m)
```

⚡ 自己的 .a / .so

```cmake
link_directories(${PROJECT_SOURCE_DIR}/lib)
target_link_libraries(my_app PRIVATE mylib)
```

5. find_package 查找外部库

OpenCV

```cmake
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
target_link_libraries(my_app PRIVATE ${OpenCV_LIBS})
```

Eigen

```cmake
find_package(Eigen3 REQUIRED)
target_link_libraries(my_app PRIVATE Eigen3::Eigen)
```

Pangolin

```cmake
find_package(Pangolin REQUIRED)
target_link_libraries(my_app PRIVATE ${Pangolin_LIBRARIES})
```

6. 编译选项

```cmake
add_compile_options(-Wall -Wextra -O2)
# 或
target_compile_options(my_app PRIVATE -O3 -march=native)
```

7. Debug / Release

```bash
cmake .. -DCMAKE_BUILD_TYPE=Release
cmake .. -DCMAKE_BUILD_TYPE=Debug
```

```

## ## 8. 控制输出路径

```
```cmake
set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/bin)
set(CMAKE_LIBRARY_OUTPUT_DIRECTORY ${PROJECT_BINARY_DIR}/lib)
```
```

---

## ## 9. 定义宏

```
```cmake
add_definitions(-DDEBUG)
# 或
target_compile_definitions(my_app PRIVATE DEBUG)
```
```

---

## ## 10. 添加子目录（多模块工程常用）

```
```cmake
add_subdirectory(src/moduleA)
add_subdirectory(src/moduleB)
```
```

---

## ## 11. 构建静态库/动态库

```
```cmake
add_library(my_lib STATIC src/a.cpp src/b.cpp)
# or
# add_library(my_lib SHARED src/a.cpp src/b.cpp)

target_include_directories(my_lib PUBLIC include)
```
```

---

## ## 12. configure\_file (变量生成配置文件)

```
```cmake
configure_file(config.h.in config.h)
```
```

config.h.in 内容示例:

```
```c
#define VERSION_MAJOR @PROJECT_VERSION_MAJOR@
```
```

---

## ## 13. install 部署命令

```
```cmake
install(TARGETS my_app RUNTIME DESTINATION bin)
```
```

```
install(FILES config.yaml DESTINATION share)
install(DIRECTORY include/ DESTINATION include)
```
---
```

14. 最常用工作流

```
```bash
mkdir build
cd build
cmake ..
make -j8
```
---
```

带库路径:

```
```bash
cmake .. -DOpenCV_DIR=/usr/local/lib/cmake/opencv4
```
---
```

★ **CMake 速查表 (1 分钟记住) **

| 功能 | 命令 |
|--------|---|
| 设置项目 | `project()` |
| 最低版本 | `cmake_minimum_required()` |
| C++ 标准 | `set(CMAKE_CXX_STANDARD 17)` |
| 可执行文件 | `add_executable()` |
| 库文件 | `add_library()` |
| 链接库 | `target_link_libraries()` |
| 头文件目录 | `target_include_directories()` |
| 查找库 | `find_package()` |
| 子目录 | `add_subdirectory()` |
| 编译选项 | `target_compile_options()` |
| 宏定义 | `target_compile_definitions()` |
| 输出路径 | `set(CMAKE_RUNTIME_OUTPUT_DIRECTORY ...)` |
| 安装部署 | `install()` |

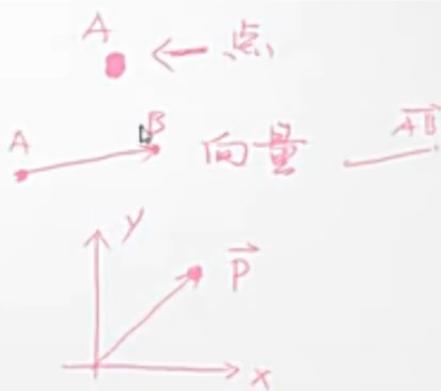
Chapter3

3.1点，向量和坐标，旋转矩阵。

点

- 点存在于三维空间之中
- 点和点可以组成向量
- 点本身由原点指向它的向量所描述

- 向量
 - 带指向性的箭头
 - 可以进行加法、减法等运算

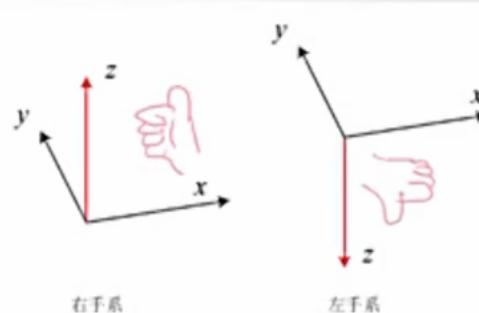


坐标系

- 定义坐标系后，向量可以由 \mathbb{R}^3 坐标表示

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

- 坐标系：由三个正交的轴组成
 - 构成线性空间的一组基
 - 左手系和右手系



向量的运算

- 向量的运算可以由坐标运算来表达

- 加减法

$$\mathbf{a} \pm \mathbf{b} = \sum_i (a_i \pm b_i)$$

- 内积

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

- 外积

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{bmatrix} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \mathbf{b} \triangleq \mathbf{a} \wedge \mathbf{b}.$$

这个定义之后还要用

反对称矩阵的

外积是面积

坐标的变换

世界坐标系

移动的机器人

- 目前为止都很平凡

- 但是有一个基本问题：

坐标系之间是如何变化的？

- 进而：

如何计算同一个向量在不同坐标系里的坐标？

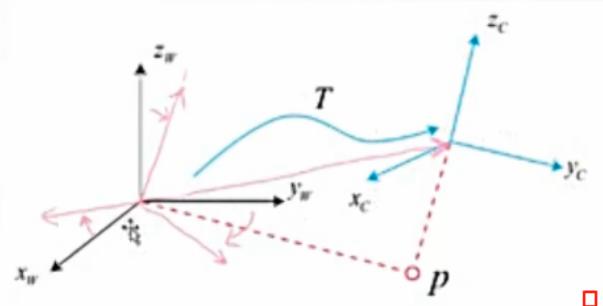
- 在SLAM中：

- 固定的**世界坐标系**和**移动的机器人坐标系**
- 机器人坐标系随着机器人运动而改变，每个时刻都有新的坐标系

平移+旋转

平移是一个向量，旋转是一个矩阵的

- 两个不同的坐标系
- 如何描述左侧到右侧的变化？
- 直观看来由两个部分组成：
 - 原点间的**平移**
 - 三个轴的**旋转**
- 平移是一个**向量**
- 旋转是什么？



$T = \text{旋转} + \text{平移}$

旋转

- 旋转
- 设某坐标系 (e_1, e_2, e_3) 发生了一次旋转，变成了 (e'_1, e'_2, e'_3)
- 对于某个固定的向量 a (向量不随坐标系旋转)，它的坐标怎么变化？
- 坐标关系：

$$[e_1, e_2, e_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = [e'_1, e'_2, e'_3] \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix}.$$

标准正交基

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

• 左乘 $\begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix}$, 得:

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

旋转矩阵

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = R \begin{bmatrix} a'_1 \\ a'_2 \\ a'_3 \end{bmatrix} \quad \text{或} \quad \mathbf{a} = R\mathbf{a}'$$

- 中间的矩阵 R 称为旋转矩阵
 - 根据定义可以验证:
 - R 是一个正交矩阵;
 - R 的行列式为 +1。
 - 满足这两个性质的矩阵称为旋转矩阵
- $SO(n) = \{R \in \mathbb{R}^{n \times n} | RR^T = I, \det(R) = 1\}.$
- 旋转矩阵描述了两个坐标的变换关系
 - 比如: $a_1 = R_{12}a_2$
 - 反之: $a_2 = R_{21}a_1$
 - 于是:

$$R_{21} = R_{12}^{-1} = R_{12}^T$$
 - 进一步, 三个坐标系亦有:

$$a_3 = R_{32}a_2 = R_{32}R_{21}a_1 = R_{31}a_1$$
- Special Orthogonal Group 特殊正交群

旋转矩阵的性质

- 加上平移:
$$\mathbf{a}' = R\mathbf{a} + \mathbf{t}.$$
- 两个坐标系的刚体运动可以由 R, t 完全描述。

- 齐次坐标与变换矩阵
- 用旋转+平移方式有一点不便之处，比如发生了两次变换：

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

- 这时：

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

- 叠加起来过于复杂

- 改变形式，写成：

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

• 这种用四个数表达三维向量的做法称为齐次坐标

• 引入齐次坐标后，旋转和平移可以放入同一个矩阵，称为变换矩阵

- 记 $\tilde{\mathbf{a}} = \begin{bmatrix} \mathbf{a} \\ 1 \end{bmatrix}$

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

- 那么多次变换就可写成：

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

• 称为特殊欧氏群 (Special Euclidean Group)

R==3*3 T==3*1

- 类似的，可定义反向的变换：

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



- 例子

• 在SLAM中，通常定义世界坐标系 T_W 与机器人坐标系 T_R

• 一个点的世界坐标为 p_W ，机器人坐标系下为 p_R ，那么满足关系：

$$p_R = T_{RW} p_W$$

- 反之亦然

• 在实际编程中，可使用 T_{RW} 或 T_{WR} 来描述机器人的位姿。

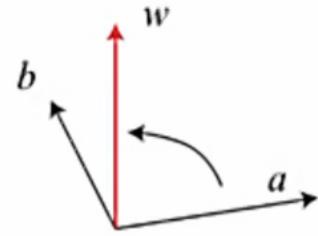
$P_R = R_{RW} * P_W$

3.2 实践部分

3.3 旋转向量，欧拉角

角轴

- 除了旋转矩阵之外的旋转表示
 - 三维旋转：三自由度，用 R^3 向量表示
 - 方向为旋转轴、长度为转过的角度
 - 称为角轴 (Angle-Axis) 或旋转向量 (Rotation Vector)
- $$w = \theta n$$



旋转表示

角轴的特点

- 角轴与旋转矩阵的不同
 - 旋转矩阵：九个量，有正交性约束和行列式值约束
 - 角轴：三个量，没有约束
- 注意它们只是表达方式的不同，但表达的东西可以是同一个
- 角轴也就是第四章要介绍的李代数

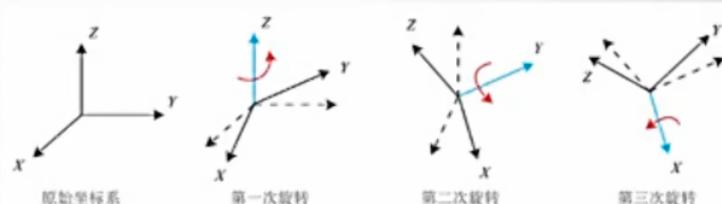
- 转换关系
 - 轴角转旋转矩阵：罗德里格斯公式 (Rodrigues's Formula)

$$R = \cos \theta I + (1 - \cos \theta) nn^T + \sin \theta n^\wedge.$$
 - 旋转矩阵转轴角
 - 角度： $\theta = \arccos(\frac{\text{tr}(R) - 1}{2}).$
 - 轴： $Rn = n.$

角轴和旋转矩阵的转换

欧拉角

- 欧拉角 (Euler Angles)
 - 将旋转分解为三次不同轴上的转动，以便理解
 - 例如：按 Z-Y-X 顺序转动
 - 轴可以是定轴或动轴，顺序亦可不同，因此存在许多种定义方式不同的欧拉角
 - 常见的有 yaw-pitch-roll (偏航-俯仰-滚转) 角等等。



1. 绕物体的Z轴旋转，得到偏航角yaw;
2. 绕旋转之后的Y轴旋转，得到俯仰角pitch;
3. 绕旋转之后的X轴旋转，得到滚转角roll。

分解旋转

z yaw, y pitch, w roll

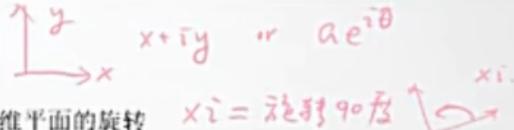
万向锁的问题：丢失一个自由度的

- 由于万向锁，欧拉角不适用于插值和迭代，往往只用于人机交互中。
- 可以证明，用三个实数来表达三维旋转时，会不可避免地碰到奇异性问题。
- SLAM 程序中很少直接使用欧拉角表达姿态

3.4 四元数

- 四元数 (Quaternion)

- 一种扩展的复数
- 回忆：（单位圆上的）复数可以表达二维平面的旋转



- 四元数有三个虚部，可以表达三维空间中的旋转：

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

- 虚部之间的关系：

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

自己和自己的运算像复数
自己和别人的运算像叉乘



四元数的运算

- 和复数一样，**单位四元数**可以表达三维空间的一次旋转
- 四元数的一些运算和性质：

加减法 $q_a \pm q_b = [q_a \pm q_b, \mathbf{v}_a \pm \mathbf{v}_b].$

共轭 $q_a^* = s_a - x_a i - y_a j - z_a k = [s_a, -\mathbf{v}_a].$

乘法

$$\begin{aligned} q_a q_b &= s_a s_b - x_a x_b - y_a y_b - z_a z_b \\ &\quad + (s_a x_b + x_a s_b + y_a z_b - z_a y_b) i \\ &\quad + (s_a y_b - x_a z_b + y_a s_b + z_a x_b) j \\ &\quad + (s_a z_b + x_a y_b - y_a x_b + z_a s_b) k. \end{aligned}$$

模长 $\|q_a\| = \sqrt{s_a^2 + x_a^2 + y_a^2 + z_a^2}.$

逆

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

数乘

$$kq = [ks, k\mathbf{v}].$$

乘法

$$q_a q_b = [s_a s_b - \mathbf{v}_a^T \mathbf{v}_b, s_a \mathbf{v}_b + s_b \mathbf{v}_a + \mathbf{v}_a \times \mathbf{v}_b].$$

点乘

$$q_a \cdot q_b = s_a s_b + x_a x_b i + y_a y_b j + z_a z_b k.$$

四元素的旋转

- 四元数和角轴的关系

- 角轴到四元数:

$$q = \left[\cos \frac{\theta}{2}, n_x \sin \frac{\theta}{2}, n_y \sin \frac{\theta}{2}, n_z \sin \frac{\theta}{2} \right]^T.$$

- 四元数到角轴:

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

- 类似可知四元数亦可转换为旋转矩阵、欧拉角

四元数的优点:

- 如何用四元数旋转一个空间点?
- 设点 p 经过一次以 q 表示的旋转后, 得到了 p' , 它们关系如何表示?
 - 将 p 的坐标用四元数表示(虚四元数): $p = [0, x, y, z] = [0, \mathbf{v}]$.
 - 旋转之后的关系为:

$$p' = qpq^{-1}.$$

可以验证这货也是虚四元数

- 四元数相比于角轴、欧拉角的优势: 紧凑、无奇异性

