目录

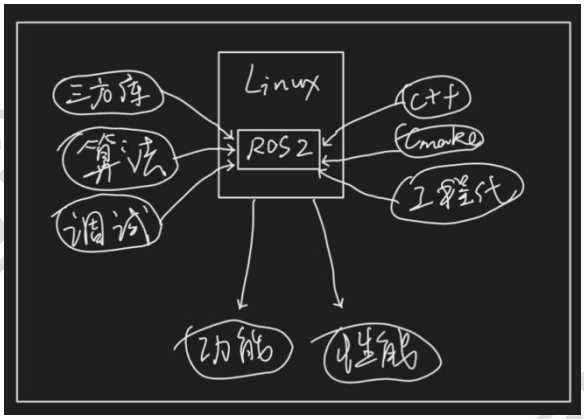
一，绪论

1，课程介绍

1.1，课程概述

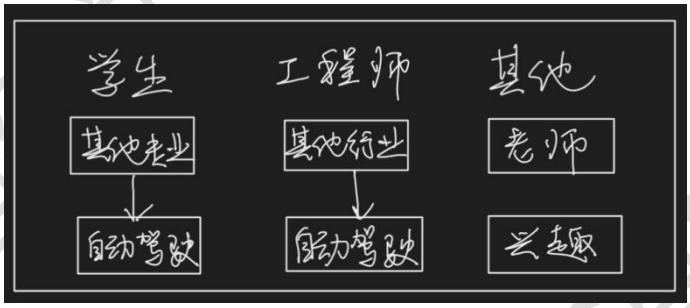
本课程讲述如何在 Linux 系统下基于 ROS2 使用 C++开发一个自动驾驶汽车的决策规划算法模块，该模块能让车辆具备基本的循迹行驶、静态绕障、动态避障等功能。该模块按照一定的编程规范编写，具备良好的运行效率、可解释性、解耦性、可读性、可扩展性，符合行业里很多企业工程化的要求，可以为企业开发提供依据和思路。

课程将详细讲解 ROS2 等工具链的使用、决策规划的算法原理、C++的语法应用、代码的编写、场景的调试等内容，按照企业实战开发的要求完成，学完后可以具备决策规划算法工程师的基本所需技能，可提升求职时的优势，并可快速融入企业实战项目开发中。



1.2，授课对象

- 希望转专业到自动驾驶决策规划算法方向相关专业的在校学生；
- 已经在自动驾驶决策规划算法方向相关专业，希望巩固基础、提升技术的在校学生；
- 希望转行到自动驾驶决策规划算法领域的工程师；
- 已经在自动驾驶决策规划算法领域从业，希望巩固基础、提升技术的工程师；
- 自动驾驶相关专业的老师、培训师；
- 对自动驾驶感兴趣的人；



本课程是一门自动驾驶的进阶课程，并不是一门“零基础”的入门课程，学习之前应该具备以下基础：

- 自动驾驶基本概念；
- C++基础语法；
- Linux 基础；
- ROS 或 ROS2 基础；
- 几何、线性代数基础；

1.3，课程收获

- 具备决策规划算法工程师的基本所需技能，提升求职时的优势；
- 快速融入到企业项目开发中；
- 完成学业相关的论文；
- 强化 C++、Linux、ROS2 等基础，扩展就业面；
- 提升解决问题的能力；
- 多一条出路，多一份选择；

1.4，课程内容

- 一，绪论：课程整体介绍，决策规划概述，前置准备工作；
- 二，工具链与环境搭建：Linux，ROS2，VScode，CMake，三方库的安装；
- 三，项目架构设计与搭建；
- 四，数据接口定义；
- 五 - 十，算法核心部分：PNC 地图与全局规划，参考线，路径决策与规划，速度决策与规划，运动的实现，轨迹合成；
- 十一，代码完善与综合调试；
- 十二，课程总结；

1.5，授课方式

- 实操讲解：工具链的安装与环境搭建；
- 理论讲解：数学基础、算法原理；
- 代码编写：数学建模、手把手代码编写；
- 大作业；
- 课后答疑；

2，版权声明

3，决策规划概述

3.1，端到端还是基于规则？

这是个开放性问题，我认为作为自动驾驶算法从业者，两者都要会，至少在目前阶段，做端到端的也要懂基于规则的算法技术原理，这是基本功。而基于规则的算法岗位，现在仍然有很多招聘需求，未来很长一段时间内将与端到端共存。

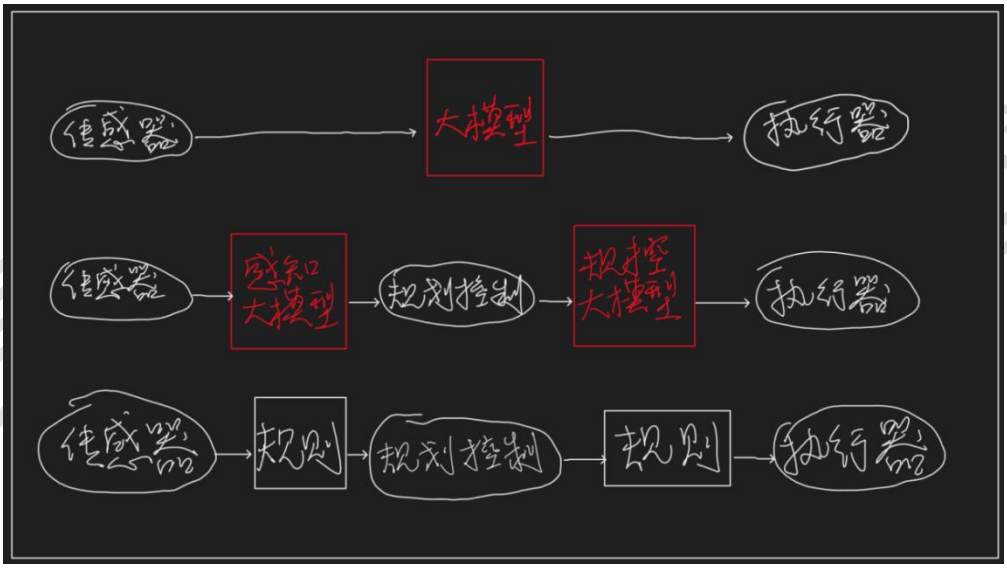
端到端的优点：

- 车辆行为更像人类；
- 有覆盖无穷多场景的能力；
- 无需写大量代码；

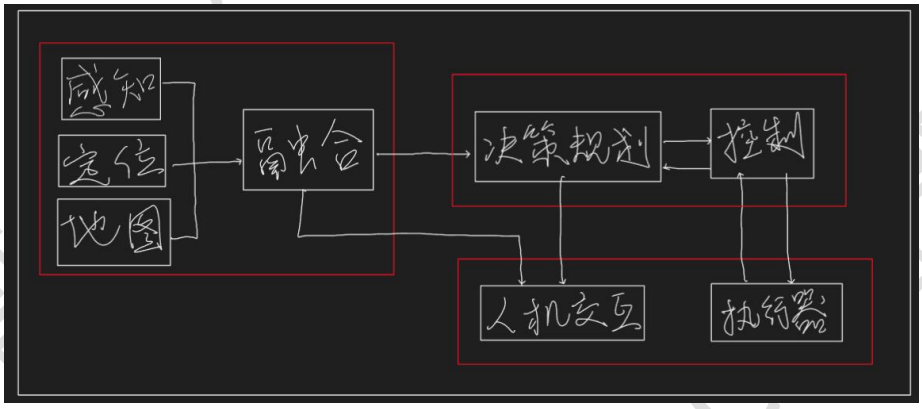
端到端的缺点：

- 可解释性弱，不易调试；
- 训练成本高，对数据量、数据的质量、算力、企业的资金和技术实力都有较高要求；

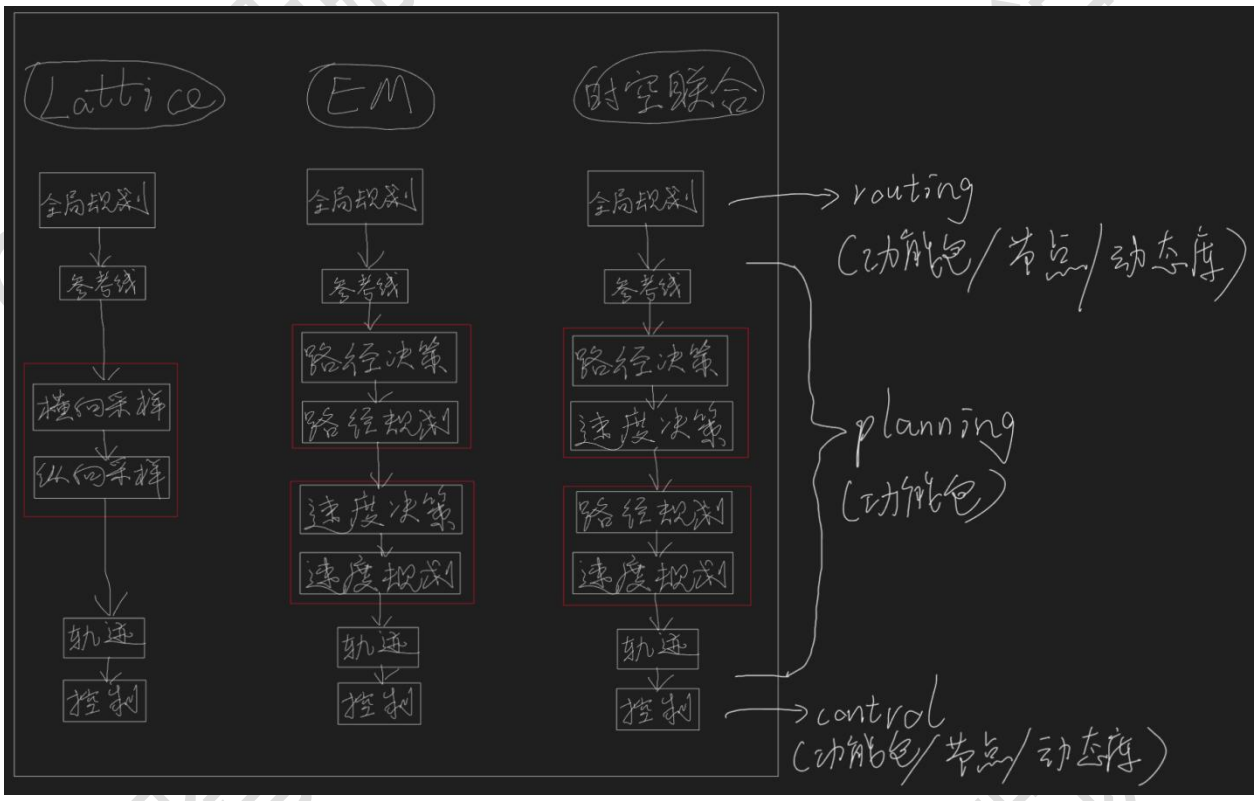
本课程所有的内容，都是基于规则的算法，目的是帮助广大从业人员强化基本功，培养良好的工程素养。至于以后我会不会出端到端的课，以后视情况而定。

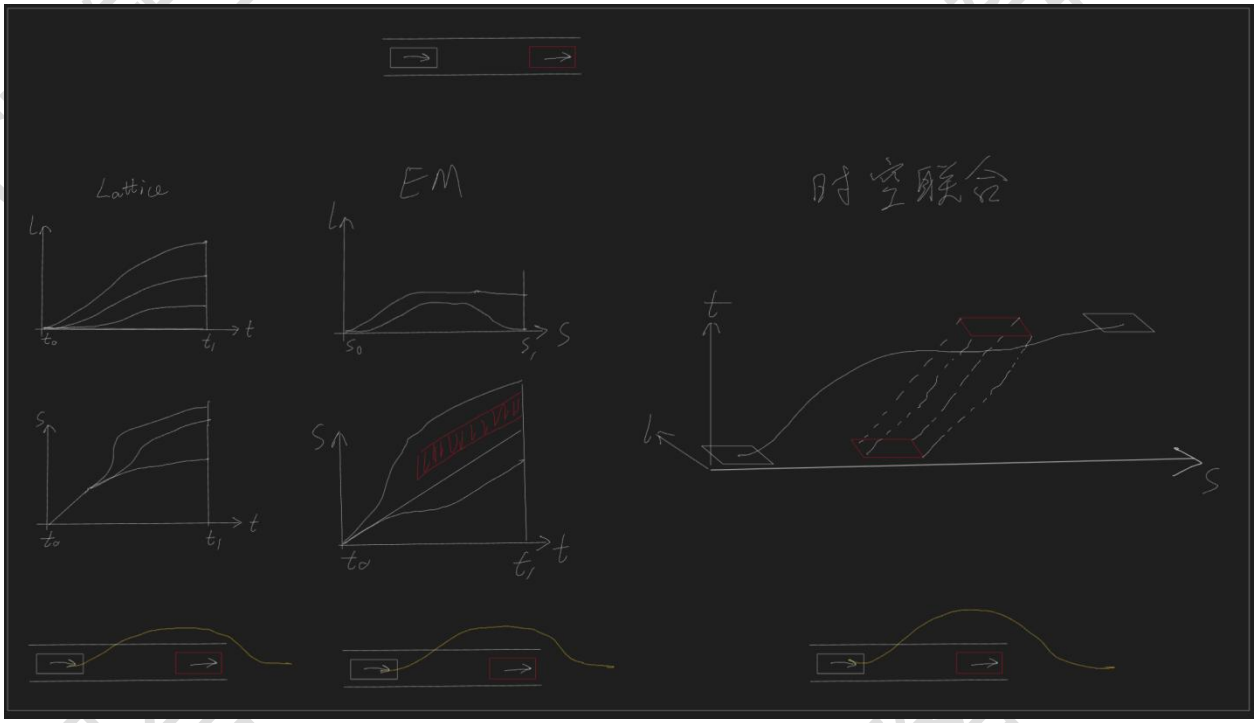


3.2，一般自动驾驶的软件架构



3.3，决策规划的几种常见代码架构





3.4，对决策规划的要求

- 功能：满足从起点到终点的行驶需求；
- 安全：无碰撞，遵守交通规则；

- 时效：车辆以尽可能短的时间或距离到达目的地；
- 稳定：输出的决策结果和轨迹连续、无跳变；
- 舒适：横纵向的平顺性；
- 可执行：满足车辆运动学和动力学约束；
- 性能：节省计算开销、运行流畅不卡顿；
- 客户的其他需求；

4，前置准备

4.1，自动驾驶基本概念

等级划分，传感器，软硬件架构，端到端，无图 ..... 仅需了解即可；

4.2，C++基础语法与数据结构

《黑马程序员 C++》；如没有相关基础，跟随本课程学即可，但建议先补齐相关理论；

4.3，Linux 基础指令

如没有相关基础，跟随本课程学即可；

4.4，ROS2 基础

赵虚左《ROS2 理论与实践》，鱼香 ROS《ROS 2 机器人开发从入门到实践》；
如没有相关基础，跟随本课程学即可；

4.5，CMake 基础

C 哥《vscode + CMake + git 实战系列》，如没有相关基础，跟随本课程学即可；

4.6，数学基础

矩阵基础，多项式曲线，二次规划，平面向量，曲线坐标系 .....
如没有相关基础，跟随本课程学即可；

4.7，电脑

CPU：越快越好
显卡：越快越好
内存：8g 及以上
硬盘：至少 20g 空间，固态
系统：ubuntu 22.04 及以上或虚拟机

二，工具链与环境搭建

1，工具链概述

1.1，为什么选择 ROS2

官网：ROS: Home

Gibhub：ROS 2 (github.com)

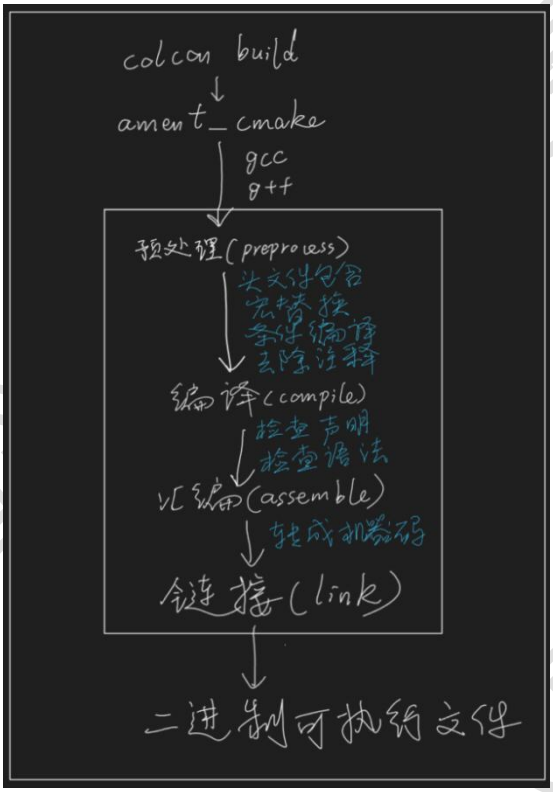
横向比较（相比于其他软件框架）：开源，免费，多平台支持，应用广泛，社区生态，快速部署 .....

纵向比较（相比于 ROS1）：去中心化，提升通信质量，对多机编队和小型嵌入式平台支持较好，更易于商业化落地，API 支持 C++ 11 及以上和 Python 3.5 以上，且 ROS1 已经停止支持 .....

1.2，Ubuntu 与 ROS2 版本对应关系

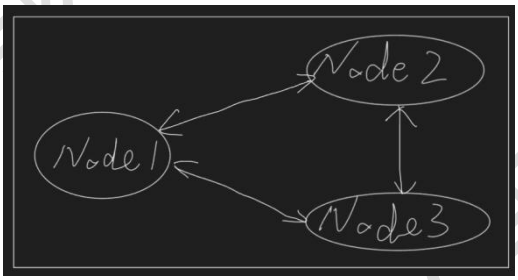
Ubuntu	ROS2	Release date	End of Life	
20.04 LTS	Foxy Fitzroy	2020 年 6 月	2023 年 5 月	ubuntu pro 可以延长支持周期
22.04 LTS	Humble Hawksbill	2022 年 5 月	2027 年 5 月	
24.04 LTS	Jazzy Jalisco	2024 年 5 月	2029 年 5 月	

1.3，基于 ROS2 的编译过程

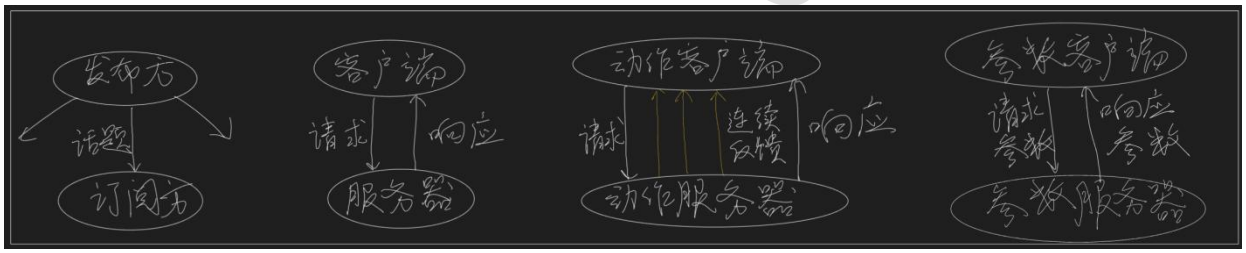


1.4，ROS2 的通信机制

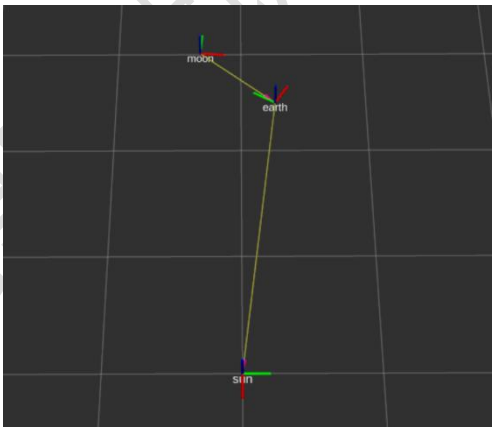
节点概念：

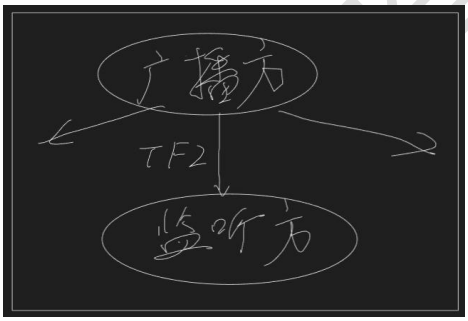


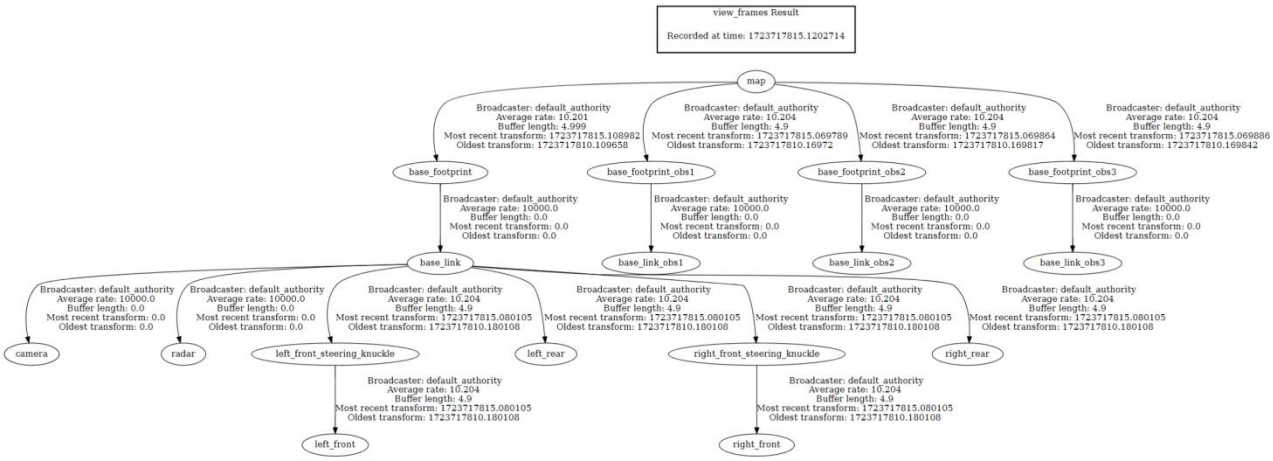
通信机制：



1.5，ROS2 的坐标系变换

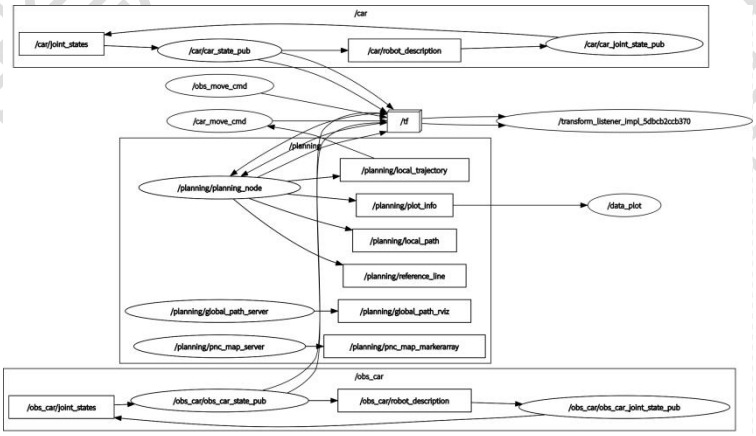




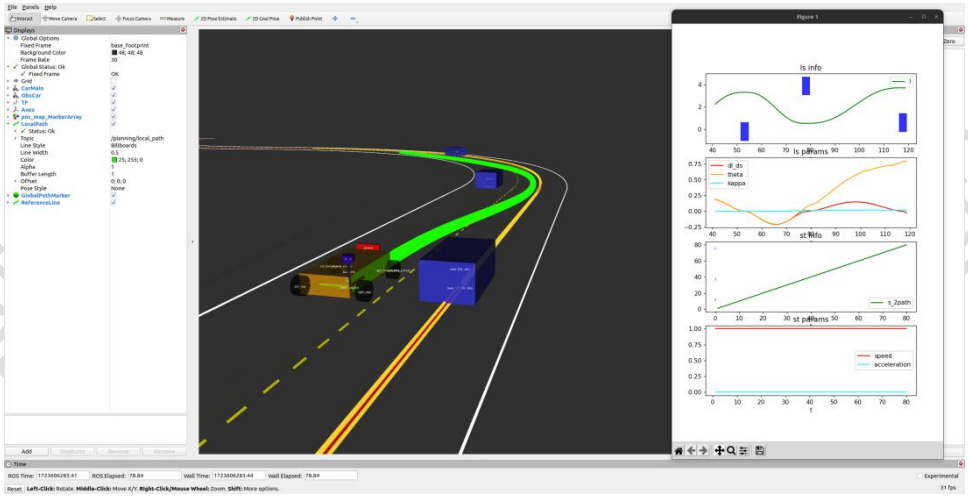


1.6，与 ROS2 相关的可视化与仿真工具

rqt：ROS2 自带的可视化工具，可以查看话题、参数、服务、动作等数据，以及节点与节点之间的关系；

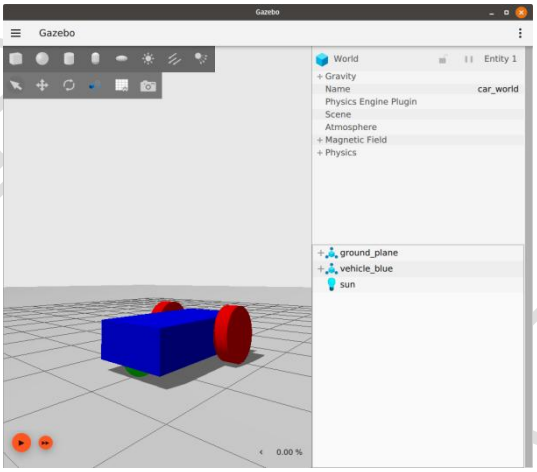


Rviz2：ROS2 自带的三维可视化平台，主要处理已有的数据，功能强大；



Gezebo：ROS2 中常用的三维物理仿真环境。不仅是一个显示工具，更是一个仿真平台。不需要预先存在的数据，而是能够创建并生成数据；

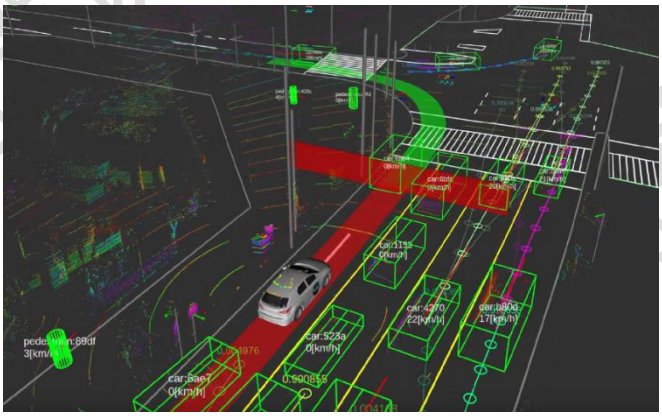
官网：[Gazebo (gazebosim.org)]



Autoware：基于 ROS（包括 ROS 2）开发的自动驾驶仿真软件；

官网：Home Page - Autoware

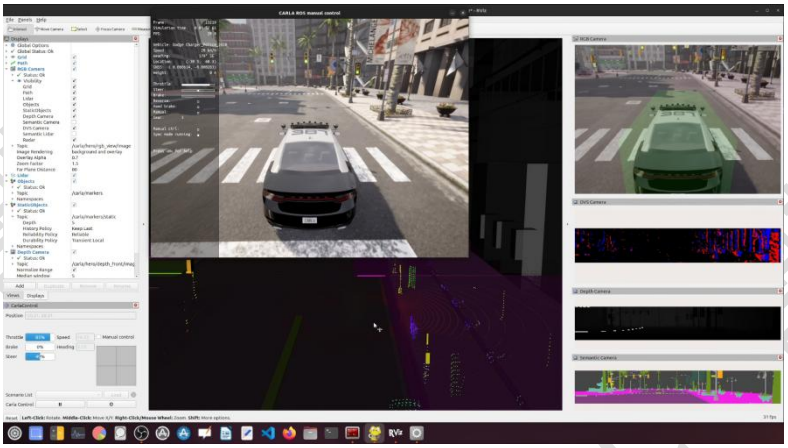
Github：The Autoware Foundation (github.com)



Carla：通过 carla-ros-bridge 实现与 ROS2 的无缝连接；

官网：CARLA Simulator

Github：CARLA (github.com)



Apollo：CyberRT 是 Apollo 自动驾驶平台中的通信框架，它基于 ROS 进行了深度定制和优化；

官网：Apollo 开发者社区 (baidu.com)

Github：Apollo Auto (github.com)

B 站官方：Apollo 开发者社区个人主页-哔哩哔哩视频 (bilibili.com)

2，Ubuntu 的安装

官网：企业开源和 Linux | Ubuntu
Github：Ubuntu (github.com)
这部分网上有很多教程，因此这里不再赘述。但要注意：
安装版本：建议 22.04 或 24.04；
安装方式：

- 直接安装 Ubuntu；
- 单硬盘安装 Ubuntu 与 Windows/macOS 双系统；
- 双硬盘安装 Ubuntu 与 Windows/macOS 双系统；
- 虚拟机安装 Ubuntu；

查看 ubuntu 系统信息：

```
lsb_release -a
```

3，Vscode 的安装

官网：Visual Studio Code - Code Editing. Redefined
注意安装插件；

4，CMake 的安装

官网：CMake - Upgrade Your Software Build System
Ubuntu 中已经自带了 CMake，无需另行安装；
ROS2 中使用 ament_cmake，是基于 CMake 软件包构建的系统，是 CMake 的强化版；
如想安装最新版的 CMake，可以参考我以前的视频；
查看 CMake 版本：

```
cmake --version
```

5，ROS2 的安装

5.1，设置语言环境

先检查本地语言环境是否支持 UTF-8 编码（大多数情况下是支持的）：

```
locale
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale
```

5.2，添加存储库

确保已启用 Ubuntu Universe 存储库；

5.3，添加软件源

```
sudo apt update
sudo apt install curl -y

sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg

(\是换行符，直接复制整条命令即可)
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] \
https://mirrors.aliyun.com/ros2/ubuntu $(. /etc/os-release && echo $UBUNTU_CODENAME) main" | \
sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

5.4，安装 ROS2

```
sudo apt update
sudo apt upgrade
sudo apt install ros-humble-desktop
```

5.5，配置环境

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

5.6，测试 ROS2

```
ros2 run turtlesim turtlesim_node
另启终端，
ros2 run turtlesim turtle_teleop_key
```

6，terminator 的安装

```
sudo apt install terminator
```

7，colcon 工具的安装

```
sudo apt install python3 && sudo apt install python3-colcon-common-extensions
```

8，tf 工具的安装

```
sudo apt install ros-humble-tf2-tools ros-humble-tf-transformations
sudo apt install python3-pip
pip3 install transforms3d
```

9，joint-state-publisher 工具的安装

```
sudo apt install ros-humble-joint-state-publisher
sudo apt install ros-humble-joint-state-publisher-gui
```

10，urdf 工具的安装

```
sudo apt install ros-humble-xacro
```

11，三方库的安装

10.1，Eigen

官网：https://eigen.tuxfamily.org
矩阵运算库，Ubuntu 自带，如不是最新的 3.4.0 版，建议安装 3.4.0 版：
解压，把文件夹移到/usr/local/include/目录下：

```
sudo mv eigen-3.4.0/ /usr/local/include/
```

10.2，osqp

官网：https://osqp.org/
Github：osqp/osqp: The Operator Splitting QP Solver (github.com)
二次规划求解库，需要依赖 Eigen 库，但不支持 C++；
如要安装 osqp-eigen，osqp 需要源码安装：

```
git clone https://github.com/osqp/osqp.git
cd osqp && mkdir build && cd build
cmake ..
cmake .. -DCMAKE_INSTALL_PREFIT=usr/local/osqp
sudo make install
source ~/.bashrc
```

10.3，osqp-eigen

官网：osqp-eigen (robotology.github.io)
Github：robotology/osqp-eigen: Simple Eigen-C++ wrapper for OSQP library (github.com)
osqp 库的 C++接口，需要依赖 osqp 和 Eigen；

```
git clone https://github.com/robotology/osqp-eigen.git
cd osqp-eigen && mkdir build && cd build
cmake ..
cmake .. -DCMAKE_INSTALL_PREFIT=usr/local/osqp-eigen
sudo make install
source ~/.bashrc
```

10.4，yaml-cpp

Github：jbeder/yaml-cpp: A YAML parser and emitter in C++ (github.com)
用 C++读取 yaml 格式配置文件；

```
sudo apt update
sudo apt install libyaml-cpp-dev
```

10.5，matplotlib

官网：Matplotlib — Visualization with Python
python 环境下的绘图库；（matplot++，matplotlib-cpp 经测试不如 matplotlib 好用）

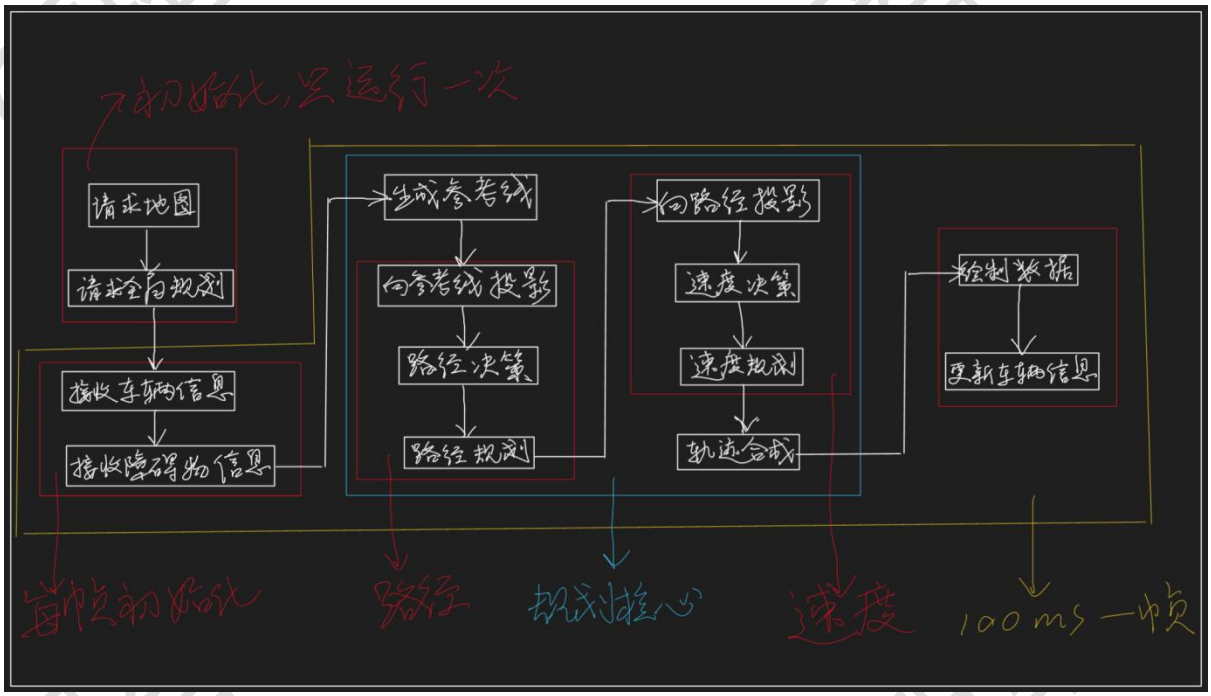
```
sudo apt update
sudo apt install python3-matplotlib

或者：
sudo apt install python3-pip （如果没有安装 pip）
pip3 install matplotlib
```

12，ROS2 命令行操作常用指令

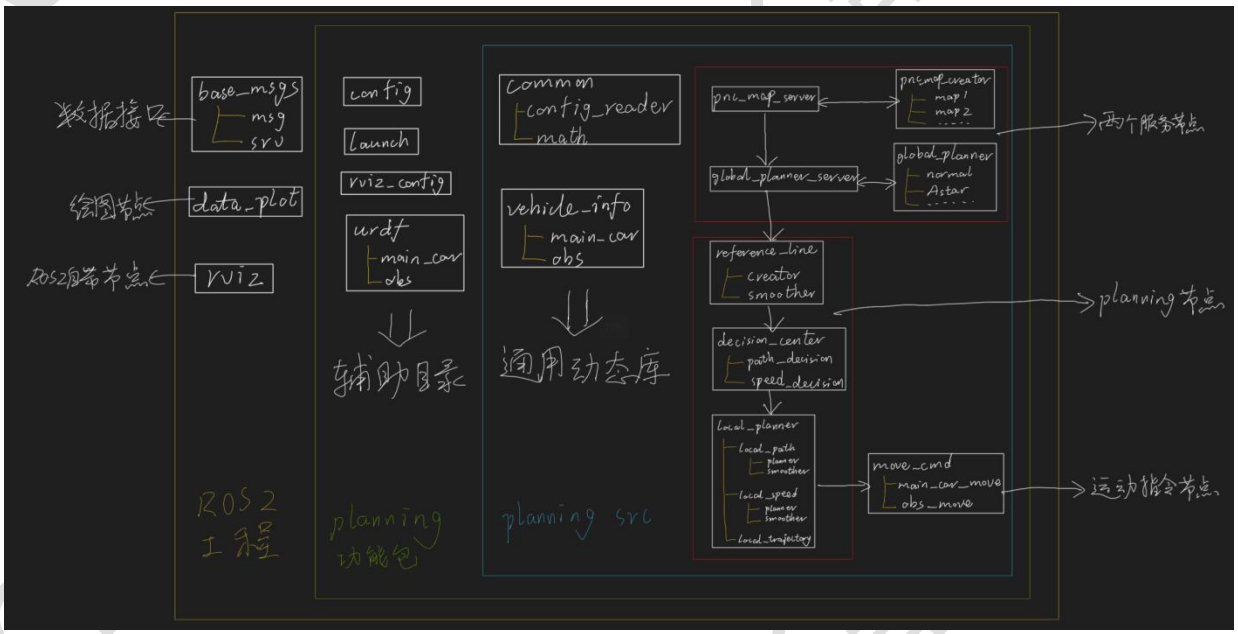
ros2 pkg list ros2 node list ros2 node info ...	ros2 topic list ros2 topic info ... ros2 topic type ... ros2 topic find ... ros2 service list ros2 service type ... ros2 service find ...	ros2 interface list ros2 interface package ... ros2 interface show ... ros2 interface proto ...

三，项目架构设计与搭建

1，程序流程设计



2，架构设计



3，代码编写

创建工程：

```
mkdir -p planning_with_ROS2_course/src
cd planning_with_ROS2_course
colcon build
cd src
```

创建功能包：

```
ros2 pkg create --build-type ament_cmake base_msgs
ros2 pkg create planning --build-type ament_cmake --node-name planning_node --dependencies rclcpp
ros2 pkg create data_plot --build-type ament_python --node-name data_plot --dependencies rclpy
```

查看目录树：

```
sudo apt install tree
tree
```

或者 tree -d

编译：

```
cd ..
colcon build
```

用 vscode 打开：

```
code .
```

环境配置：

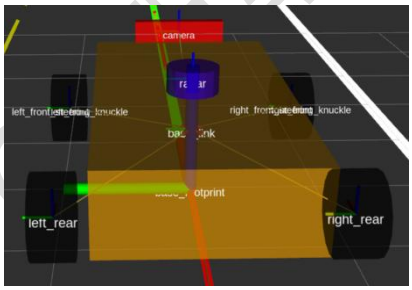
```
# for ros2 planning course project
export ROS2_PLANNING_COURSE_INSTALL=/home/codspielen/project/ROS2/planning_with_ROS2_course/install
source $ROS2_PLANNING_COURSE_INSTALL/setup.bash
export LD_LIBRARY_PATH=$ROS2_PLANNING_COURSE_INSTALL/planning/lib:$LD_LIBRARY_PATH
```

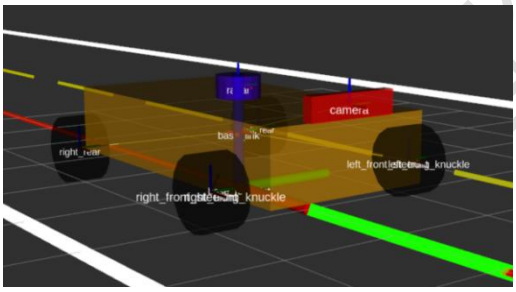
4，配置文件的实现

配置文件可以在不更改代码的情况下，快速更改一些参数，来提高调试程序的灵活性和方便性，可以通过不同的配置文件去快速实现不同的场景；
配置文件可以有多种格式，可以是 txt、yaml、xml、conf 或 ini 等格式，根据项目需求选择；
配置文件的实现可以有多种方式，如 C++中直接读取，或在 launch 文件中设置并读取，或利用 ROS2 中的参数通信方式实现；
本项目采用 C++中读取 yaml 格式的方式，在车辆信息、PNC 地图、全局路径、参考线、局部路径、速度、决策中心、总流程、运动指令每个模块中分别读取对应的参数；

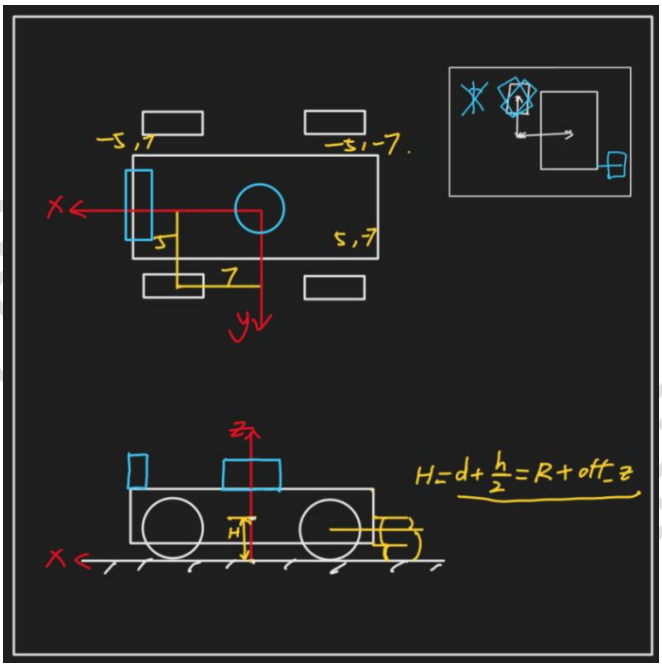
5，车辆模型的实现

5.1，车辆模型效果



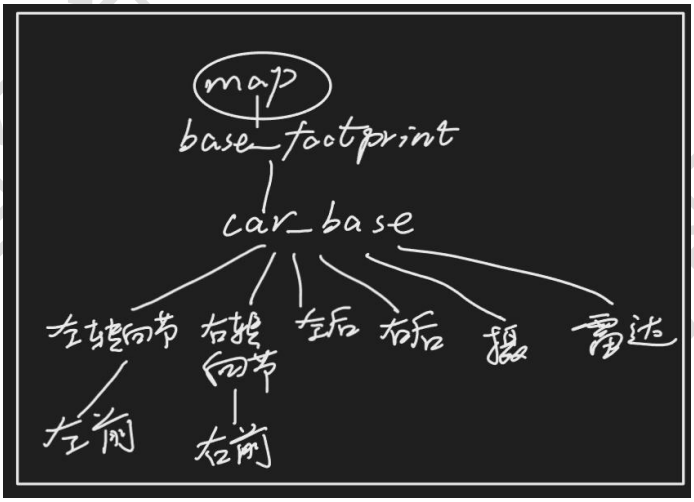


5.2，车辆模型的搭建原理



几何关系：
车身长 l、宽 w、高 h；
车身离地距离 H；
车身底部离地间隙 d；
车轮中心（或转向节）在 x、y、z 轴方向关于车身中心的偏距 off_x、off_y、off_z；
车轮宽度；
车轮半径 R；
前轮转角的约束；
雷达的半径和高度；
雷达在 z 轴方向的偏距（让雷达询问与车身顶部贴合）；
摄像头的长、宽、高；
摄像头在 x、y、z 轴方向的偏距；

坐标关系：



5.3，车辆模型的代码实现

6，luanch 文件的实现

launch 文件可以实现多节点启动，方便管理多节点的工程；
launch 文件可以实现参数的配置与传递；
python 编写的 launch 文件可以启动和停止不同的节点，也可以触发和作用于各种事件；

planning_launch.py 设置配置文件路径； 设置参数（如有需要）； 封装指令； 启动 rviz2 节点； 启动车辆模型分组： 　　state_publisher 节点； 　　joint_state_publisher 节点； 启动 planning 分组： 　　pnc_map_server 节点； 　　global_path_server 节点； 　　planning_node 节点； 启动 planning_launch.py： ros2 launch planning planning_launch.py	move_cmd_launch.py 启动 car_move_cmd 节点； 启动 obs_move_cmd 节点； 启动 move_cmd_launch.py： ros2 launch planning move_cmd_launch.py

7， 本章小节

介绍了程序流程的设计和架构的设计；
完成了基于 ROS2 的工程项目框架的编写；
完成了配置文件的编写；
完成了 launch 文件的编写；

四，数据接口定义

1，数据接口概述

数据接口一般也叫消息（msg），是 ROS2 跨节点通信的桥梁；
跨节点通信，包括了跨功能包、跨机器、跨平台、跨编程语言通信等情况，在很多通信系统中是刚需；
数据接口是一种简单的类似结构体的数据结构，不受编程语言限制；
数据接口包括话题、服务、动作三种类型，文件名后缀分别是.msg，.srv，.action；
ROS2 自带了很多种数据接口，可以满足一部分通信需求；
当 ROS2 自带数据接口满足不了需求时，需要自定义数据接口；

2，ROS2 自带的数据接口

2.1，ROS2 原始数据类型

9 大类原始数据类型，可以组成其他各种数据类型：

bool	整型范围
byte	int8: [-128 - 127]
char	int16: [-32768 - 32767]
string	int32: [-2147483648 - 2147483647]
float32, float64	int64: [-9223372036854775808 - 9223372036854775807]
int8, uint8	
int16, uint16	无符号整型范围
int32, uint32	uint8: [0 - 255]
int64, uint64	uint16: [0 - 65535]
	uint32: [0 - 4294967295]
	uint64: [0 - 18446744073709551615]

2.2，std_msgs

提供了很多基本消息类型，包括 Bool，Byte，Char，String，Float64，Int64，Uint64，Header 等消息类型；

- Header：表示消息头部，包含时间戳和坐标系信息，用于表明数据相对于哪个坐标系；

builtin_interfaces/Time stamp
string frame_id

2.3，geometry_msgs

用于表示机器人系统中几何形状和变换，包括 Point，PointStamped，Vector3，Vector3Stamped，Quaternion，Pose，Pose_stamped，Twist，Transform，TransformStamped 等消息类型；

- Point：表示三维空间中的一个点，包含 x、y、z 三个坐标值；

```
float64 x
float64 y
float64 z
```

- Vector3：向量，通常用于表示方向和速度，也包含 x、y、z 三个分量；

```
float64 x
float64 y
float64 z
```

- Quaternion：表示空间中的旋转，使用四元数表示，包含 x、y、z、w 四个参数；

```
float64 x    0
float64 y    0
float64 z    0
float64 w    1
```

- Pose：结合了 Point 和 Quaternion，用于表示空间中的一个位置和方向；

```
Point position
Quaternion orientation
```

- PointStamped，Vector3Stamped，PoseStamped：这些消息类型在相应的几何类型基础上增加了一个 Header；

std_msgs/Header header Point point	std_msgs/Header header Vector3 vector	std_msgs/Header header Pose pose

- Twist：描述线速度和角速度，包含 linear（线速度）和 angular（角速度）；

```
Vector3 linear
Vector3 angular
```

- Transform：用于描述两个坐标系之间的变换，包括平移（Vector3）和旋转（Quaternion）；

```
Vector3 translation
Quaternion rotation
```

- TransformStamped：在 Transform 的基础上增加了 Header 和子坐标系，子坐标和当前坐标组成了坐标变换的对象；

```
std_msgs/Header header
string child_frame_id
Transform transform
```

2.4，nav_msgs

用于机器人的导航和路径规划，包括话题，服务，动作三类消息，其中话题消息包括 Odometry，Path，MapMetaData，OccupancyGrid 等消息类型；

- Odometry（里程计信息）：用于表示机器人在三维空间中的位姿（位置和方向）以及线速度和角速度，包含 Header，child_frame_id（子坐标系），pose（位姿），twist（线速度和角速度）；

```
std_msgs/Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

- Path（路径）：用于表示一系列连续的位姿点，这些点共同构成了一条机器人可以跟踪的路径，包含 Header，poses（包含多个 geometry_msgs/PoseStamped 的数组，每个元素代表路径上的一个点，包括该点的位置和方向）；

```
std_msgs/Header header
geometry_msgs/PoseStamped[] poses
```

- MapMetaData（地图元数据）：提供关于栅格地图的基本信息，如地图的分辨率、尺寸、加载时间等，包含 map_load_time（地图加载的时间），resolution（地图的分辨率，以米/像素为单位），width、height（地图的宽度和高度，以像素为单位），origin（地图的原点位置，包括位置和方向）；

```
time map_load_time
float32 resolution
uint32 width
uint32 height
geometry_msgs/Pose origin
```

- OccupancyGrid（栅格地图）：用于表示环境的栅格地图，其中每个栅格都有一个被占据的概率值，包含 Header，info（MapMetaData 类型，包含地图的元数据），data（一个一维数组，表示栅格地图中每个栅格的占据概率）；

```
std_msgs/Header header
MapMetaData info
int8[] data
```

2.5，visualization_msgs

提供了一组专门用于可视化目的的消息类型，允许 ROS 节点创建和发布可视化数据，这些数据可以被 Rviz 接收并显示；

- Marker：用于在三维空间中创建单个标记，包含颜色、大小、形状、位置、方向等属性；

```
uint8 ARROW=0
uint8 CUBE=1
uint8 SPHERE=2
uint8 CYLINDER=3
uint8 LINE_STRIP=4
uint8 LINE_LIST=5
uint8 CUBE_LIST=6
uint8 SPHERE_LIST=7
uint8 POINTS=8
uint8 TEXT_VIEW_FACING=9
uint8 MESH_RESOURCE=10
uint8 TRIANGLE_LIST=11
uint8 ADD=0
uint8 MODIFY=0
uint8 DELETE=2
uint8 DELETEALL=3
std_msgs/Header header
string ns
int32 id
int32 type
int32 action
geometry_msgs/Pose pose
geometry_msgs/Vector3 scale
std_msgs/ColorRGBA color
duration lifetime
bool frame_locked
geometry_msgs/Point[] points
std_msgs/ColorRGBA[] colors
string text
string mesh_resource
bool mesh_use_embedded_materials
```

- MarkerArray：用于发布一系列 Marker 消息，允许同时在三维空间中显示多个标记；

```
visualization_msgs/Marker[] markers
```

3，自定义话题通信接口

3.1，PNCMap

PNC 地图：规划和控制模块用的地图，从高精地图或其他类型的地图提取了对规划模块有用的信息，承上启下，同时也提供用于 rviz 显示的信息；
虽然叫地图，但和规划模块关系更紧密，因此放在规划模块中；
没有统一的形式，根据项目需求定义，本项目中是一种比较简略的形式；

```
std_msgs/Header header                              # 消息头
float64 road_length                                 # 道路长度
float64 road_half_width                             # 车道半宽
visualization_msgs/Marker left_boundary             # 左边界
visualization_msgs/Marker right_boundary            # 右边界
visualization_msgs/Marker midline                   # 中心线
```

3.2，ReferlinePoint 和 Referline

参考线的点，组成参考线；

ReferlinePoint：

```
geometry_msgs/PoseStamped pose    # 位置和航向
float64 rs                        # 作为投影点的 rs
float64 rtheta                    # 作为投影点的航向角
float64 rkappa                    # 作为投影点的曲率
float64 rdkappa                   # 作为投影点的曲率变化率
```

Referline：

```
std_msgs/Header header            # 消息头, 用于坐标系和时间同步
ReferlinePoint[] refer_line       # 参考线
```

3.3，LocalPathPoint 和 LocalPath

局部路径的点，组成局部路径；

LocalPathPoint：

```
geometry_msgs/PoseStamped pose    # 位置和航向
float64 s                         # s
float64 l                         # l
float64 ds_dt                     # ds/dt
float64 dl_ds                     # l'
float64 dl_dt                     # dl_dt
float64 dds_dt                    # dds/dt
float64 ddl_ds                    # l''
float64 ddl_dt                    # ddl_dt
float64 theta                     # 航向角
float64 kappa                     # 曲率
float64 dkappa                    # 曲率变化率
float64 rs                        # 作为投影点的 rs
float64 rtheta                    # 作为投影点的航向角
float64 rkappa                    # 作为投影点的曲率
float64 rdkappa                   # 作为投影点的曲率变化率
```

LocalPath：

```
std_msgs/Header header              # 消息头, 用于坐标系和时间同步
LocalPathPoint[] local_path         # 局部路径
```

3.4，LocalSpeedsPoint 和 LocalSpeeds

速度点，组成速度结果（加 Local 表示属于局部规划的范围，命名更统一）；
LocalSpeedsPoint：

```
float64 t                           # t
float64 s_2path                     # s_2path
float64 ds_dt_2path                 # ds_dt_2path
float64 dds_dt_2path                # dds_dt_2path
float64 speed                       # 速度
float64 acceleration                # 加速度
float64 dacceleration               # 加加速度
```

LocalSpeeds：

```
std_msgs/Header header              # 消息头, 用于坐标系和时间同步
LocalSpeedsPoint[] local_speeds     # 速度结果
```

3.5，LocalTrajectoryPoint 和 LocalTrajectory

轨迹点，组成最终轨迹（加 Local 表示属于局部规划的范围，命名更统一）；
LocalTrajectoryPoint：

```
LocalPathPoint path_point       # 路径点
LocalSpeedsPoint speed_point    # 速度点
```

LocalTrajectory：

```
std_msgs/Header header                  # 消息头, 用于坐标系和时间同步
LocalTrajectoryPoint[] local_trajectory # 轨迹
```

3.6，ObsInfo

障碍物信息；

```
float64 obs_length      # 长
float64 obs_width       # 宽
float64 l               # 参考线投影的 l
float64 s               # 参考线投影的 s
float64 s_2path         # 路径投影的 s
float64 ds_dt_2path     # 路径投影的速度(就是笛卡尔坐标下的速度)
float64 t_in            # 进入路径时间
float64 t_out           # 退出路径时间
```

3.7，PlotInfo

绘图信息；

```
std_msgs/Header header              # 消息头，用于坐标系和时间同步
LocalTrajectory trajectory_info     # 轨迹信息
ObsInfo[] obs_info                  # 障碍物信息
```

4，自定义服务通信接口

4.1，PNCMapService

地图服务：

```
Int8 map_type    # 请求：地图类型
---
PNCMap pnc_map    # 响应：地图
```

4.2，GlobalPathService

全局路径规划服务：

```
Int8 global_planner_type       # 请求 1：全局规划类型
PNCMap pnc_map                 # 请求 2：地图
---
nav_msgs/Path global_path    # 响应：全局路径
```

五，PNC 地图与全局规划

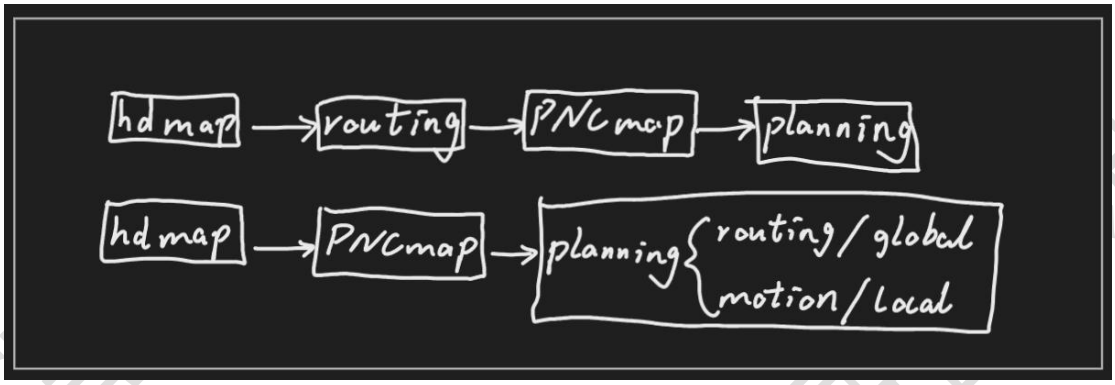
1，PNC 地图

1.1，常见地图类型

高精度地图，导航地图，拓扑地图，点云地图，栅格地图，语义地图，PNC 地图；

1.2，作用

规划和控制模块用的地图，从高精地图或其他类型的地图提取了对规划模块有用的信息，承上启下，同时也提供用于 rviz 显示的信息；
虽然叫地图，但和规划模块关系更紧密，因此放在规划模块中；
没有统一的形式，根据项目需求定义，本项目中是一种比较简略的形式；



1.3，流程

- planning 节点向地图服务器请求地图；
- 地图服务器调用地图生成函数（动态库），生成地图；
- 地图服务器作出响应，把地图发回给 planning 节点；

2，全局规划

2.1，作用

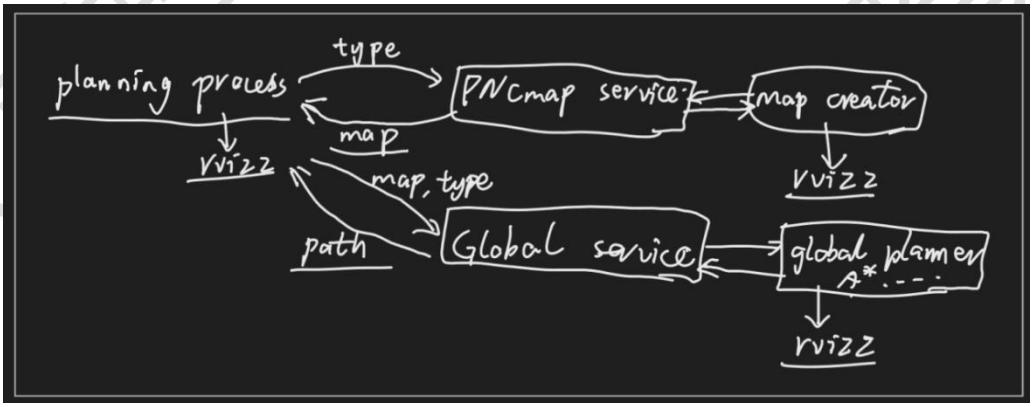
导航，生成从 A 点到 B 点的初始路径；

2.2，算法

A*，D*，广度优先，Dijkstra，动态规划等；
本项目重点是局部路径规划，不涉及全局规划，所以简化处理；
想学 A*算法的，可以去学我上一门课《讲透 A*算法》；

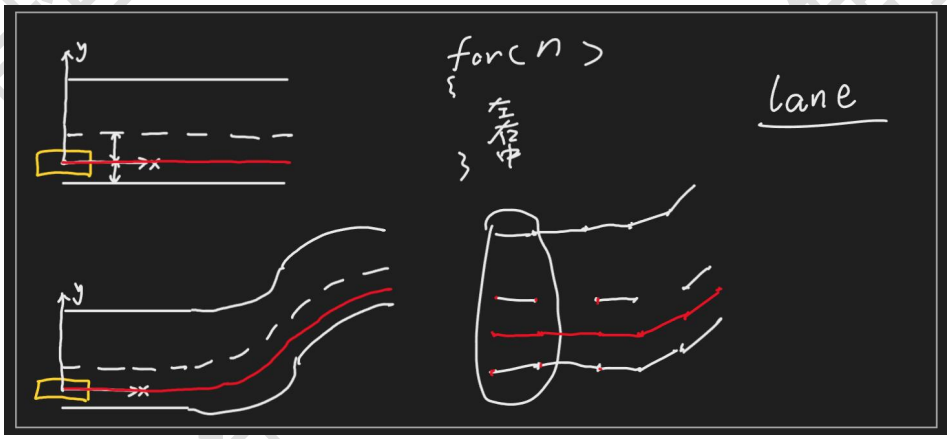
2.3，流程

- planning 节点向地图服务器请求全局路径；
- 地图服务器调用全局路径函数（动态库），生成全局路径；
- 地图服务器作出响应，把全局路径发回给 planning 节点；

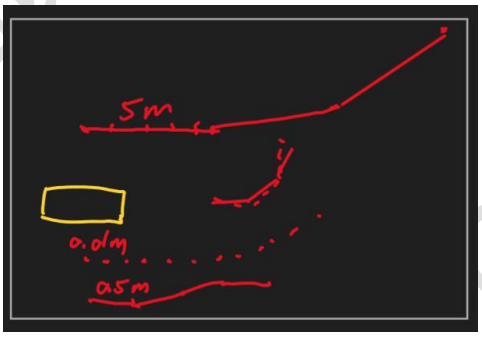


3，代码编写

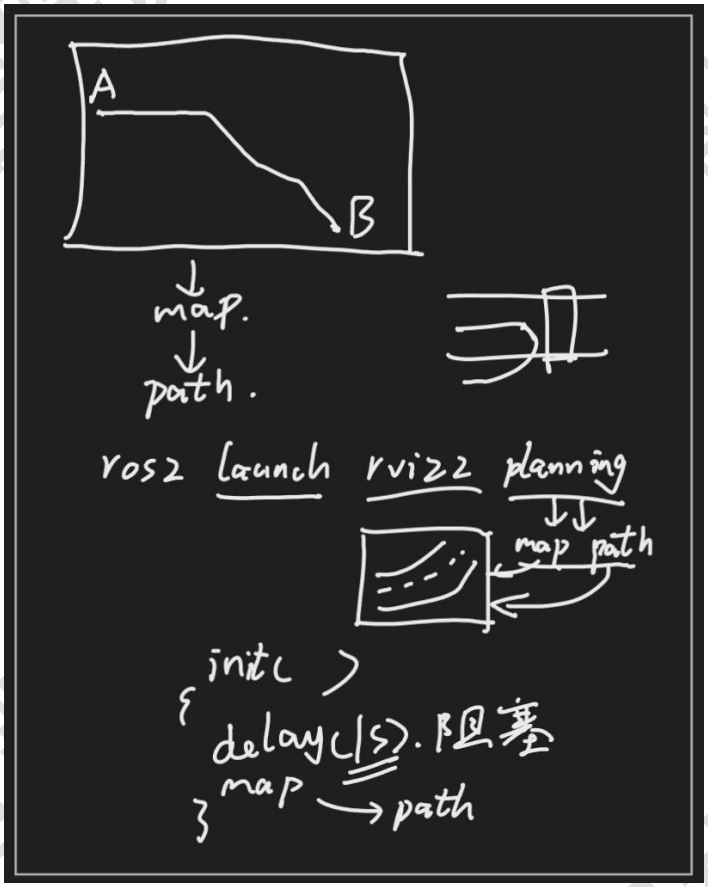
地图元素：左右边界+中心线，左右边界为实线，中心线为虚线，构成双车道的道路；
编程思路：使用 for 循环，一段一段地绘制，每段同时绘制左中右的小段；
地图位置与方向：右车道的中心起点与车身中心点重合，也就是让车辆初始位置位于右车道的中心起点，方向沿 x 轴延伸；
全局规划实现方式：直接用中心线和右边界的中心；



全局路径的密度：每小段 0.5m；



地图和全局规划运行的频率：初始化时运行 1 次，按需触发；
启动顺序控制：初始化时阻塞 1s，让 rviz2 先运行起来，才能接收到地图和全局路径并显示；



4，大作业 1：自行设计地图并用 A*算法实现全局路径规划

地图：拓扑地图，自定义的栅格地图，占据栅格地图；
要求：使用 A*算法，生成最短路径，并可视化显示；