



北京航空航天大学  
BEIHANG UNIVERSITY

# 机器人导航python实践(入门)

## Lecture4-同步定位与建图 part1

---

北航 国新院 实验实践课  
智能系统与人形机器人国际研究中心



教师： 欧 阳 老 师



邮 箱： ouyangkid@buaa.edu.cn



学 期： 2025年秋季

# 目录

## Contents

01 课程内容安排

02 ICP

03 EKF SLAM

04 FastSLAM 1

05 FastSLAM 2

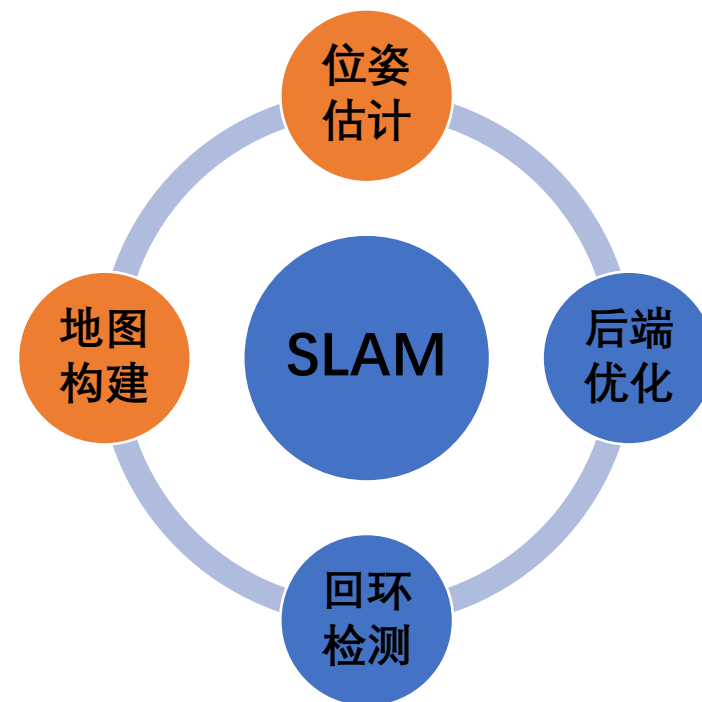
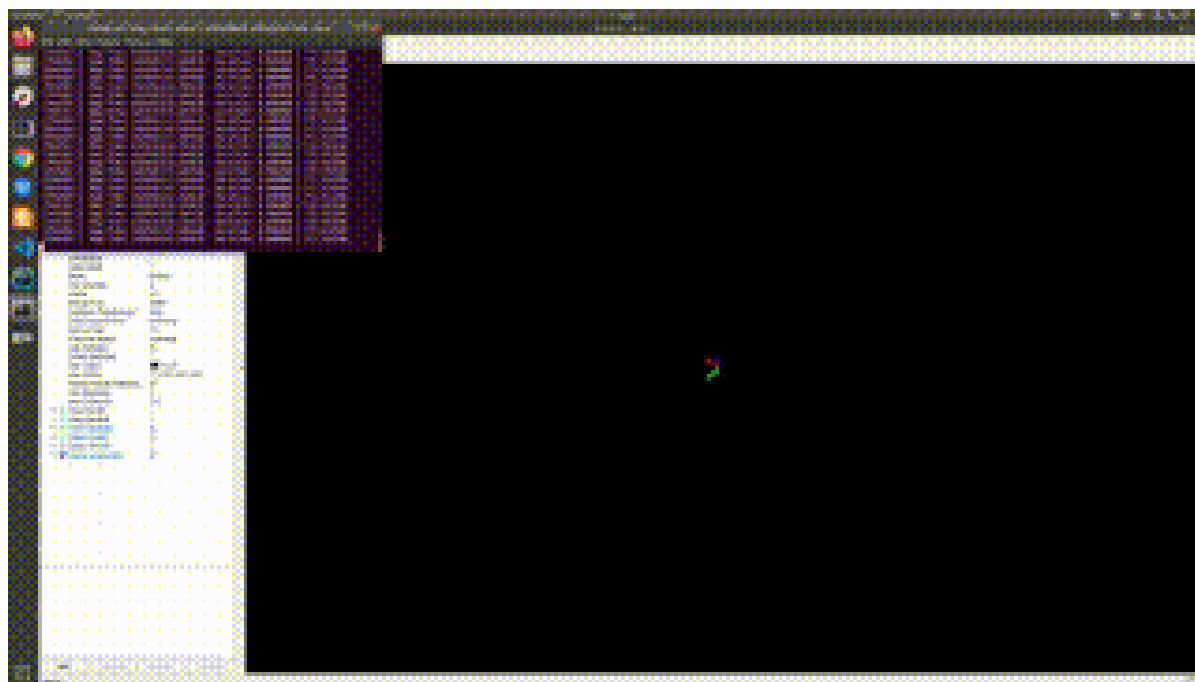
06 Graph-based SLAM

# Part 1 | 课程内容安排

## » 内容安排

**同步定位与建图**，故名思意包含了定位与建图两个子模块，实际上就是模拟人的空间认知，将连续扫描得到的环境信息进行时空融合。机器人在运动过程中，即基于相邻的观测信息估计了自身位姿（pose=位置+姿态）的变化，又基于相邻位姿将观测中的稳定特征按时序进行叠加，构建了对环境的地图。

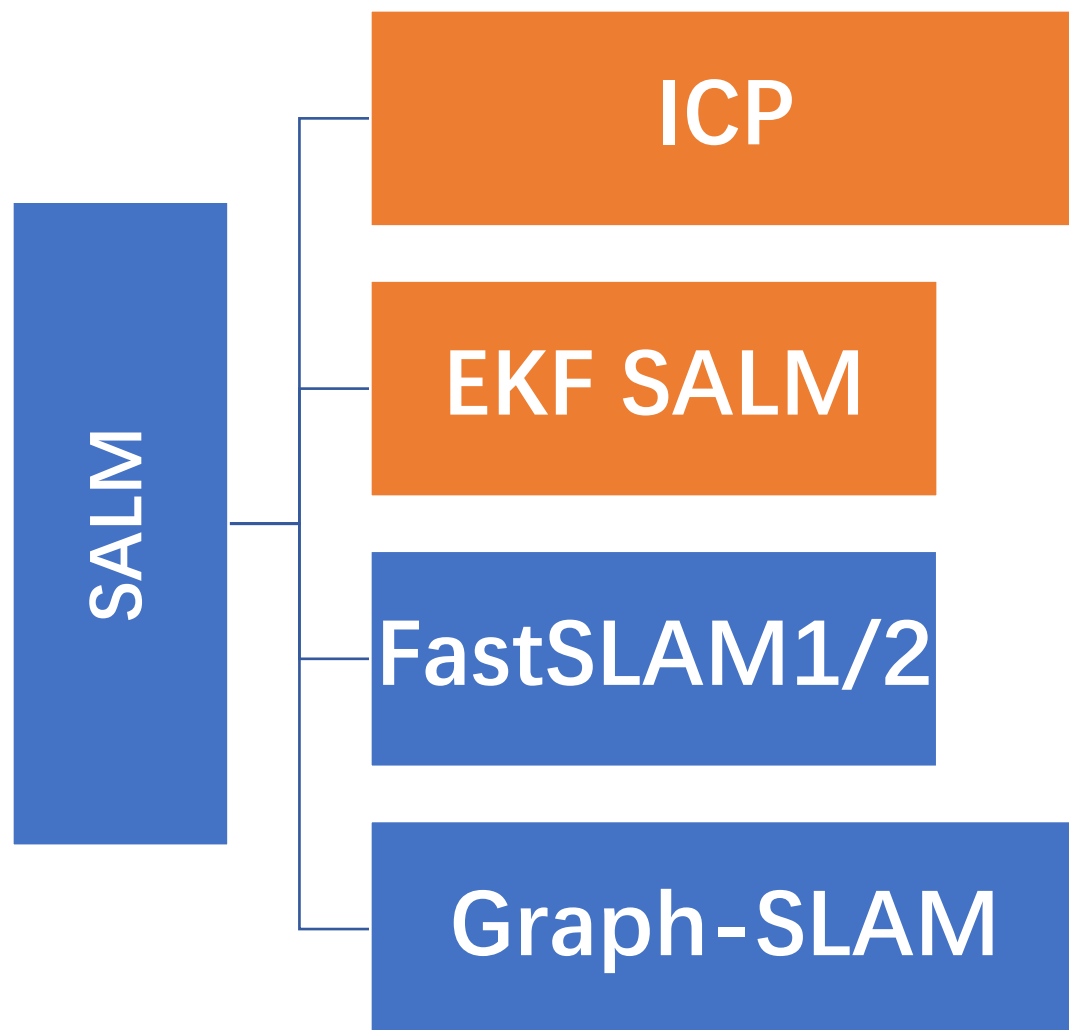
整个过程可以类比人在新环境中，通过逐渐探索记住环境中的关键标识特征，形成地图的过程。



# Part 1 | 课程内容安排

课程内容

» 内容安排



# 目录

## Contents

01 课程内容安排

02 ICP

03 EKF SLAM

04 FastSLAM 1

05 FastSLAM 2

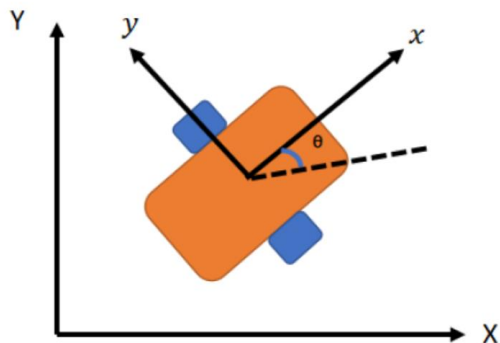
06 Graph-based SLAM

# Part 2 | SLAM

## SLAM-ICP: Iterative Closest Point Matching

迭代最近点匹配算法，主要涉及帧间配准。以2D情况为主，介绍相关算法。

刚体2D坐标系下的运动—位姿变换。



对任意时刻下，刚体的位姿 (Pose) 由：

- 位置：(x,y)，表示刚体参考点（例如机器人中心）在全局坐标系中的平移；
- 朝向： $\theta$ ，表示刚体相对于全局坐标系的旋转角度（通常逆时针为正）。

$$\mathbf{p} = (x, y, \theta)$$

为了方便计算，一般通过构建齐次坐标下的 位姿矩阵：

$$\mathbf{T} = \begin{bmatrix} \cos \theta & -\sin \theta & x \\ \sin \theta & \cos \theta & y \\ 0 & 0 & 1 \end{bmatrix}$$

旋转分量R

平移分量t

可逆：  $\mathbf{T}^{-1} = \begin{bmatrix} \cos \theta & \sin \theta & -x \cos \theta - y \sin \theta \\ -\sin \theta & \cos \theta & x \sin \theta - y \cos \theta \\ 0 & 0 & 1 \end{bmatrix}$

级联：  $\mathbf{T}_{k+1} = \mathbf{T}_k \mathbf{T}_{k+1,k}$

# Part 2 | SLAM

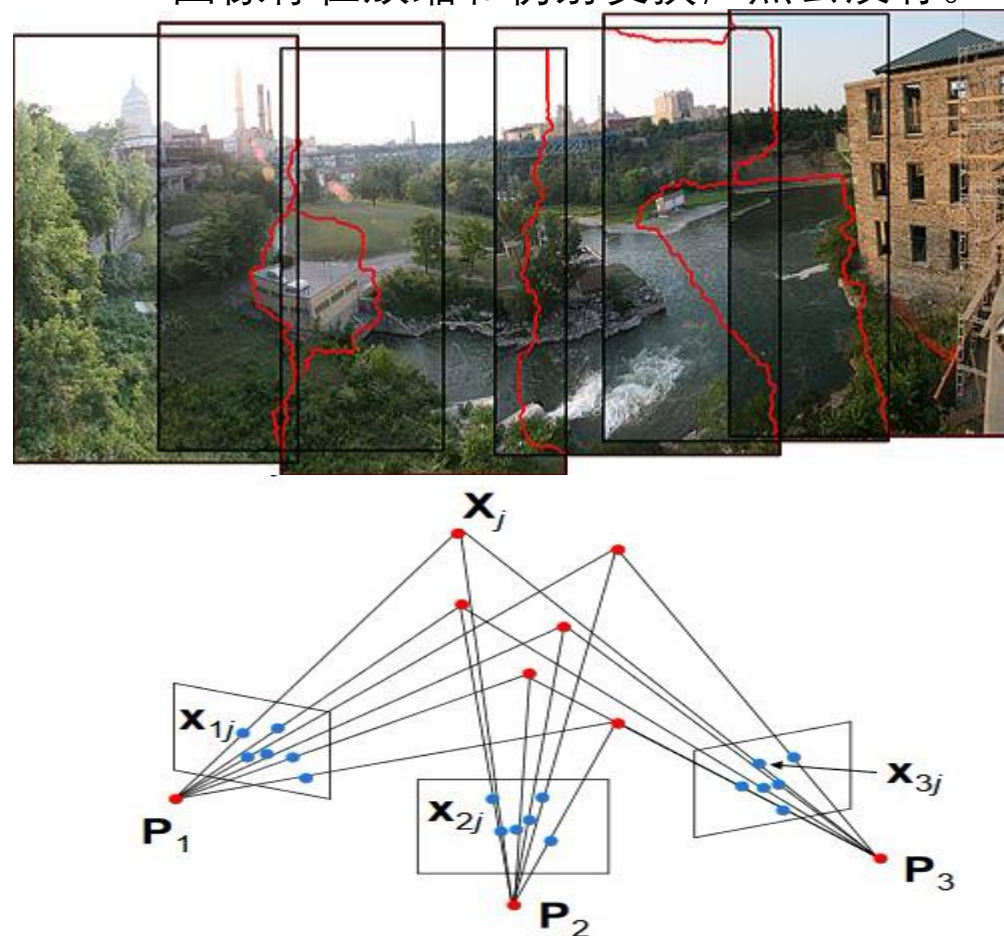
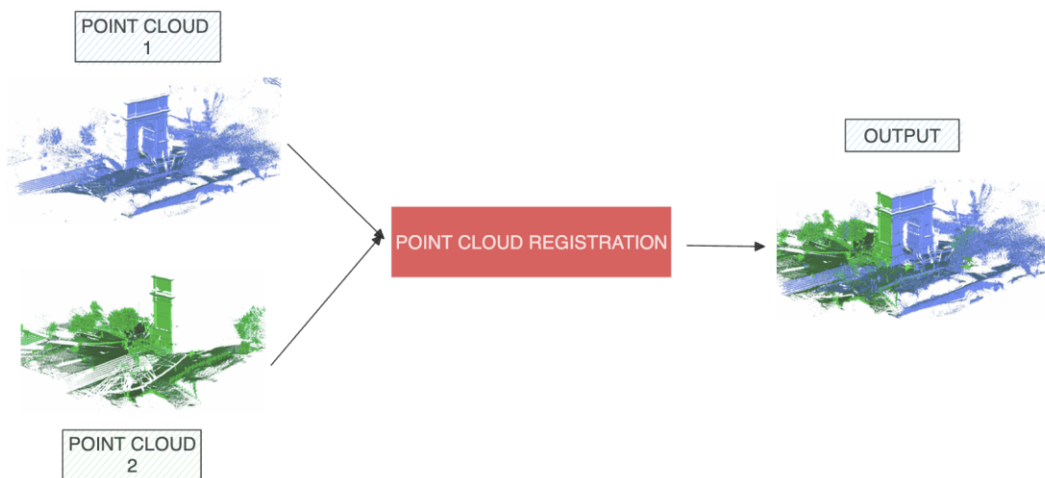
## SLAM-ICP: Iterative Closest Point Matching

迭代最近点匹配算法，主要涉及帧间配准。以2D情况为主，介绍相关算法。

图像存在放缩和仿射变换；点云没有。

### 传感器帧间观测模型：

1. 传感器在时间上以离散采样的方式获取环境信息，记录为每个时间 $t$ 下的观测数据（图像、点云等）；
2. 假设帧间观测存在重叠视场，可以支持特征提取和匹配。
3. 基于相邻帧的特征匹配可以估计帧间的位姿变化 $T$ ，获取连续位姿序列，即可叠加得到地图。



# Part 2 | SLAM

## SLAM-ICP: Iterative Closest Point Matching

**迭代最近点匹配 (ICP)**，解决的就是观测模型中特征提取和匹配问题。

求解思路：通过反复迭代，找到两组点云中**最近的对应点对**，然后估计一个最优的**刚体变换**（旋转+平移），使得源点云尽可能与目标点云对齐，即配准后的逐点误差最小。

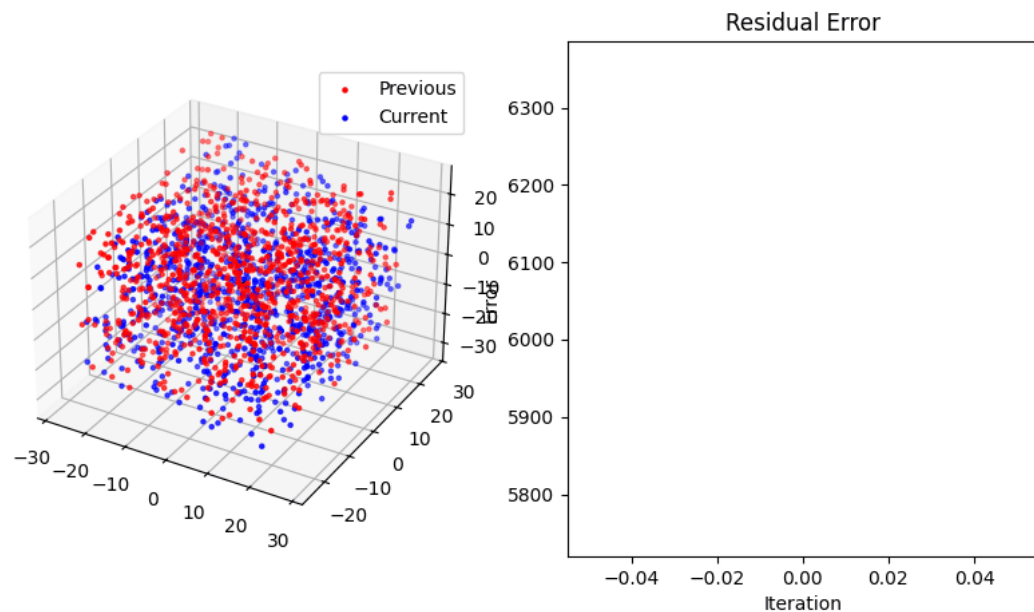
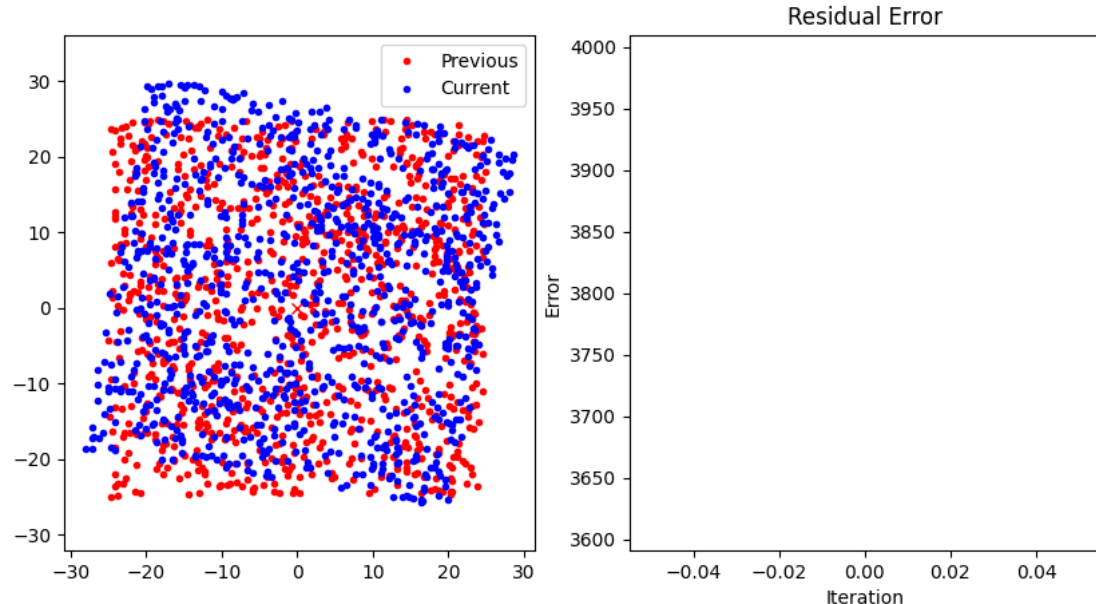
拓展到点云序列上，依次对相邻的两帧点云采用ICP，获取相对位姿。可以采用矩阵乘法叠加位姿变换，获得全局地图。

两簇点 $x=\{x_1,...,x_n\}$ 和 $p=\{p_1,...,p_n\}$

希望找到一组旋转和平移位姿，使得两簇点之间平均距离最小

$$E(R, t) = \frac{1}{N_p} \sum_{i=1}^{N_p} \|x_i - \overset{\text{旋转}}{\boxed{R}} \overset{\text{平移}}{\boxed{p_i}} - t\|^2$$

理想情况下 $x_i$ 和 $p_i$ 能够一一匹配，但实际上两组点规模都不一致。

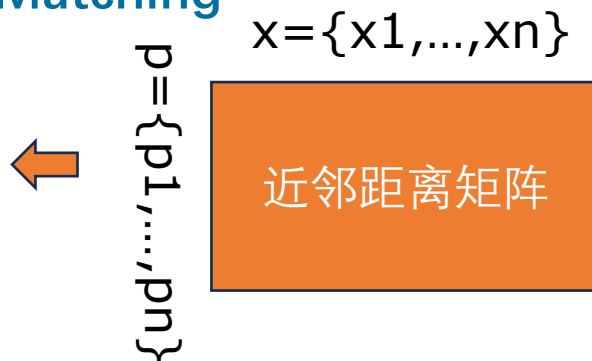
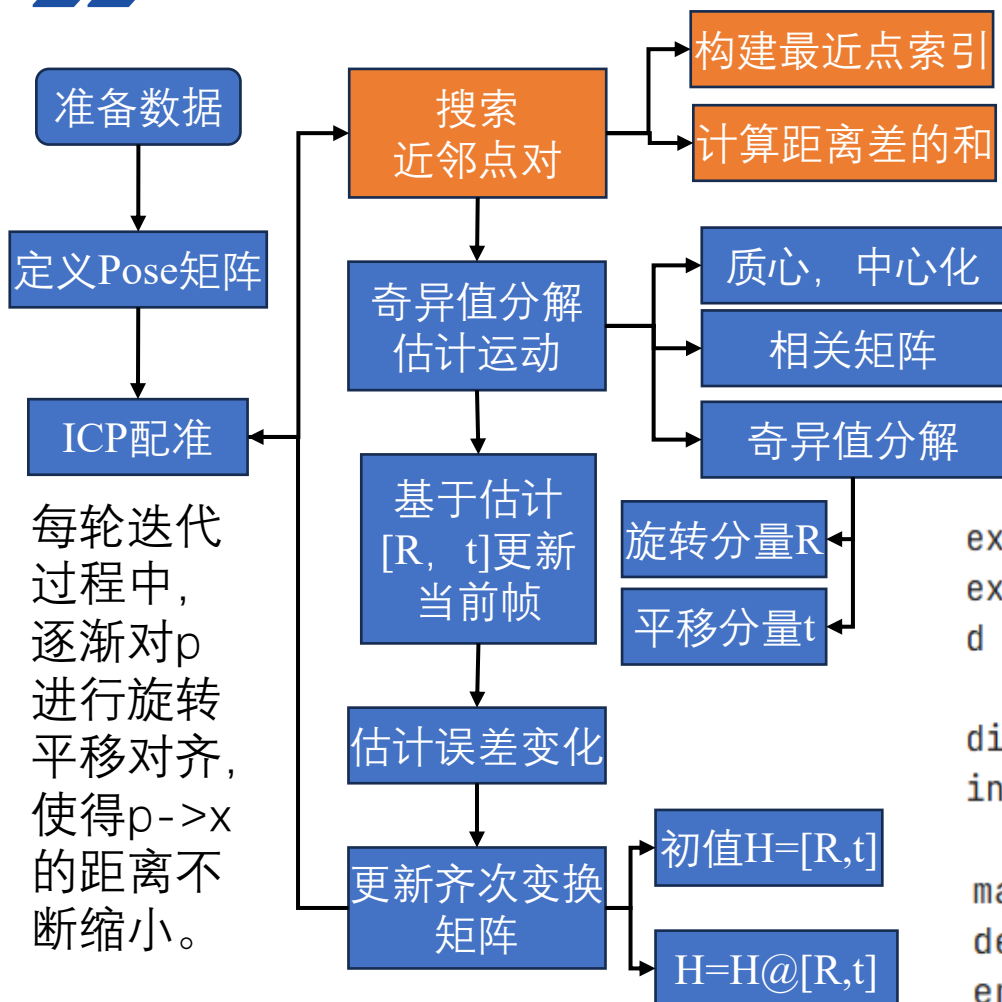




# Part 2 | SLAM



## SLAM-ICP: Iterative Closest Point Matching



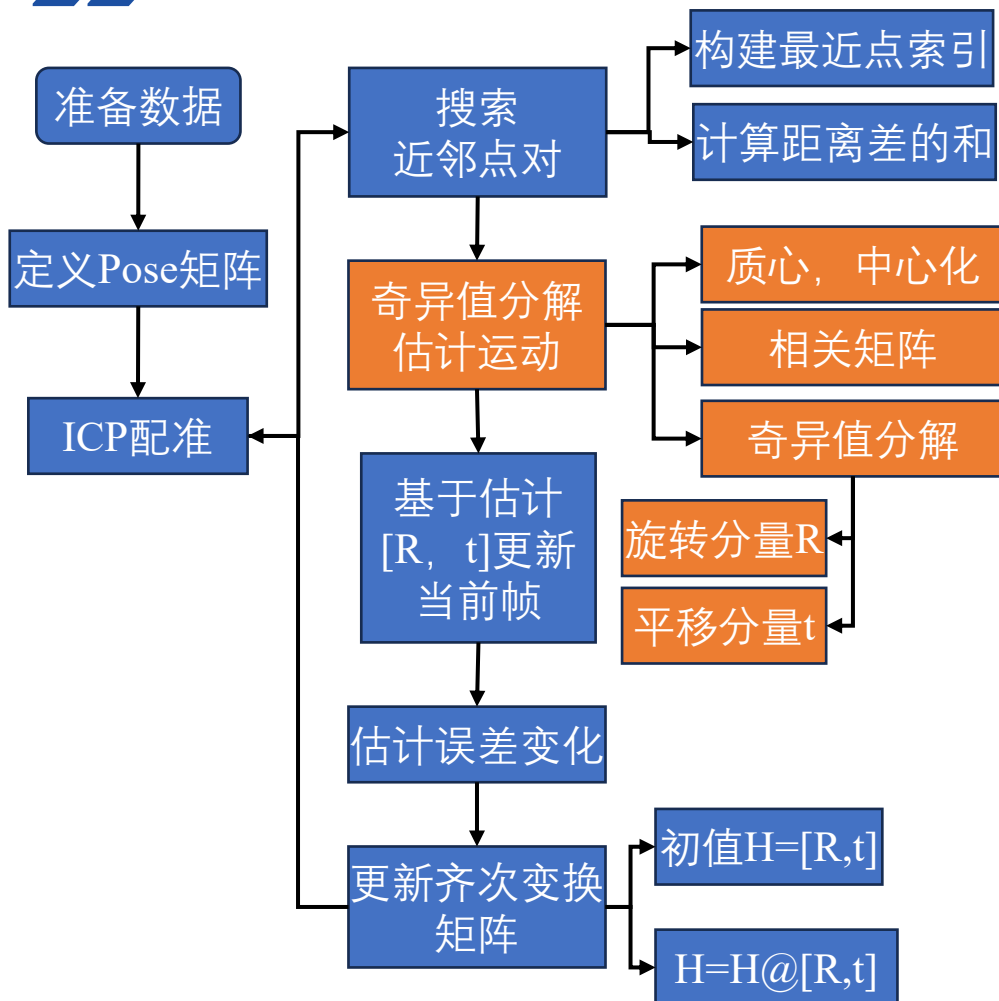
```
expanded_current = np.repeat(current_points, previous_points.shape[1], axis=1)
expanded_previous = np.tile(previous_points, (1, current_points.shape[1]))
d = np.linalg.norm(expanded_current - expanded_previous, axis=0)
```

```
distance_matrix = d.reshape(current_points.shape[1], previous_points.shape[1])
indexes = np.argmin(distance_matrix, axis=1)
```

```
matched_previous = previous_points[:, indexes]
delta = matched_previous - current_points # D
error = np.sum(np.linalg.norm(delta, axis=0))
```

# Part 2 | SLAM

## SLAM-ICP: Iterative Closest Point Matching



```
pm = np.mean(previous_points, axis=1)
cm = np.mean(current_points, axis=1)
```

```
p_shift = previous_points - pm[:, np.newaxis]
c_shift = current_points - cm[:, np.newaxis]
```

$$\mu_x = \frac{1}{N_x} \sum_{i=1}^{N_x} x_i \quad \text{通过坐标质心归一化}$$
$$\mu_p = \frac{1}{N_p} \sum_{i=1}^{N_p} p_i \quad \text{and}$$
$$X' = \{x_i - \mu_x\} = \{x'_i\}$$
$$P' = \{p_i - \mu_p\} = \{p'_i\}$$
$$W = \sum_{i=1}^{N_p} x'_i p'^T_i$$

相关矩阵

`W = c_shift @ p_shift.T`

奇异值分解

$$W = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T$$

```
u, s, vh = np.linalg.svd(W)
```

当W的秩为3时, 存在唯一最优解, 即位姿矩阵[R, t]:

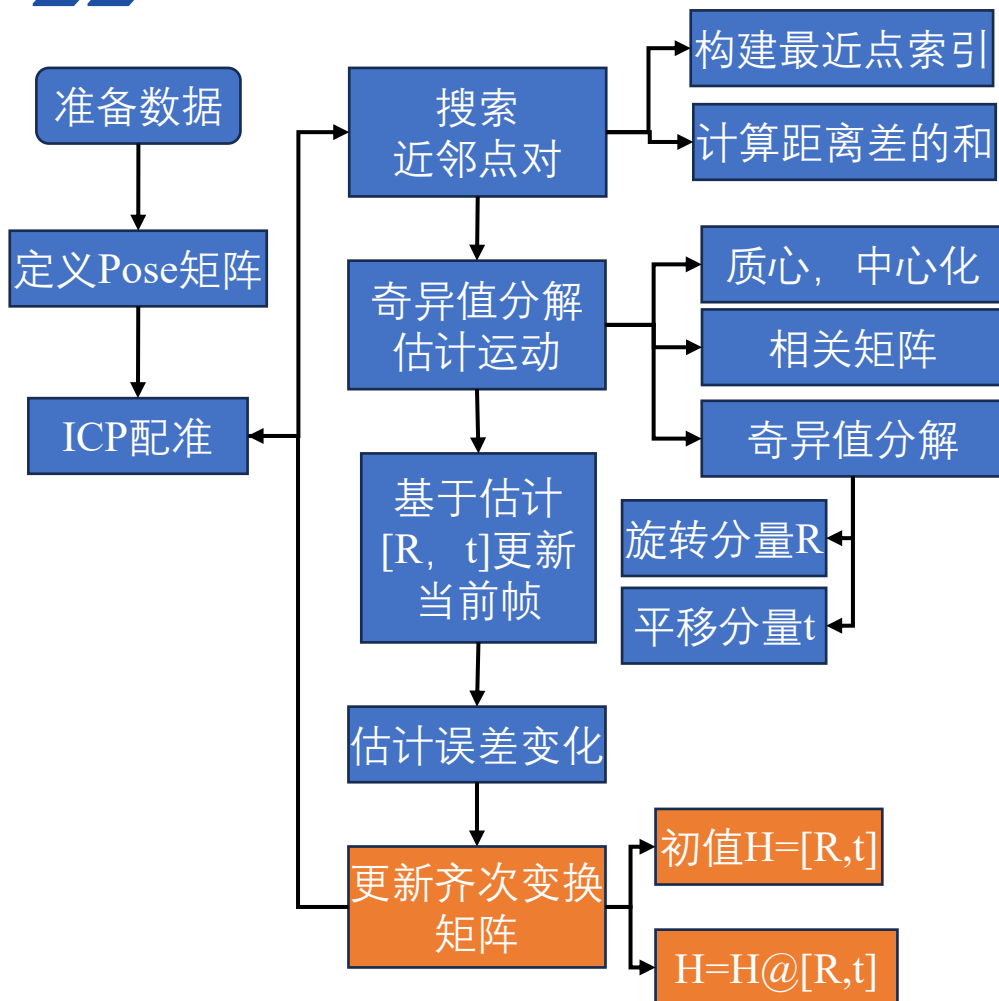
$$R = UV^T$$
$$t = \mu_x - R\mu_p$$

```
R = (u @ vh).T
t = pm - (R @ cm)
```

# Part 2 | SLAM



## SLAM-ICP: Iterative Closest Point Matching



先基于R,t 构建齐次矩阵。 $\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$   
再使用矩阵乘法更新

```
if Hin is None:
    return H
else:
    return Hin @ H
```

# Part 2 | SLAM



## SLAM-ICP: Iterative Closest Point Matching

准备数据

定义Pose矩阵

ICP配准

针对2D情况，生成规模为 $n=1000$ 的随机点previous\_points

```
px = (np.random.rand(nPoint) - 0.5) * fieldLength  
py = (np.random.rand(nPoint) - 0.5) * fieldLength  
previous_points = np.vstack((px, py))
```

并定义初始的运动位姿变换motion=[0.5, 2.0, -10] # 平移[0.5, 2.0], 逆时针旋转10度。实际上是估计的位姿矩阵的GT。

```
motion = [0.5, 2.0, np.deg2rad(-10.0)]
```

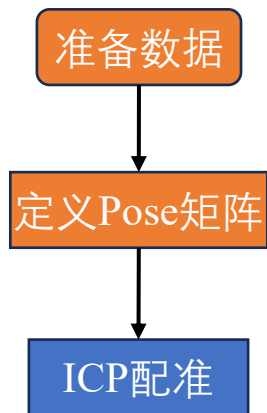
对初始点按motion进行投影，

```
cx = [math.cos(motion[2]) * x - math.sin(motion[2]) * y + motion[0]  
      for (x, y) in zip(px, py)]  
cy = [math.sin(motion[2]) * x + math.cos(motion[2]) * y + motion[1]  
      for (x, y) in zip(px, py)]  
current_points = np.vstack((cx, cy))
```

```
est_motion: [np.float64(0.2291326591639688), np.float64(2.055202208466637), np.float64(-10.000000000000005)]
```

# Part 2 | SLAM

## SLAM-ICP: Iterative Closest Point Matching



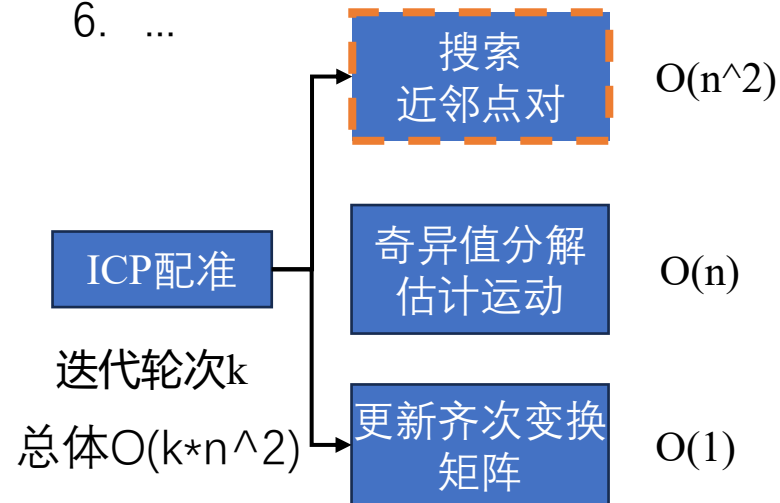
1. 扩展到3D情形，存在万向锁问题。

$$[R, t] = \begin{bmatrix} 0 & -u_z & u_y & x \\ u_z & 0 & -u_x & y \\ -u_y & u_x & 0 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

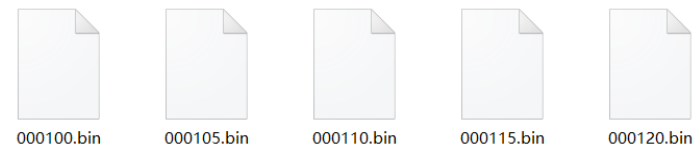
2. 两帧点云实际的规模和点并非理想的一一对应关系。

3. 点规模对算法的影响。





1. 初始噪声处理
2. 构建KD-tree等数据结构，提升搜索效率
3. 使用NDT算法进行初始解估计。
4. 使用 **点-面** 的距离替换 **点-点** 的距离
5. 在多传感器融合下，引入纹理约束colored-ICP
6. ...



实验：在数据读取脚本example.py的基础上，给定了五帧真实点云，基于ICP将点云进行叠加。



虽然大部分库都集成了ICP等算法，建议从基础版本的实现，体会整体框架。有兴趣的可以参考Open3d-pipelines

				
Points Per Second (Single Return Mode)	~ 1,300,000	~ 695,000	~ 300,000	~ 300,000
Points Per Second (Dual Return mode)	~ 2,200,000 <sup>5</sup>	~ 1,390,000	~ 600,000	~ 600,000
Refresh Rate	5-20 Hz	5-20 Hz	5-20 Hz	5-20 Hz

# 目录

## Contents

01 课程内容安排

02 ICP

03 EKF SLAM

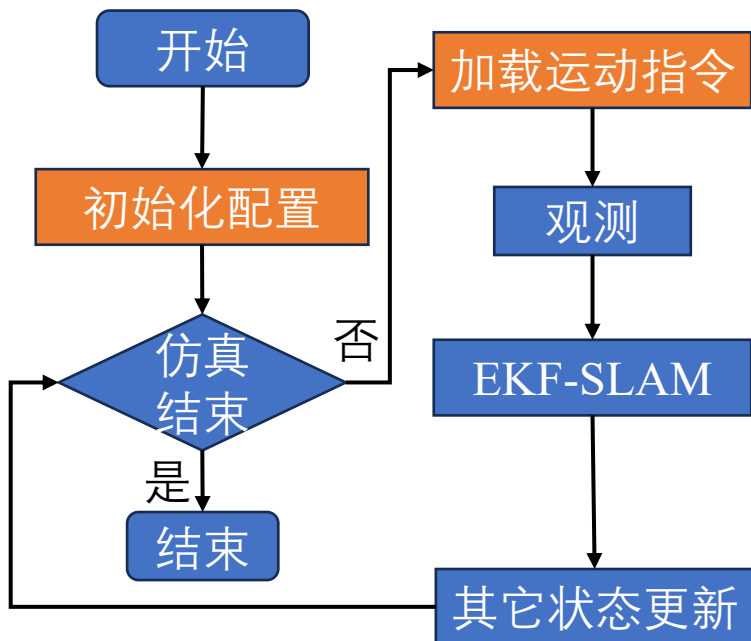
04 FastSLAM 1

05 FastSLAM 2

06 Graph-based SLAM

# Part 3 | SLAM

## » EKF-SLAM:



不同于雷达和图像的建图，是通过环境数据提炼特征。  
当观测输入直接为定位锚点时，SALM的建图对象直接为锚点的坐标。

运动指令：恒定速度和角速度的匀速圆周运动。

```
def calc_input():  
    v = 1.0 # [m/s]  
    yaw_rate = 0.1 # [rad/s]  
    u = np.array([[v, yaw_rate]]).T  
    return u
```

场景定位信息：平面不同分布的RF\_ID定位标签。

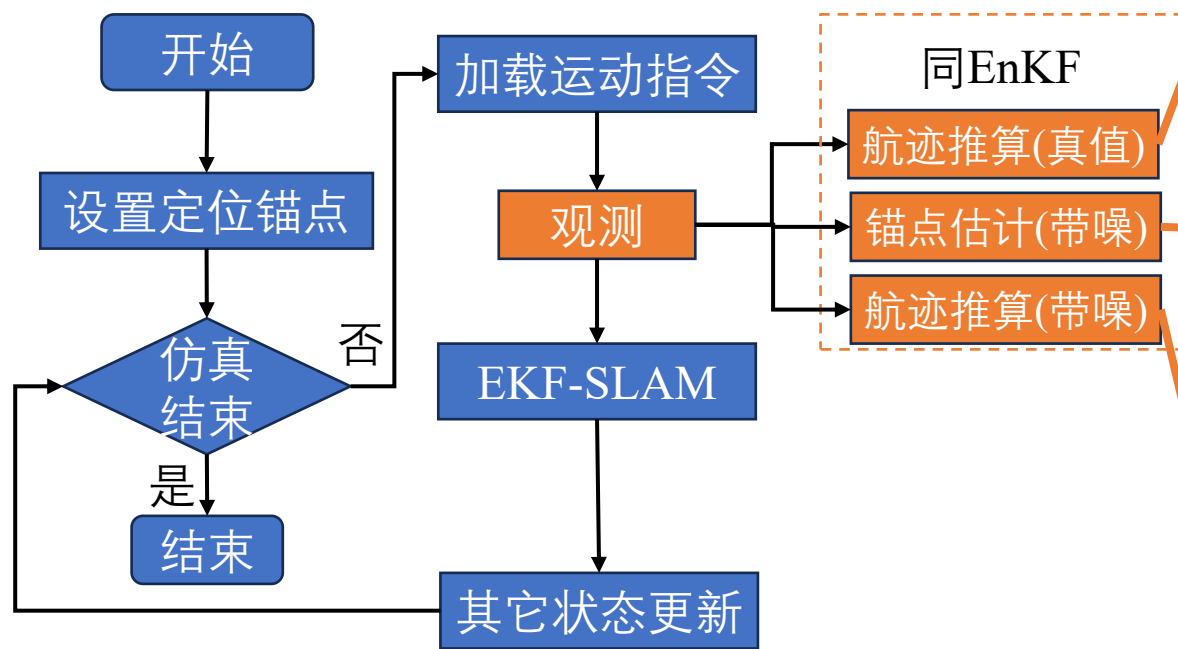
```
# RFID positions [x, y]  
RFID = np.array([[10.0, -2.0],  
                 [15.0, 10.0],  
                 [3.0, 15.0],  
                 [-5.0, 20.0]])
```

**2D 刚体运动状态模型：**平面上移动目标（如机器人）的位置和航向随时间的变化规律。用三维状态向量，位置坐标  $(x,y)$  和航向角 $\theta$ 。

```
def motion_model(x, u):  
    F = np.array([[1.0, 0, 0],  
                 [0, 1.0, 0],  # 去掉了对速度状态的估计  
                 [0, 0, 1.0]])  
  
    B = np.array([[DT * math.cos(x[2, 0]), 0],  
                 [DT * math.sin(x[2, 0]), 0],  
                 [0.0, DT]])  
  
    x = (F @ x) + (B @ u)  
    return x
```

# Part 3 | SLAM

## » EKF-SLAM:



- 基于机器人真实位姿和环境中路标 (RFID) 的位置，生成带有噪声的观测数据（距离、角度、路标 ID）。
- 对原始控制输入添加噪声，模拟实际运动中的控制误差，并更新航迹推算（DR）的位姿。

```
def observation(xTrue, xd, u, RFID):
```

```
    xTrue = motion_model(xTrue, u)
```

```
    # add noise to gps x-y
```

```
    z = np.zeros((0, 3))
```

```
    for i in range(len(RFID[:, 0])):
```

```
        dx = RFID[i, 0] - xTrue[0, 0]
```

```
        dy = RFID[i, 1] - xTrue[1, 0]
```

```
        d = math.hypot(dx, dy)
```

```
        angle = pi_2_pi(math.atan2(dy, dx) - xTrue[2, 0])
```

```
        if d <= MAX_RANGE:
```

```
            dn = d + np.random.randn() * Q_sim[0, 0] ** 0.5 # add noise
```

```
            angle_n = angle + np.random.randn() * Q_sim[1, 1] ** 0.5 # add
```

```
            zi = np.array([dn, angle_n, i])
```

```
            z = np.vstack((z, zi))
```

```
    # add noise to input
```

```
    ud = np.array([[
```

```
        u[0, 0] + np.random.randn() * R_sim[0, 0] ** 0.5,
```

```
        u[1, 0] + np.random.randn() * R_sim[1, 1] ** 0.5]])
```

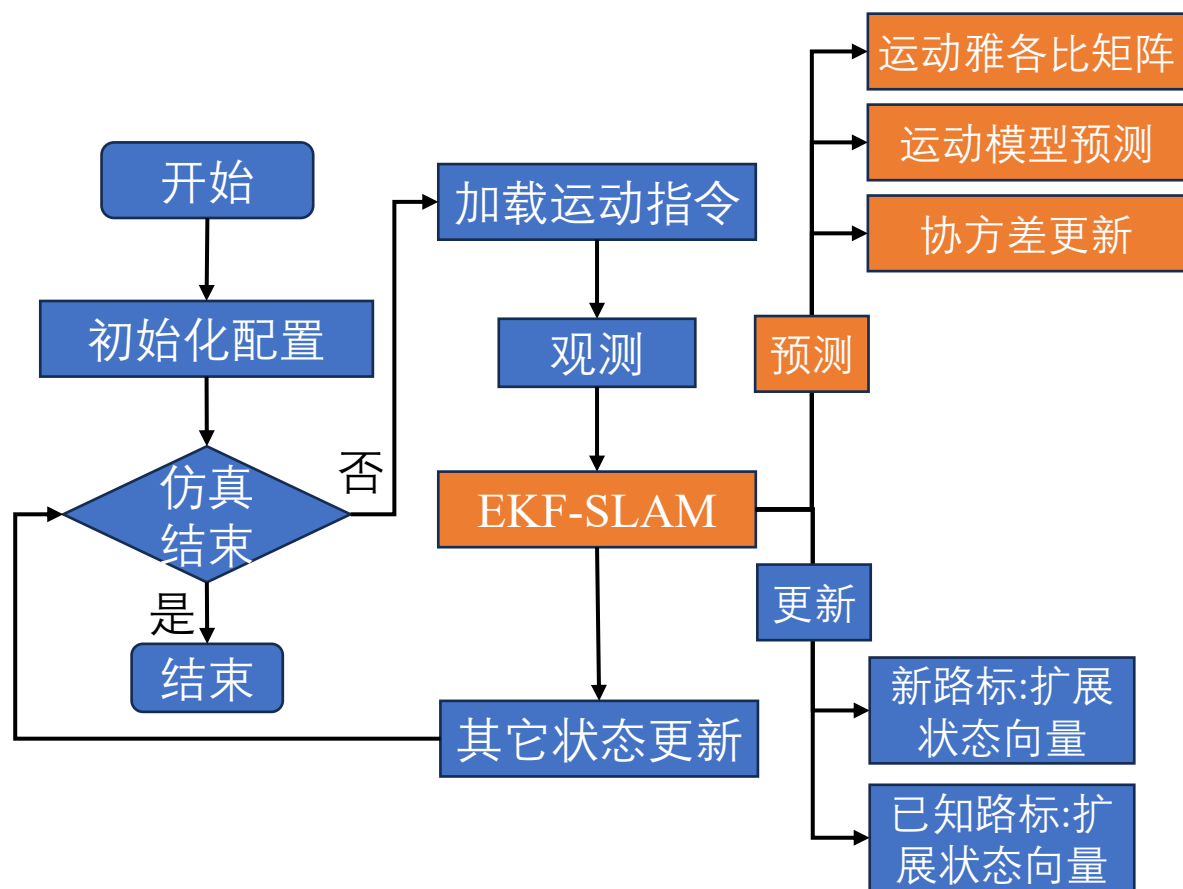
```
    xd = motion_model(xd, ud)
```

```
    return xTrue, z, xd, ud
```



# Part 3 | SLAM

## » EKF-SLAM:



$$x = \begin{bmatrix} x_r \\ y_r \\ \theta_r \\ m_1 \\ m_2 \\ \vdots \\ m_N \end{bmatrix}$$

$m_i = (x_i, y_i)$ : 第  $i$  个路标的坐标

运动指令, 观测中添加噪声

```
G, Fx = jacob_motion(xEst, u)
xEst[0:STATE_SIZE] = motion_model(xEst[0:STATE_SIZE], u)
PEst = G.T @ PEst @ G + Fx.T @ Cx @ Fx
```

运动控制只考虑Agent

雅各比矩阵计算时, 则要考虑对RF\_ID状态的扩展。  
线性化运动方程时 (无RF\_ID状态), 直接基于状态转移函数对状态求一阶偏导一位姿变换的偏导。

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ \theta_{t+1} \end{bmatrix} = \begin{bmatrix} x_t + v \Delta t \cos \theta_t \\ y_t + v \Delta t \sin \theta_t \\ \theta_t + \omega \Delta t \end{bmatrix} \rightarrow \begin{bmatrix} \frac{\partial x_{t+1}}{\partial x_r} & \frac{\partial x_{t+1}}{\partial y_r} & \frac{\partial x_{t+1}}{\partial \theta_r} \\ \frac{\partial y_{t+1}}{\partial x_r} & \frac{\partial y_{t+1}}{\partial y_r} & \frac{\partial y_{t+1}}{\partial \theta_r} \\ \frac{\partial \theta_{t+1}}{\partial x_r} & \frac{\partial \theta_{t+1}}{\partial y_r} & \frac{\partial \theta_{t+1}}{\partial \theta_r} \end{bmatrix} \rightarrow \begin{bmatrix} 0 & 0 & -v \Delta t \sin \theta_t \\ 0 & 0 & v \Delta t \cos \theta_t \\ 0 & 0 & 0 \end{bmatrix}$$

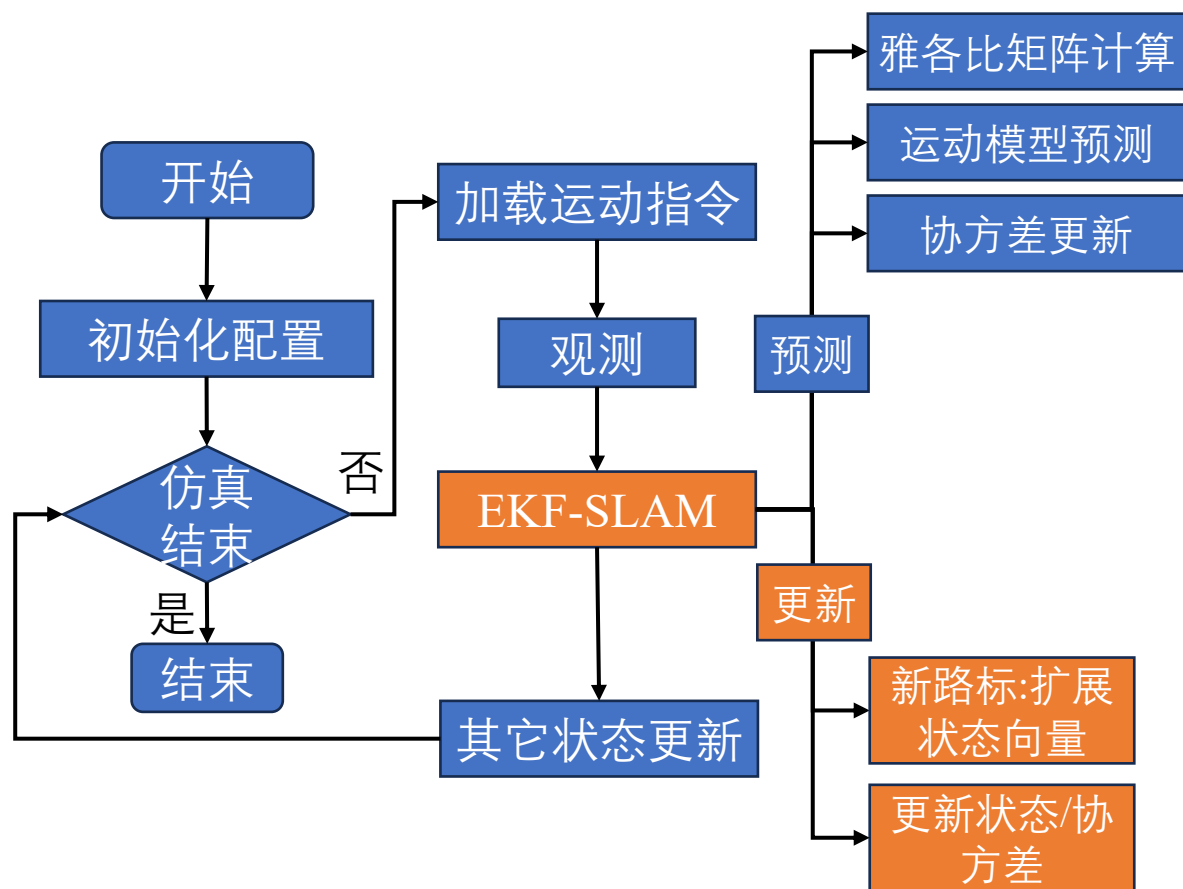
RF\_ID部分保持不变, 将位姿的偏导嵌入。  $G = I + F_x^\top j F F_x$

带入  $P_{t+1} = G_t P_t G_t^\top + F x^\top C_x F x$

- 将非线性的运动模型线性化, 以更新状态协方差矩阵。
- 基于观测路标更新已知路标信息, 更新状态和协方差。

# Part 3 | SLAM

## » EKF-SLAM:



- 将非线性的运动模型线性化，以更新状态协方差矩阵。
- 基于观测路标更新已知路标信息，更新状态和协方差。

观测模型：带噪的相对距离和方向  $z=h(x,m)+v$

$$z = \begin{bmatrix} \sqrt{(x_i - x_r)^2 + (y_i - y_r)^2} \\ \arctan 2(y_i - y_r, x_i - x_r) - \theta_r \end{bmatrix} + v$$

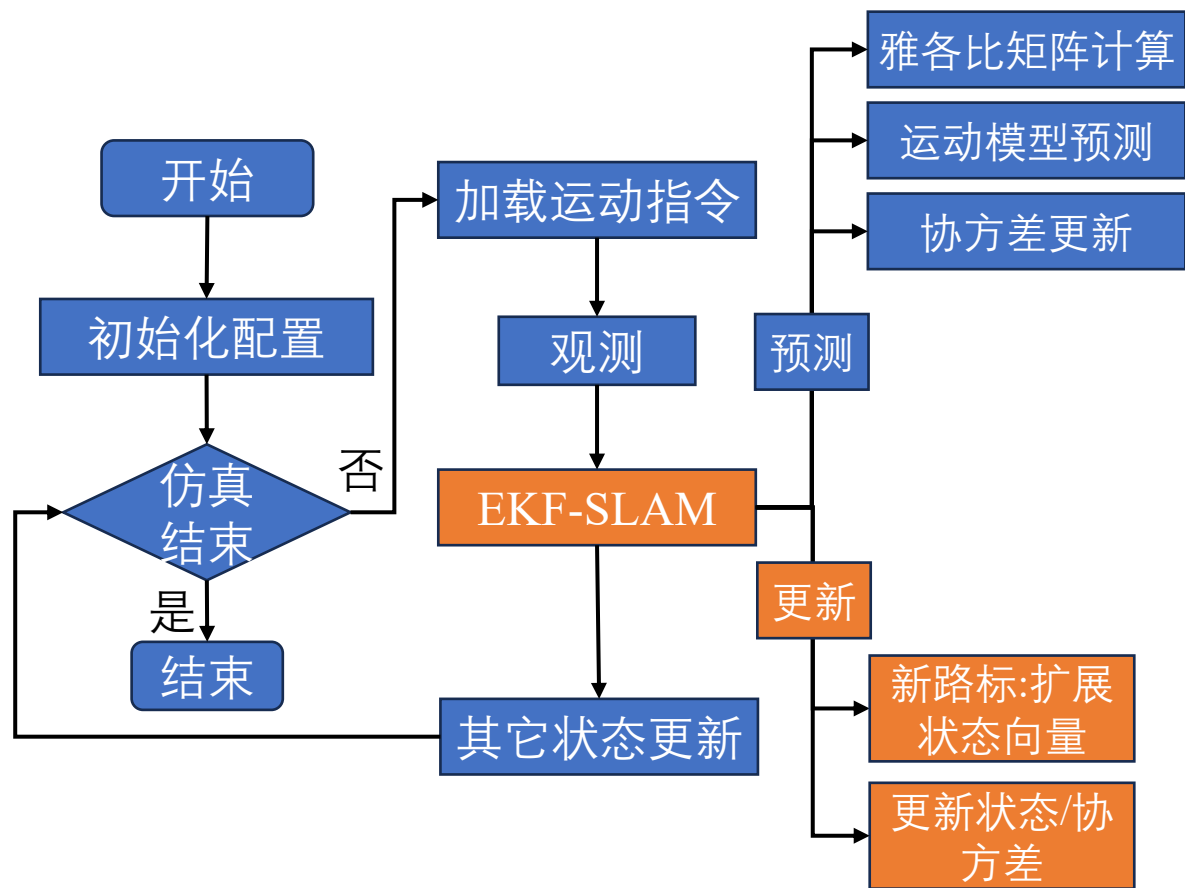
数据关联：马氏距离（考虑协方差方向），判断观测对应的路标。大于阈值时为新路标，触发更新状态向量 $x$ 。

- 更新过程，初始不涉及路标。直接使用预测结果输出。
- 出现新观测 $z$ 时，对每一个观测数据：
  - 数据关联：获取路标坐标信息。
  - 计算innovation（仅当前遍历的路标）：
    - 到观测路标的距离和角度
    - 计算观测雅各比矩阵：Jacob\_h
    - 更新观测状态
  - 基于马氏距离筛选最相关路标
- 若为新路标，更新估计状态，状态的协方差矩阵
- 基于所有路标计算innovation
- 更新Kalman增益
- 更新状态协方差

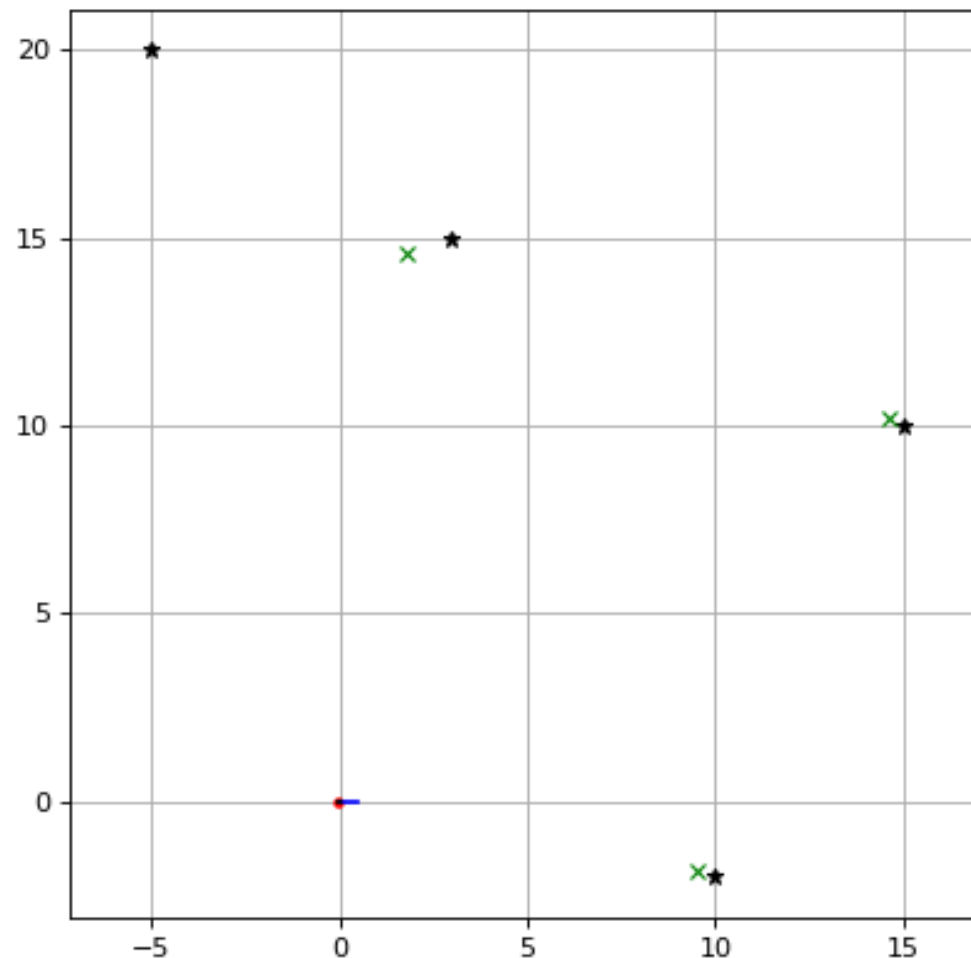
判断当前路标是否为未知 (Mapping)

# Part 3 | SLAM

## » EKF-SLAM:



- 将非线性的运动模型线性化，以更新状态协方差矩阵。
- 基于观测路标更新已知路标信息，更新状态和协方差。



可以观测到，在SLAM的前半段，因为存在RF\_ID，红色EKF结果相对较好；但经过位于 $[-5, 20]$ 的最后一个路标之后，后续因为缺乏有效观测信息，算法没有能够修正自身定位的参考，累积误差持续放大。实验：1.通过增加额外的RF\_ID，优化定位结果；2.通过修改仿真过程的噪声分布，再观察结果。

# 开始实验



北京航空航天大学  
BEIHANG UNIVERSITY