Project

环境部署（noetic）

项目仓库位置：https://github.com/zzzzzzzzjx/myproject.git

install Plain Text

```
# Download code
git clone https://github.com/zzzzzzzzjx/myproject.git
git submodule init && git submodule update

# Install dependencies
bash install_dependencies.sh
```

terminal Plain Text

```
# Basic example of how to build and run
# Open one terminal
cd xju-robot && source devel/setup.bash
roslaunch xju_simu simple_world.launch
# Open another terminal
roslaunch xju_pnc move_base_flex.launch
```

项目结构

这是一个集成式的导航项目，核心内容主要由以下几个部分组成：

1. sim：仿真环境 & 机器人模型（URDF & World）
2. pnc：导航项目（move_base, move_base_flex）
3. slam：建图项目（amcl, carto_localization, localiazation, lio_sam, etc）
4. explore：自主建图（cartographer）
5. coverage_path_planner：全覆盖路径算法（path_planning）

导航

move_base_flex

简介：move_base_flex.launch主要由以下四个部分组成（接下来的关于割草区域的全路径生成、路径跟随均集成至该launch文件中）：

1. Global planner:move_base
2. Local planner:TEB
3. Localization:carto（amcl）
4. Structure:FSM

   move_base_flex提供四个action,这些动作可以被外部执行者用来执行各种导航任务：

   a. 获取路径get_path

   b. 执行路径exe_path

   c. 脱困recovery

d. 原始move_base

terminal Plain Text

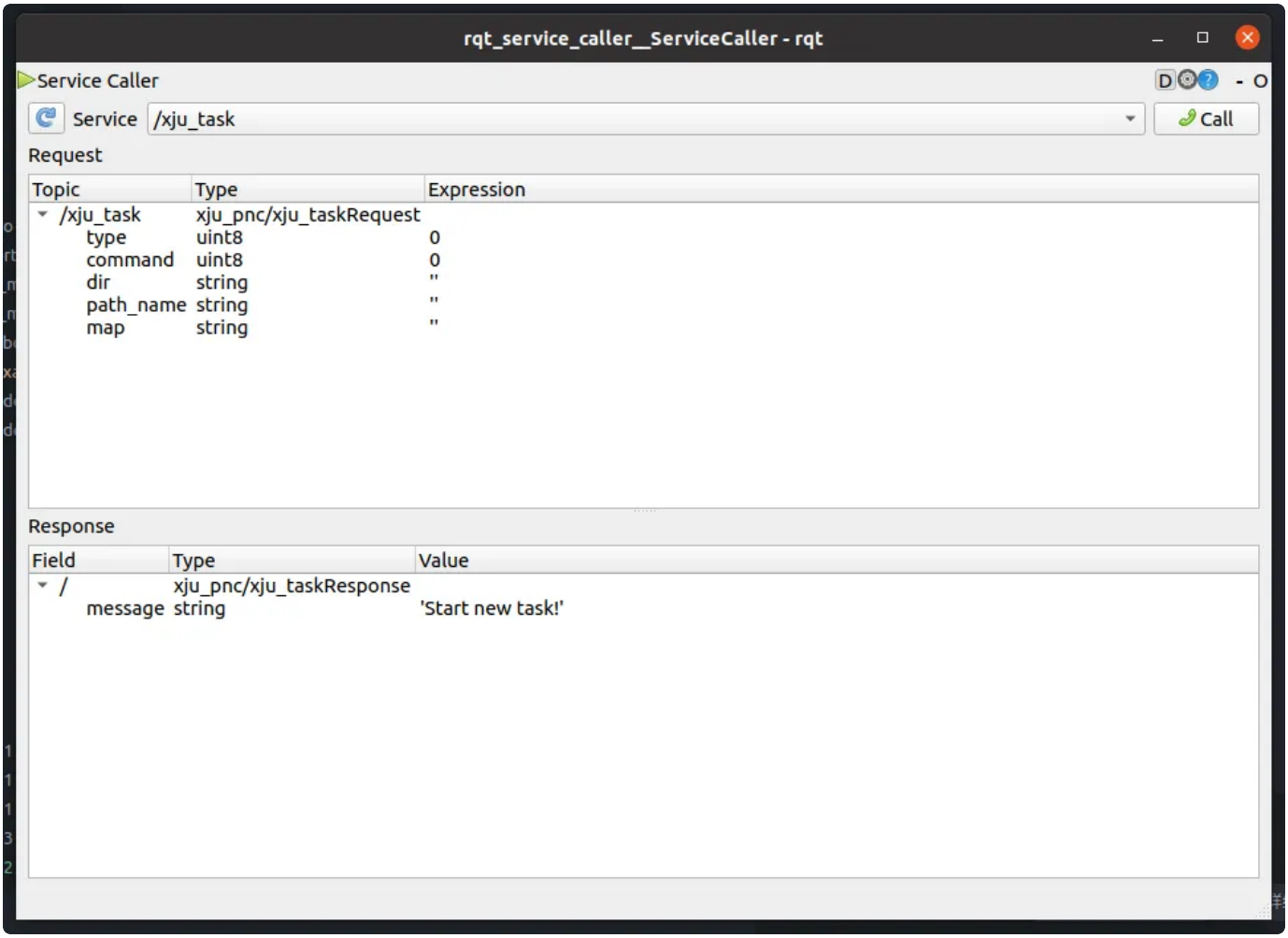
```
roslaunch xju_pnc move_base_flex.launch
```

1. Task Manual

```
uint8 EXECUTE = 0
uint8 RECORD = 1
uint8 LOAD_TRAFFIC_ROUTE = 2
uint8 QR_NAV = 3

uint8 START = 0
uint8 PAUSE = 1
uint8 STOP = 2

uint8 KEEP_TEACH = 1
uint8 KEEP_COVER_ZZ = 2
uint8 KEEP_COVER_BS = 3
uint8 DISCARD = 4
uint8 KEEP_TRAFFIC_ROUTE = 5 # only support two points for now

uint8 type #EXECUTE RECORD LOAD_TRAFFIC_ROUTE QR_NAV
uint8 command #START PAUSE STOP KEEP_TEACH KEEP_COVER DISCARD KEEP_TRAI
string dir
string path_name
string map
---

string message
```

服务调用： Plain Text

```
rosrun rqt_service_caller rqt_service_caller
```



2. Teching Path

简介：示教路径是通过人工操作（如遥控或物理引导）让机器人移动，同时记录其运动轨迹（位置、方向）生成的路径，通过记录机器人的位姿（geometry_msgs/Pose）序列实现。调用xju_task服务开启录制功能，通过Rviz中的Pubilish Point/移动机器人进行路径点的生成，再次调用服务对生成的路径进行抓取并存入path中。后续可用于割草区域全覆盖路径的外围边框设置。

使用方法：

1. 调用服务，选择xju_task
2. 开始录制：type：1 command：0
3. 设置路径：以下使用其中一个控制器运行举例

terminal — Plain Text

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

示教路径跟随：type：0 command：0 （复制路径名字）

3. Prohibition Layer

简介：为满足割草区域的分离与筛选，通过对无需割草区域设置为虚拟墙而进行隔离。对虚拟墙设置costmap_params.yaml（plugin ： keepout layer）

使用方法：

1. 调用服务，选择/xju_zone（brigde同时调用了global和local的zone）
2. 开始记录：command：3
3. 设置虚拟墙：Publish Point选择线、区域
4. 结束录制：command：4

4. Coverage Path Planning

简介：参考市面上目前较为成熟的两种全覆盖路径算法，一种面向于割草区域为矩形的Z字型全覆盖路径，另一种则面向于割草区域为不规则形状的回字型全覆盖路径。故采用Zigzag和Backshape两种成熟的策略生成Coverage path，其中加入了贝塞尔做路径平滑优化。状态机通过get_path获得路径切换至execute进入exe_path进行路径跟随。同时通过状态机接入move_base进行Navigation & 动态避障。

使用方法：

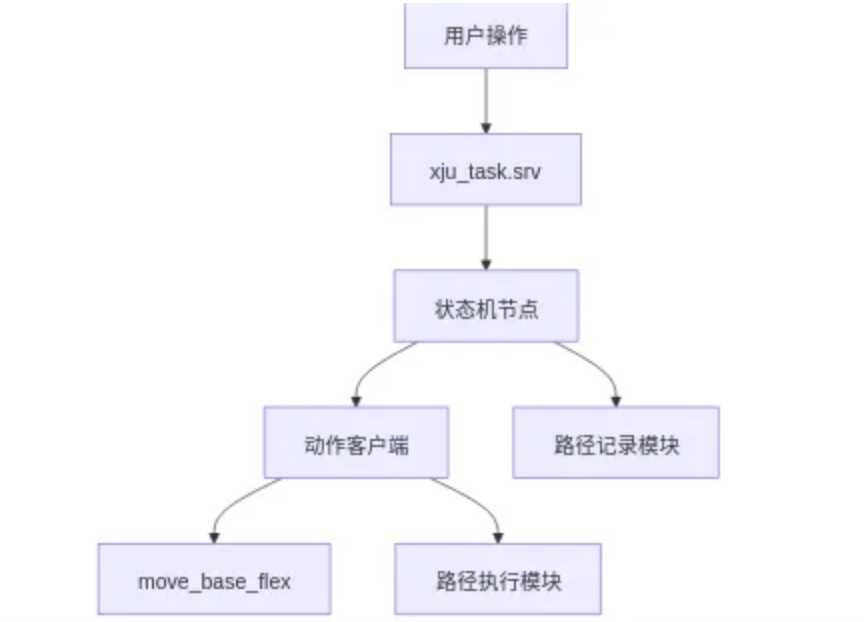
全覆盖路径规划器算法：CoverPathPlanner.cpp

服务文件（已接入到xju_task.srv）：Zigzag/Backshape

1. 调用服务，选择xju_task
2. 开始录制：type：1 command：0
3. 设置边框：Publish Point划线/示教路径Teching Path
4. 生成路径：type：1 command：2 （Z字型：2，回字型：3）
5. 执行路径：type：0 command：0 （复制路径名字）

state_machine中coverage_path的逻辑

简介：跟示教路径逻辑无差别，但是添加了一个EOP作为两个路径的分割，并且通过EOP判断读进来的路点是不是属于两点路径，若是两点路径，则将其分开存入routes_(是一个存储多条路径的队列）；其中在子状态running_state()里面在走完第一条路径后不会直接结束，而会判断如果routes_.empty()为真，说明所有路径均已完成，调用reset() 重置状态机回到Idle，未完成则切换第二条路径的front点作为cur_route。



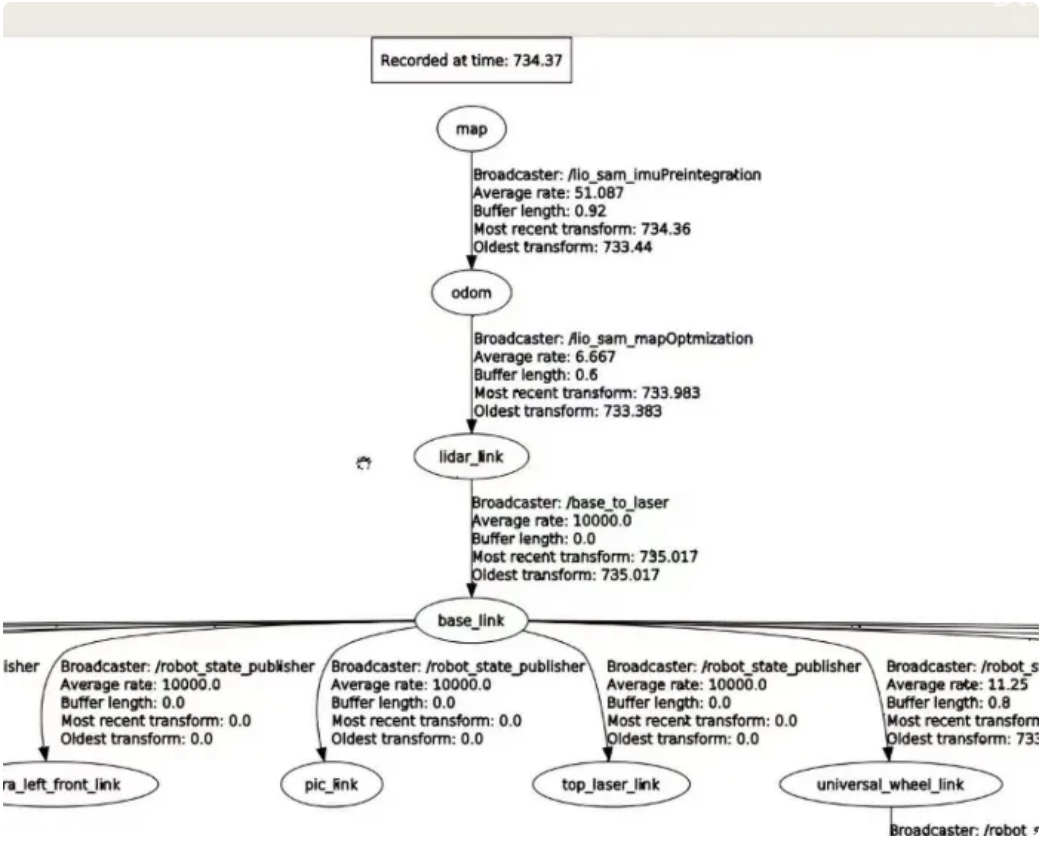
- **路径完成检测**：当exe_ctrl_（路径执行动作）的状态为SUCCEEDED时，表示当前路径（cur_route_）已成功执行完毕。
- **路径队列检查：**routes_是一个存储多条路径的队列（例如，从文件读取的多条全覆盖路径）。如果routes_.empty()为真，说明所有路径均已完成，调用reset()重置状态机。
- **切换至下一条路径：**取出新路径：cur_route_ = routes_.front()将下一条路径设为当前路径。
- **移除已处理路径：**routes_.pop_front()从队列中移除已完成的路径。
- **重置索引：**cur_index_ = 0将路径点索引重置为0，即从新路径的起点开始。
- **切换至Goto状态：**running_state_ = RunStateValue::Goto进入点对点导航状态。
- **送Goto目标：**send_goto(cur_index_)向导航系统发送新路径起点的目标。

5. LIO_SAM

简介：基于割草区域存在着坡度变化，采取三维的激光建图；同时考虑到后续在户外场景需融合RTK提高定位精度，选用了LIO_SAM的三维点云建图算法。相较于其他算法，LIO_SAM提供了扩展卡尔曼滤波(EKF)的转换节点navsat_transform进行GPS的融合，无需自行进行额外融合。

配置文件：xju_robot/src/slam/param/lio_config.yaml（包含话题设置、不同型号的传感器（雷达）的设置、IMU参数设置以及GPS的设置）

注意：odom由lio_sam中的lio_sam_mapOptmization发布，注意是否与urdf中的gazebo重合发布，若重复发布会影响tf，需关闭gazebo中的odom发布。（具体修改参考原作者fork的lio_sam：https://github.com/Mr-Tony921/LIO-SAM.git）下图为整个tf的关系：



数据集测试：

数据集：https://drive.google.com/drive/folders/1gJHwfdHCRdjP7vuT556pv8atqrCJPbUq

数据集测试LIO_SAM Python

```
cd ~/bag的路径/bag
rosbag play campus_small_dataset.bag --clock -r 0.5 # 降速播放便于调试
```

6. StateMcahine

简介：以下是关于项目中的决策架构分层有限状态机（FSM）的源代码解读。

基于ROS框架：命名空间是xju::pnc，类名为StateMachine。这个类处理机器人的状态，比如空闲、记录、运行、暂停等。状态机的run()方法在一个循环中处理不同的状态，比如Idle、Record、Run，然后

Run中拥有三个子状态Follow、Wait、Goto等。

关键点：

a. **状态机管理（StateMachine）**
    i. **主要状态包括Idle、Record、Run、Pause**
    ii. **Run的三个子状态：Follow、Wait、Goto**
b. **路径安全检测（Wait）**
    i. **is_free，is_danger**
c. **动作客户端（Action Clients）**
    i. **mbf_msgs::MoveBaseGoal，mbf_msgs::ExePathGoal**

StateMachine的构造函数：初始化了一些ROS相关的组件，比如TF缓冲区和监听器，动作客户端（GotoCtrl和ExeCtrl），以及半结构化路径规划器。这里还初始化了成本转换表，将百分比转换为代价地图的值，范围是0–254。

init()函数：里面设置了各种发布器和订阅器，比如速度发布器、路径发布器、服务客户端等。服务包括任务服务（task_srv_）和覆盖路径服务（coverage_srv_）。动作客户端连接到move_base_flex的move_base和exe_path动作服务器。

主循环run()：以10Hz的频率运行，根据当前状态执行相应的操作。比如在Record状态时，调用record_path()方法记录路径，并发布记录的路径。在Run状态时，处理运行时的子状态，如Follow、Wait、Goto。

running_state()：处理运行状态下的三个子状态。

- Follow状态处理路径跟随，检查执行结果，失败则切换到Goto状态。
- Wait状态等待障碍清除，超时则切换至Goto。
- Goto状态处理点对点导航，成功后回到Follow状态。

send_goto和send_exe方法：分别发送目标点和执行路径的动作。这些动作使用ROS的actionlib库，发送目标并设置回调函数。例如，当Goto动作完成时，调用goto_done回调。

任务服务（task_service）：处理不同类型的请求，例如，当接收到执行任务请求时，读取路径文件，启动运行状态。记录路径时，根据命令保存或丢弃数据。

状态机的逻辑和核心算法：

- 状态机的主要状态包括Idle、Record、Run、Pause
- 在Run状态下，有三个子状态：Follow、Wait、Goto
- 状态之间的转换由动作结果或障碍检测触发

路径的记录record_path()：通过获取机器人当前位姿，并检查距离和角度变化，决定是否记录新的路径点。记录的路径点会被保存到文件，或者用于覆盖路径生成。

执行路径move_base_flex：使用move_base_flex的动作客户端进行导航。路径跟随（Follow）使用ExeCtrl执行路径，而点对点导航（Goto）使用GotoCtrl。状态机根据执行结果调整当前路径点和状态。

核心避障算法（is_free，is_danger）：路径点的安全检查，这些方法基于代价地图的代价值判断路径点是否可行。动态障碍处理通过等待或重新规划路径点来实现。

动作客户端（Action Clients）：通过mbf_msgs::MoveBaseGoal和mbf_msgs::ExePathGoal发送目标，利用actionlib的客户端与服务器通信。回调函数处理完成状态，如goto_done和exe_done。