

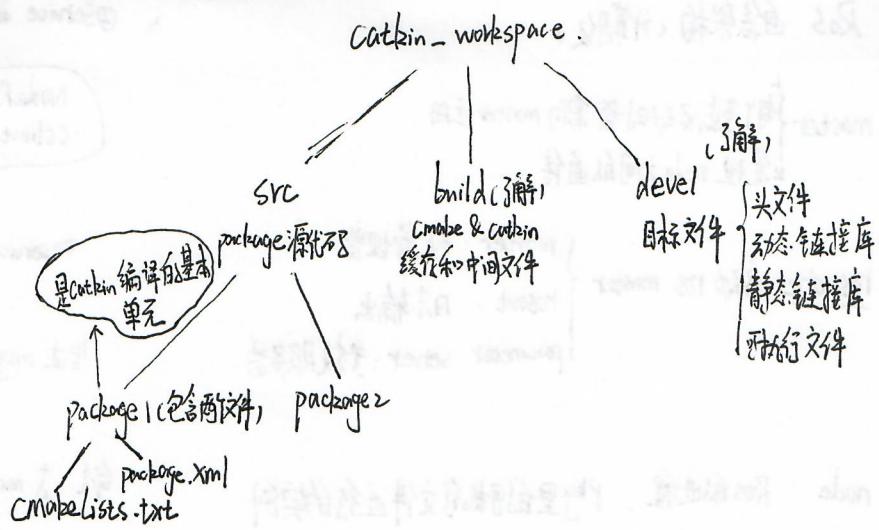
Ros.

ROS 官网 Ros.org., IDE. Roboware

1. ROS 工程系统(文件系统)

Catkin : ROS 定制的编译构建系统, 对 Cmake 扩展
用来编译程序。

Catkin 工作空间: 组织和管理功能包的文件夹



Catkin 工作空间的建立、使用命令 Catkin-make.

(和使用 git init 相同, 先建立一个文件夹, 再使用 catkin-make)

建立工作空间 | mdir -P ~ /catkin-ws /src
| cd → —
| catkin-make.

编译 | catkin-make
| Source ~ /catkin-ws /devel /stmp-bash
| 使用 source 刷新环境, 让系统可联系到执行文件

package : (Ros 软件的基本组织形式) | 最终包含 CMakeList.txt
| (Catkin 编译的基本单元) | package.xml
| -> package 包含多个可执行文件(节点)

CMakeList.txt 规定 catkin 编译的规则, 例如: 源文件、依赖项、目标文件

package.xml 定义 package 的属性, 例如: 包名、版本号、作者、依赖.

package 中的代码文件 | 脚本 cshell .python,
| C++ 头文件 .h 文件

package 中的代码文件 | scripts < .sh
| include - .h
| src - .cpp

package 可放其它文件 | 消息 (msg), msg - .msg
| 服务 (srv), srv - .srv
| 3. 作 (action), action - .action

package . { launch 文件 launch - .launch
配置文件 (yaml) config - .yaml

CMakeLists.txt	编译规则文件
package.xml	定义 package 的属性
scripts	脚本文件 .sh / .py
include	C++ 头文件 .h
src	C++ 源文件 .cpp
msg	消息 .msg
srv	服务 .srv
action	动作 .action
launch	.launch
config	配置文件 .yaml

常用包管理操作:

1. ros pack. 查找 package.

rosdep find package-name
rosdep list

2. roscd. 跳转到某 Pkg 路径下

roscd package-name.

3. rosfs. 列某包 package 基础文件

rosfs package-name

4. rosed 编辑 Pkg 目录文件 (vi)

rosed package-name file-name

5. catkin-create-pkg. 创建一个 Pkg.

catkin-create-pkg < name > [deps] (rospp, rospy, std-msgs, nav-msgs,

Ros 通信架构(计算机级)

master 每个节点启动时都需向 master 注册。
管理 node 之间的通信。

roscore 启动 ros master
 master: 节点管理器
 rosout: 日志输出
 parameter server: 参数服务器

node Ros 的进程。Pkg 里白何执行文件运行的实例

rosmn 启动一个 node 例 | rosmn [pkg_name] [node_name]

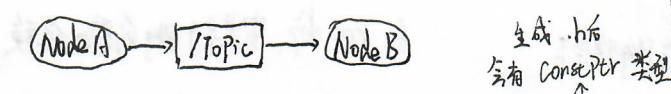
rosnode
 rosnode list 列出当前运行的 node 信息
 rosnode info [node_name] 显示某个 node 的详细信息
 rosnode kill [node_name] 结束某个 node

roslaunch
 启动 master 和若干 node
 roslaunch [pkg_name] [file_name.launch]
 *-bringup 启动规则

Ros 通信方式
 Topic
 Service
 parameter Service
 Actionlib

① Topic: Ros 中的异步通信方式

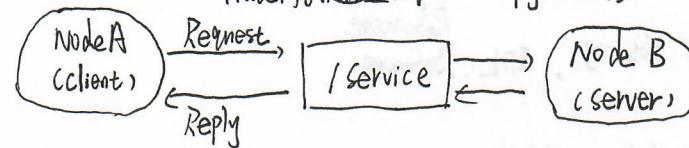
Node 间通过 publish - subscribe 机制通信。



Message: topic 内容的数据类型, 定义在 *.msg 文件中

命令: rostopic
 rostopic list 列出 topic
 rostopic info /topic-name 显示某个 topic 信息
 rostopic echo /topic-name 显示某个 topic 内容
 rostopic pub /topic-name ... 向某个 topic 发布内容
 type ... 看 topic 格式
rosmsg
 rosmsg list 列出系统上所有 msg
 rosmsg show /msg-name 显示某个 msg 内容

② Service 通信方式: { 同步通信方式
 Node 间可以通过 request - reply 方式通信



* service 通信格式, 定义在 *.srv 文件中

{ 生成.h 后会有
 Request
 Response

写过 msg、srv 后需修改 package.xml、CreateList.txt

命令, 与 ros topic 类似。

rosservice
 rosservice list 列出当前活跃的 service
 rosservice info service-name 显示某个 service 的信息
 rosservice call service-name args. 调用某个 service

rossrv
 rossrv list 列出所有 srv
 rossrv show srv-name 显示某个 srv 内容

③ parameter server { 有诸多参数的字典, 可用命令行、launch 文件 和 node API 读写

rosparam
 rosparam list 列出当前所有参数
 rosparam get param-key 显示某个参数的值
 rosparam set param-key param-value 设置某个参数的值
 rosparam dump file-name 保存参数到文件
 rosparam load file-name 从文件读取参数
 rosparam delete param-key 删除参数

参数保存在 yaml 中。

④ Action (带有反馈的 service), 通常用在长时间、可抢占的任务中



action 通信的数据格式, 定义在 *.action 文件中

Ros 常用工具

仿真: Gazebo ✓

调试、可视化: Rviz, rqt ✓

命令行工具: rostopic, rosbag ...

专用工具: Moveit

Rviz 可视化工具 通过 topic 通信.

rqt, 基于 Qt.

rqt-graph, rqt-plot, rqt-console. (用途)

显示通信架构, 绘制曲线, 查看日志. rosrun rqt-plot rqt-plot.

rosbag 记录和回放数据流 .bag 文件

<pre>rosbag record <topic-names> 记录某些 topic 到 bag</pre>	<pre>rosbag record -a 记录 all topic 到 bag</pre>
<pre>rosbag play <bag-file> 回放 bag</pre>	

ros::master (Namespace)

常用函数: bool check(), 检查 master 是否启动

const string & getHost(); 通过 master 从系统 host name

bool getNodes(...); 从 master 返回已知的 node 名称列表

bool getTopic(...); 返回所有正在被发布的 topic 列表

bool getURL(); 返回到 master 的 URL 地址, 如 http://host:port

uint32_t getPort(); 返回 master 运行的端口

ros::this_node (Namespace)

常用函数: void getAdvertisedTopics(...); 返回本 node 发布的 topic

const string & getName(); 返回当前 node 的名称

const string & getNamespace(); 返回当前 node 的命名空间

void getSubscribedTopics(); 返回当前 node 订阅的 topic

ros::service (Namespace)

bool call(...); 调用一个 PRC 服务

ServiceClient createClient(...); 创建一个服务的 service

bool exists(...); 确认服务可调用

bool waitForService(...); 等待一个服务直到它可调用

ros::names (Namespace)

string append(...); 追加名称

消除圆资源名称: 删去双斜线, 给尾斜线

返回重映射 remapping

对名称重映射.

解析出名称的全名.

验证名称

例: rosCPP, rosPy

C++ Python

rosCPP:

- ros::init(); 解析传入的 Ros 参数, 使用 rosCPP 第一步要用的函数, 为本 node 命名
- ros::NodeHandle 和 topic.service.param 等交互的会话接口, 为一个类
- ros::master 包含从 master 查询信息的函数.
- ros::this_node 包含查询这个进程(node)的函数
- ros::service 包含查询服务的函数
- ros::param 包含查询参数服务器的函数, 而不要解 NodeHandle
- ros::names 包含处理 ROS 圆资源名称的函数.

rosinit(), 为 node 命名

ros::NodeHandle class.

ros::Publisher advertise<类型>(string topic_name, size_t msg)

ros::Subscriber

ros::Subscriber subscribe(..., 创建话题的 subscribe)

ros::ServiceServer advertiseService(..., 创建服务的 server)

ros::ServiceClient serviceClient(..., 创建服务的 client)

bool getParam(..., 查询某个参数的值)

bool setParam(..., 给参数赋值)

第4个topic-demo 步骤

1. 创建一个工作空间 `mkdir -p catkin_ws/src`
2. `catkin_make`
3. 创建一个插件 `catkin_create_plugin <plugin-name>`
4. 写一个msg
5. 编写 `talker.cpp`, `listener.cpp`
6. 修改 `CMakeLists.txt`, `package.xml`
7. `catkin_make`
8. source `~/.catkin_ws/devel/setup.bash`

TF (Transform, 坐标变换)

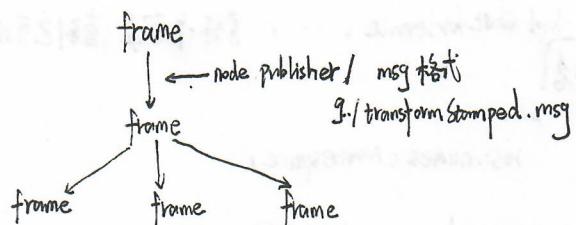
URDF (ROS制定的机器模型)

Transform 坐标变换(位置+姿态), 坐标系数据维护的工具

+ 坐标转换的标准, 话题, 工具, 接口

+ tf tree

/tf tree



+ tf 类型

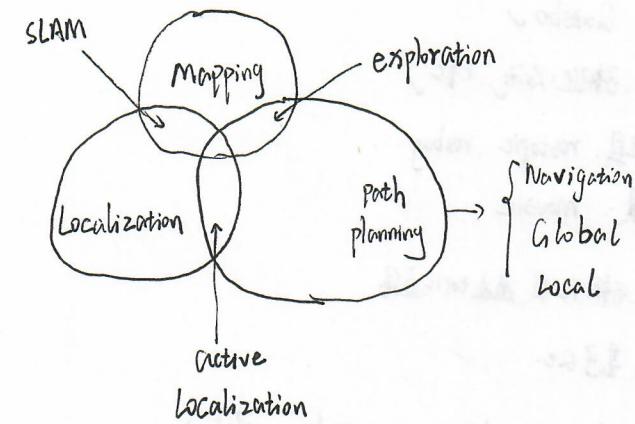
URDF (Unified Robot Description Format)

主要写URDF文件

启动 `Robot_sim_diana Camera`.

`rosrun image_view image_view image:=/camera/rgb/image_raw`

无人车运动和任务分配



Route Planning → Plug Navigation Stack, github ros-planning

{ amcl 自适应定位
map server 提供地图.

move-base { 全局规划, 局部规划, 处理异常行为.
静态 动态

costmap { Global-costmap { 静态信息
local-costmap { 动态信息
inflation layer 膨胀

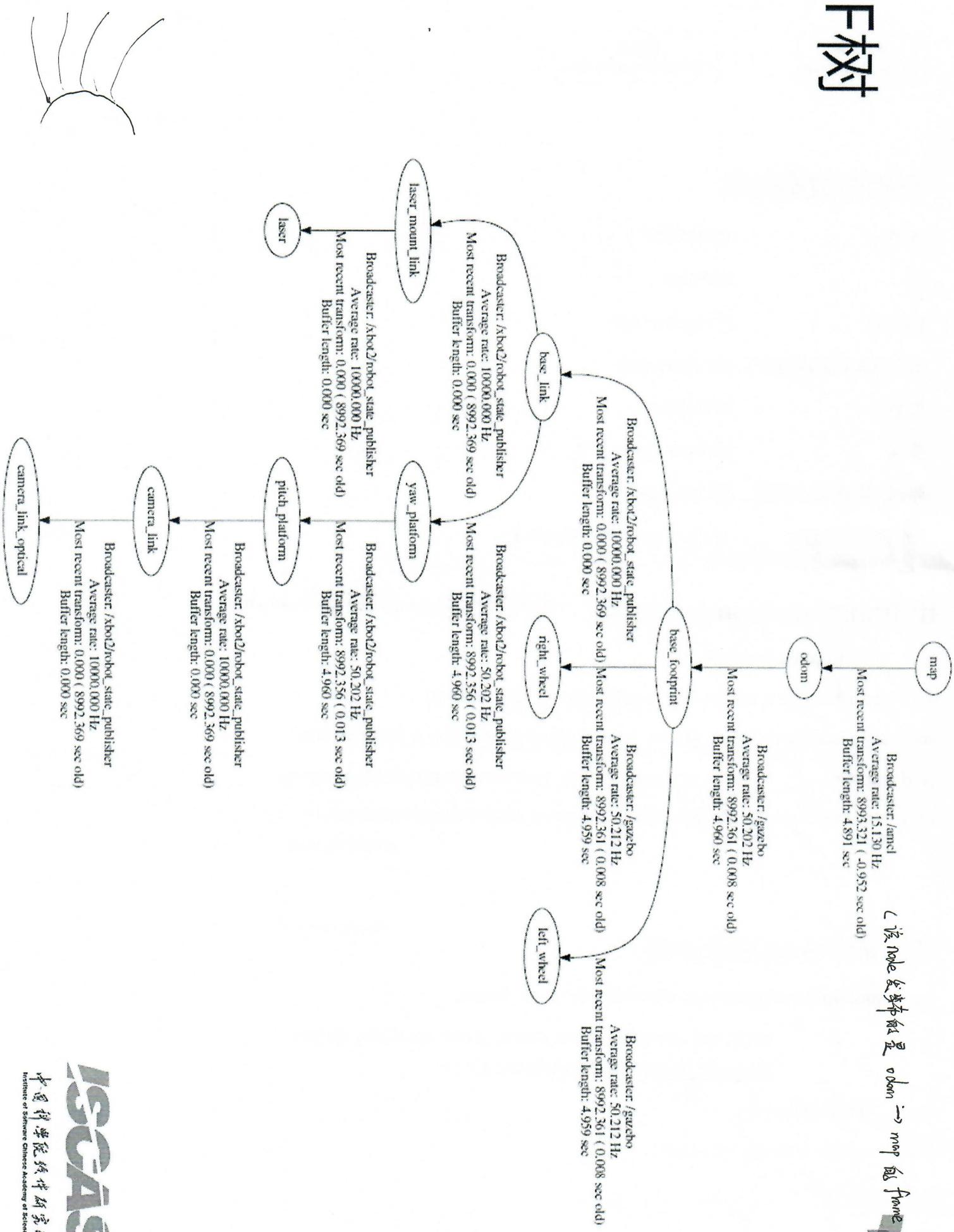
mapserver { roshub map-server map-server my-map.yaml
发布 map-server, 发布地图.
保存地图.

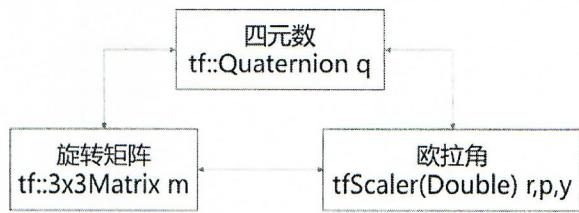
roshub map-server map-server [-f my-map]

rviz 使用教程. ros官网 百度 ros rviz. → ~~tutorials~~, tutorials

TF树

（该图展示了从odom到map的frame转换）





TF相关数据类型

向量	tf::Vector3
点	tf::Point
四元数	tf::Quaternion
3x3矩阵 (旋转矩阵)	tf::Matrix3x3
位姿	tf::Pose
变换	tf::Transform <small>类</small>
带时间戳的以上类型	tf::Stamped<T>
带时间戳的变换	tf::StampedTransform <small>类</small>

tf::TransformBroadcaster类 使用包含 <t>/transform_broadcaster.h >

```

TransformBroadcaster()
void sendTransform (const StampedTransform &transform)
void sendTransform (const std::vector< StampedTransform > &transforms)
void sendTransform (const geometry_msgs::TransformStamped &transform)
void sendTransform (const std::vector<geometry_msgs::TransformStamped>
&transforms)

```

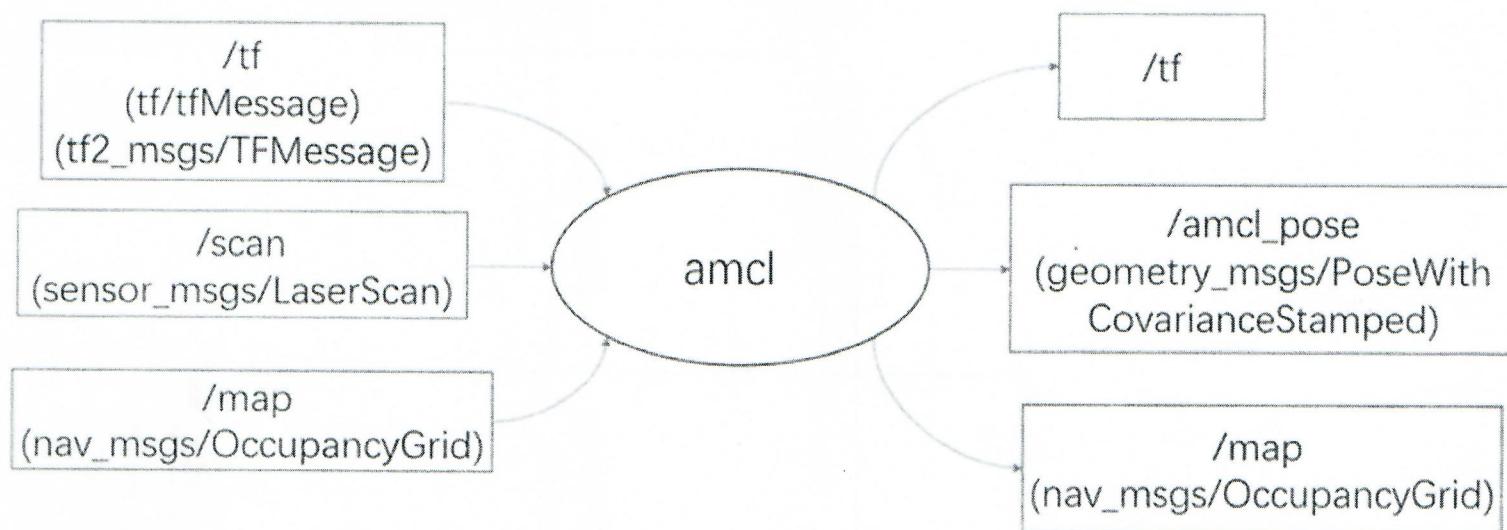
tf::TransformListener类

```

void lookupTransform(const std::string &target_frame,
                    const std::string &source_frame, const ros::Time &time,
                    StampedTransform &transform) const
bool canTransform() const
bool waitForTransform() const

```

AMCL

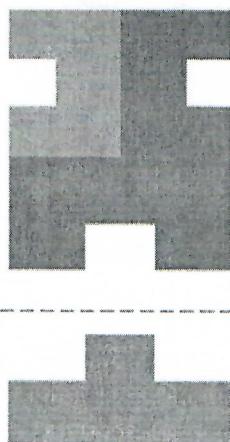


Base Local Planner

base_local_planner
dwa_local_planner



move_base



Base Global Planner

carrot_planner
navfn
global_planner

Recovery Behavior

clear_costmap_recovery
rotate_recovery
move_slow_and_clear

Navigation Stack Setup

"move_base_simple/goal"
geometry_msgs/PoseStamped

move_base

"/map"
nav_msgs/GetMap

base controller

global_planner

amcl

sensor transforms

tf/tfMessage
/tf

internal
nav_msgs/Path

recovery_behaviors

sensor_topics
sensor_msgs/LaserScan
sensor_msgs/PointCloud

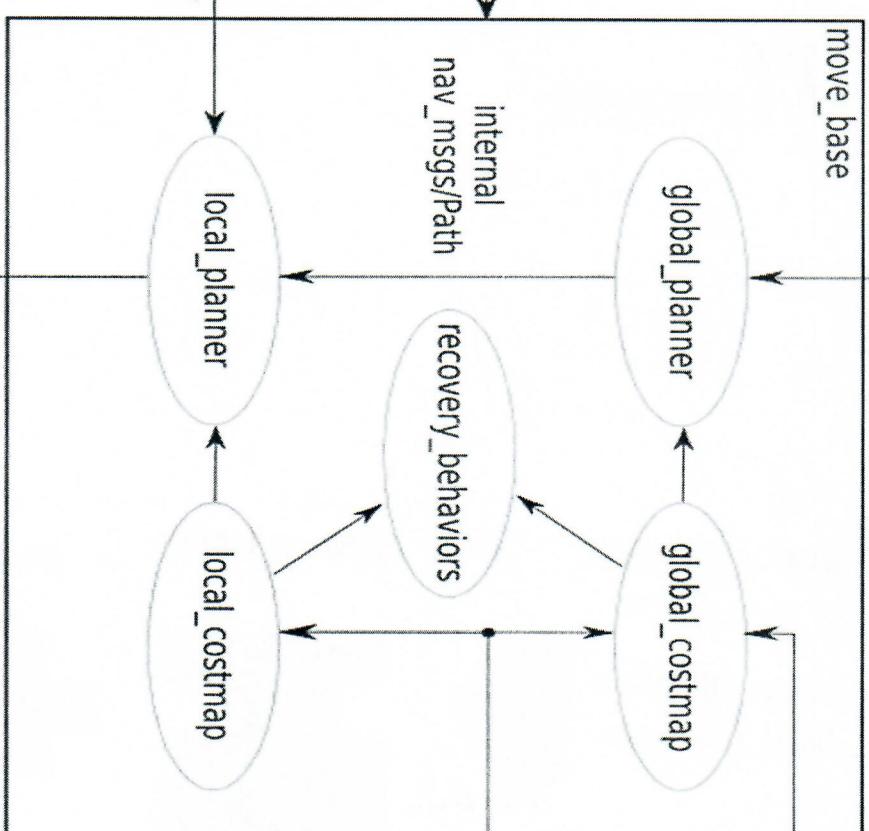
sensor sources

odometry source

"odom"
nav_msgs/Odometry

local_planner

local_costmap



- provided node
- optional provided node
- platform specific node

论文摘记：

路径规划 $\left\{ \begin{array}{l} \text{全局 (静态)} \\ \text{局部 (动态)} \end{array} \right.$

Route - Planning
trajectory - planning.
vehicle Routing Problem

path planning.

几种传统方法：

- ① 人工势场法：模拟引力分布，物体与目标点存在引力，与障碍物存在斥力，可以得到很平顺的行驶路径，但存在局部最优化问题。
- ② 禁忌搜索算法：
- ③ 模拟退火算法。

图形类算法：

- ① 棋格法：将环境划分为一系列网格，并对网格进行一系列编码， $\left\{ \begin{array}{l} \text{障碍物为 1} \\ \text{空闲为 0.} \end{array} \right\}$ 然后使用图形搜索法，类似于之前在 C++ 上做的 Planning-algorithm。
- ② 可视图空间法。
- ③ voronoi 法。

智能估量算法：

① 蚁群算法 (ACO) 迭代寻优算法。

② 神经网络算法。

③ 遗传算法

无人车轨迹跟随算法，路径跟随、速度跟随

路径跟随 $\left\{ \begin{array}{l} \text{半经验控制类模型} \\ \text{强化控制类模型} \end{array} \right.$

→ $\left\{ \begin{array}{l} \text{定速巡航控制} \quad (\text{PD 控制}, \text{模糊 PD 控制}), \\ \text{自适应巡航控制} \quad (\text{MPC}, \text{ACC}, \text{模糊神经网络控制理论}) \end{array} \right.$

智能水滴算法 (IWD)

模糊 PIP. [NB, NM, NS, ZO, PS, PM, PB]

量化指标: $\{-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6\}$, 隶属度函数.

MPC (动态矩阵控制) 模型预测控制

步骤:

- 1. 预测系统未来动态
- 2. (数值)求解开环优化问题
- 3. 将优化解的第一个元素(或者说第一部分)作用于系统

\Leftrightarrow

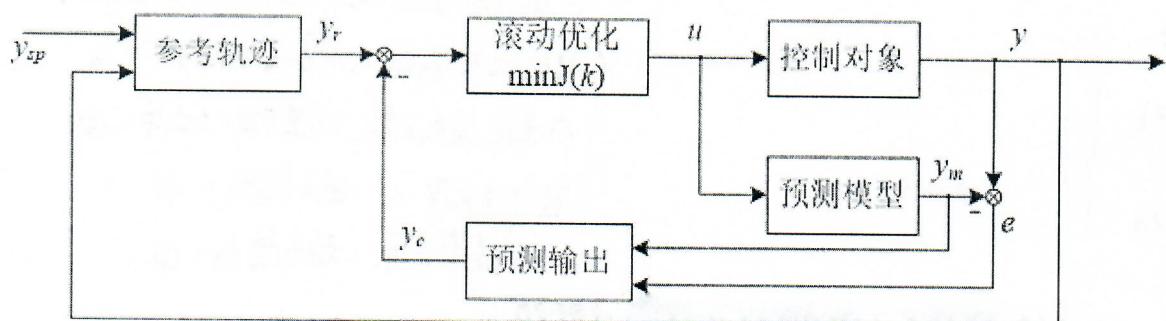
- 内部预测模型
- 参考轨迹
- 控制算法

碰撞检测算法:

- AABB
- Spheres
- OBB

$\left\{ \begin{array}{l} \text{空间分解法: k-d 树、八叉树, BSP 树、四面体网和规则网.} \\ \text{层次包围盒法: AABB, 包围球, 方向包围盒 OBB.} \end{array} \right.$

经典MPC的控制流程如图1所示：



ψ psi
 ϕ phi

图中， y_{sp} 表示系统的设定输出， y_r 表示参考轨迹， u 为输入， y 为实际输出值， y_m 为模型输出， y_c 为预测输出。

每控制周期 1. 位置 (x, y)
2. 速度 v
3. 车身角度 ψ
4. 转向角 δ_f
5. 加速度 a
获得的传感器信息

轨迹模型：
 $y = f(x) = \alpha_3 x^3 + \alpha_2 x^2 + \alpha_1 x + \alpha_0$
 $\tan(\psi) = \frac{dy}{dx} = 3\alpha_3 x^2 + 2\alpha_2 x + \alpha_1$

约束：
 $\delta \in [-25^\circ, 25^\circ]$
 $a \in [1, 1]$

动态模型

$$x_{t+1} = x_t + \cos \psi v_t dt \quad ①$$

$$y_{t+1} = y_t + \sin \psi v_t dt \quad ②$$

$$\psi_{t+1} = \psi_t + \underline{w_t} dt$$

$\because \beta = 0$, \therefore v 与车身方向同线

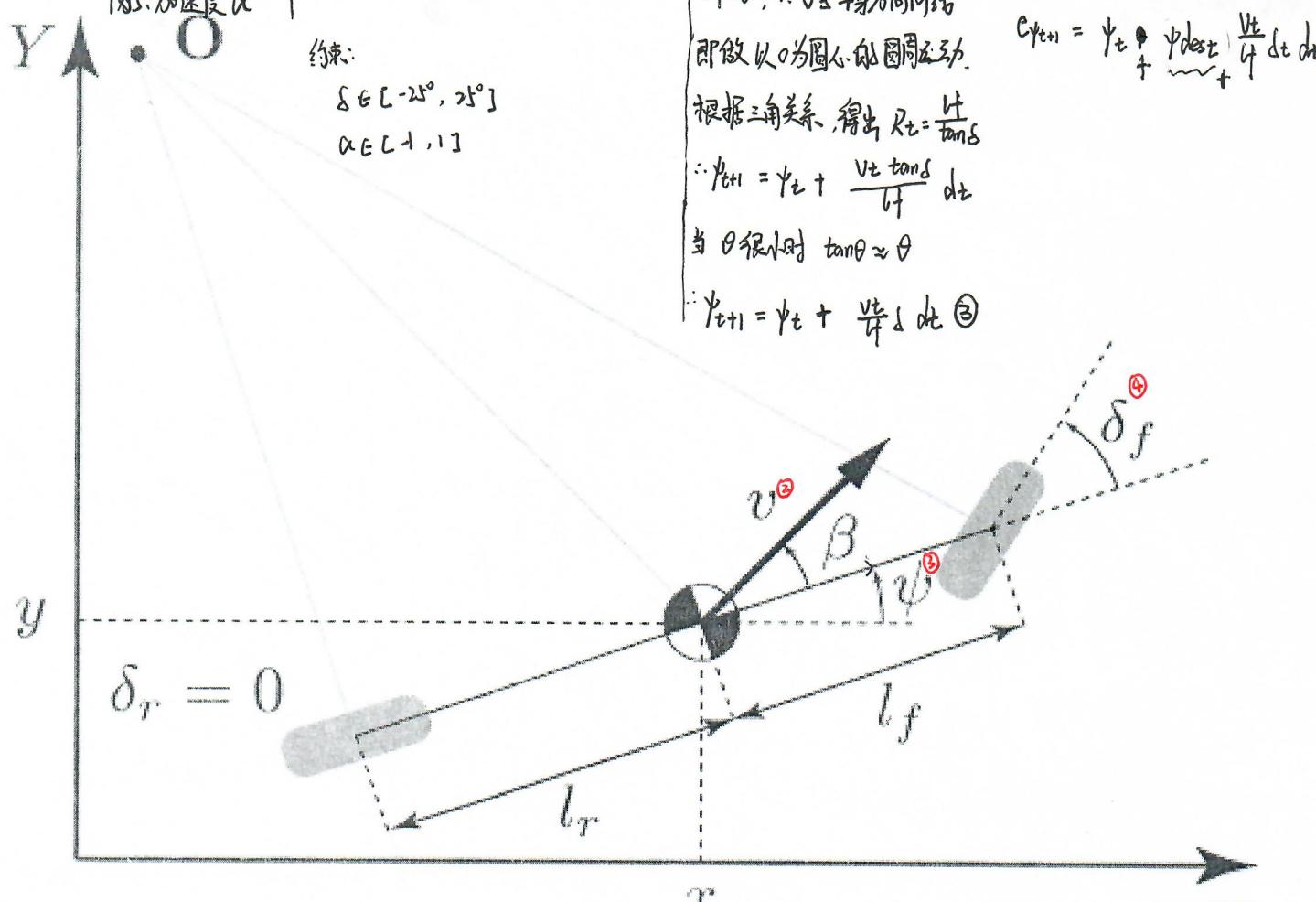
即做以 ψ 为圆心的圆周运动

$$\text{根据三角关系, 得出 } R_t = \frac{l_f}{\tan \theta}$$

$$\therefore \psi_{t+1} = \psi_t + \frac{v_t \tan \theta}{l_f} dt$$

当 θ 很小时 $\tan \theta \approx \theta$

$$\therefore \psi_{t+1} = \psi_t + \frac{v_t}{l_f} \theta dt \quad ③$$



(x, y) 重心， ψ 车身当前角度， β 为车身和速度夹角， δ_f 为前轮转角

配置 MPC-project 环境

cmake >= 3.5

make >= 4.1

gcc/g++ >= 5.4

配置 uWebSocket 难

配置 IPOPT 难

安装 CPPAD

编译环境 {
 mkdir build && cd build
 cmake .. && make
 ./mpc.

环境搭建完成，self-driving-car-sim。

运行在 term2 上。

损失函数

应包含：{
 跟踪误差 ①
 转向误差 ② (车身角度误差)
 速度损失函数项 (尽量保持在 100 英里/小时) ③
 转向损失函数 (尽量避免转向) ④
 加速度损失函数 (尽量保持加速度) ⑤
 转向变化率 (越小越好) ⑥
 加速度变化率 (越小越好) ⑦

如下，加上权重的损失函数。

$$\begin{aligned}
 J = & \sum_{t=1}^N \underbrace{w_{cte} \|c_{te,t}\|^2}_\text{状态误差} + \underbrace{w_{ep} \|e_{\psi_t}\|^2}_\text{误差} + \underbrace{w_v \|v_t - v_{target}\|^2}_\text{速度误差} \\
 & + \sum_{t=1}^{N-1} \underbrace{w_s \|\delta_t\|^2}_\text{控制量} + \underbrace{w_a \|a_t\|^2}_\text{控制量} \\
 & + \sum_{t=2}^N \underbrace{w_{rate_s} \|\delta_t - \delta_{t-1}\|^2}_\text{效率} + \underbrace{w_{rate_a} \|a_t - a_{t-1}\|^2}_\text{效率}
 \end{aligned}$$

模型预测控制理论：

如下：预测模型、滚动优化、反馈校正

1. 在过程中始终有一条期望参考线。
2. 在 k 时刻，预测 $[k, k+N]$ 时域内的输出。
3. 加入结束优化预测的输出。
4. 根据输出得到控制序列。
5. 将第控制序列的第一个元素作为控制输出。
6. 到 $k+1$ 时刻，重复上述过程。

离散化模型：

$$x(k+1) = A_{k,t} x(k) + B_{k,t} u(k)$$

$$\text{设定: } \xi(k+1) = \begin{bmatrix} x(k+1) \\ u(k+1) \end{bmatrix}$$

$$\text{得: } \xi(k+1|t) = \tilde{A}_{k,t} \xi(k|t) + \tilde{B}_{k,t} \Delta u(k|t)$$

$$\eta(k|t) = \tilde{C}_{k,t} \xi(k|t)$$

将模型预测问题转化为二次规划问题。

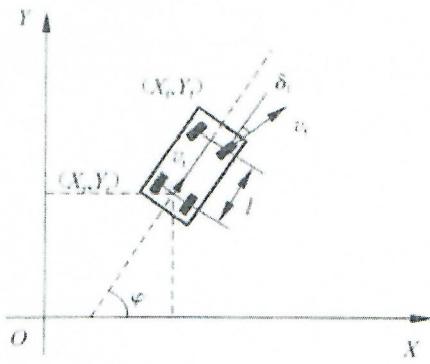


图 2.1 车辆运动模型

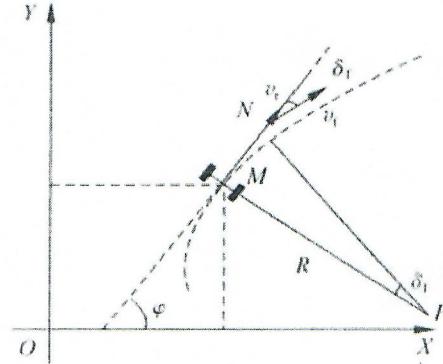


图 2.2 车辆前轮转向示意图

在后轴行驶轴心 (X_r, Y_r) 处, 速度为:

$$\dot{v}_r = \dot{X}_r \cos\varphi + \dot{Y}_r \sin\varphi \quad (2.1)$$

前、后轴的运动学约束为:

$$\begin{cases} \dot{X}_r \sin(\varphi + \delta_r) - \dot{Y}_r \cos(\varphi + \delta_r) = 0 \\ \dot{Y}_r \sin\varphi - \dot{Y}_r \cos\varphi = 0 \end{cases} \quad (2.2)$$

由式 (2.1) 和式 (2.2) 联合可得:

$$\begin{cases} \dot{X}_r = v_r \cos\varphi \\ \dot{Y}_r = v_r \sin\varphi \end{cases} \quad (2.3)$$

根据前后轮的几何关系可得:

$$\begin{cases} X_r = X_e + l \cos\varphi \\ Y_r = Y_e + l \sin\varphi \end{cases} \quad (2.4)$$

将式 (2.3) 和式 (2.4) 代入式 (2.2), 可解得横摆角速度为:

$$\omega = \frac{v_r}{l} \tan\delta_r \quad (2.5)$$

式中, ω 为车辆横摆角速度; 同时, 由 ω 和车速 v_r 可得到转向半径 R 和前轮偏角 δ_r :

$$\begin{cases} R = v_r / \omega \\ \delta_r = \arctan(l/R) \end{cases} \quad (2.6)$$

由式 (2.3) 和式 (2.5) 可得到车辆运动学模型为:

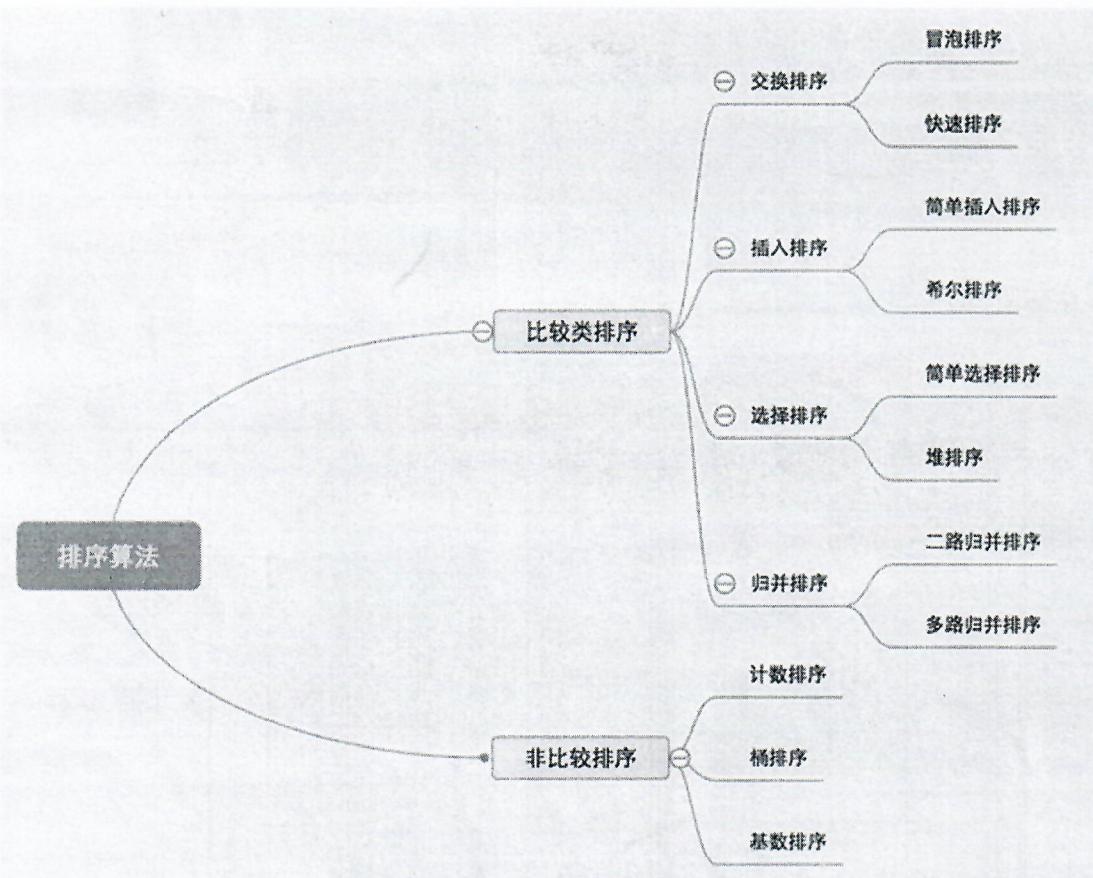
$$\begin{bmatrix} \dot{X}_r \\ \dot{Y}_r \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ \tan\delta_r/l \end{bmatrix} v_r \quad (2.7)$$

该模型可被进一步表示为更为一般的形式:

$$\dot{\xi}_{kin} = f_{kin}(\xi_{kin}, u_{kin}) \quad (2.8)$$

其中, 状态量 $\xi_{kin} = [X_r, Y_r, \varphi]^T$, 控制量 $u_{kin} = [v_r, \delta_r]^T$ 。在无人驾驶车辆的路径跟踪控制过程中, 往往希望以 $[v_r, \omega]$ 作为控制量, 将式 (2.5) 代入式 (2.7) 中, 该车辆运动学模型可以被转换为如下形式:

$$\begin{bmatrix} \dot{X}_r \\ \dot{Y}_r \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \cos\varphi \\ \sin\varphi \\ 0 \end{bmatrix} v_r + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega \quad (2.9)$$



排序方法	时间复杂度 (平均)	时间复杂度 (最坏)	时间复杂度 (最好)	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1-\epsilon})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(n\log_2 n)$	$O(n^2)$	$O(n\log_2 n)$	$O(n\log_2 n)$	不稳定
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	稳定
计数排序	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	稳定
桶排序	$O(n+k)$	$O(n^2)$	$O(n)$	$O(n+k)$	稳定
基数排序	$O(n*k)$	$O(n*k)$	$O(n*k)$	$O(n+k)$	稳定

优先队列使用 heap

map, set 使用红黑树

路径规划算法

Bug1, Bug2 算法

Dijkstra, BFS, 启发式 A*

PRM 随机路图法
 RRT

7. 红黑树

- 1. 根节点一定是黑色的
- 2. 节点只能是红或黑
- 3. 每个叶节点一定是黑色的
- 4. 每个红色节点的子节点一定是黑色的
- 5. 从任一节点到根节点的所有路径包含相同的黑色结点数。

8. 线程、进程、协程

进程是并发执行的，每个进程都是独立的，有独立的数据存储空间。

线程是 CPU 调度的一个基本单位，在一个进程中两个线程，共享进程中的数据，堆
在内部有自己的栈

考点：

1. 伪类 → 本质上重载了一个 operator.

2. 结构体、共同体区别
对齐 所有数据共用一个内存

3. static 和 const 区别

↓
改变其存储区域，生命周期，不改变其作用域

static

- 局部变量
- 全局变量
- 成员变量

const

- 限定变量不可修改
- 限定成员函数不能修改或设
- const 指针

4. 指针与引用的区别

1. 指针是变量，是个地址值，引用是别名，没有额外的空间
2. 指针可以在定义时不赋值，但引用定义时必须初始化
3. 指针是变量可以变，但引用一旦定义便不可变了。
4. sizeof(指针) 得到指针位数， sizeof(引用) 得到变量大小
5. 指针自增++ 与引用不同

5. 多态（一个接口，多种方法），运用虚函数实现。

用法声明基类的指针指向任意子类对象，即可调用相应子类的函数

虚函数表。

6. 排序复杂度

	平均	最坏	查找
冒泡	n^2	n^2	顺序: n
选择	n^2	n^2	二分: $\log n$
插入	n^2	n^2	插入: $\log n$
希尔	$n^{1.3}$	n^2	
快速	$n \log n$	n^2	Fibonacci: $\log n$
堆	$n \log n$	$n \log n$	Hash O(1)
归并	$n \log n$	$n \log n$	

9. 堆：由程序员申请、释放、动态。
 栈：由操作系统自动释放。

10. STL：标准模板库

- 容器
- 迭代器
- 算法
- 函数对象
- 内存分配器
- 适配器

面试准备

1. C++ (`static` & `const` 区别, 指针与引用、类、线程进程, STL 模板库, 函数重载, 指针类型和大小, 继承方式, `protected` 成员, 前置++、后置++, 左值、右值)
2. Linux
3. ROS (文件结构, 通信方式 (`msg`, `srv`, parameter, `srv`, Action, TF tree)
4. `Robot - planning` (BFS, 广度优先搜索, Dijkstra, A*, RRT, PRM, IWD)
5. PID, MPC control (PID, 模糊PID, MPC)
6. Data-structure (排序复杂度, 查找复杂度, tree, 平衡树, graph)