目录

[common/grid\_map.h 3](#_Toc49265248)

[MapInfo 3](#_Toc49265249)

[GlobalCoordinateMap (Abstract) 3](#_Toc49265250)

[common/simple\_grid\_map.h 3](#_Toc49265251)

[SimpleGridMap 3](#_Toc49265252)

[common/landmark\_map.h 4](#_Toc49265253)

[LandmarkMap 4](#_Toc49265254)

[LandmarkMapGenerator 4](#_Toc49265255)

[common/sensor\_data.h 4](#_Toc49265256)

[ImuData2D 4](#_Toc49265257)

[PointCloudData 5](#_Toc49265258)

[WheelSensorData 5](#_Toc49265259)

[common/time.h 5](#_Toc49265260)

[get\_current\_time\_us() 5](#_Toc49265261)

[common/pose.h 6](#_Toc49265262)

[Position 6](#_Toc49265263)

[operator-(const Position& a, const Position& b) 6](#_Toc49265264)

[operator+(const Position& a, const Position& b) 6](#_Toc49265265)

[operator\*(const Position& a, const Position& b) 6](#_Toc49265266)

[common/huffman.h 6](#_Toc49265267)

[huffman\_compress(std::vector<char> \*, std::vector<char> \*) 6](#_Toc49265268)

[huffman\_decompress(std::vector<char> \*, std::vector<char> \*) 6](#_Toc49265269)

[common/thpool.h 7](#_Toc49265270)

[msg\_queue.h 7](#_Toc49265271)

[MsgQueueListener (Abstract) 7](#_Toc49265272)

[MsgQueue 7](#_Toc49265273)

[imu.h 8](#_Toc49265274)

[IMUModule 8](#_Toc49265275)

[radar.h 8](#_Toc49265276)

[RadarModule 8](#_Toc49265277)

[wheel.h 8](#_Toc49265278)

[WheelSensorModule 8](#_Toc49265279)

[move.h 8](#_Toc49265280)

[MoveInstruction 8](#_Toc49265281)

[MoveModule 9](#_Toc49265282)

[odometer.h 9](#_Toc49265283)

[Odometer 9](#_Toc49265284)

[scan\_matching.h 9](#_Toc49265285)

[scan\_matching(Position &, GlobalCoordinateMap \*, PointCloudData &, ScanMatchingOptions &) 9](#_Toc49265286)

[main.cpp 10](#_Toc49265287)

[TCPDataPackage 10](#_Toc49265288)

[handle\_package(TCPDataPackage \*, TCPDataPackage \*) 10](#_Toc49265289)

[handle\_socket(void \*) 10](#_Toc49265290)

[running\_network() 10](#_Toc49265291)

[main(int, char \*[]) 10](#_Toc49265292)

# common/grid\_map.h

## MapInfo

处理格栅地图的元数据及坐标转换。

**Members**

width 宽，单位格栅

height 高，单位格栅

origen\_x 左上角在全局坐标系中的坐标，单位米

origen\_y 同上

resolution 格栅大小/地图分辨率，单位米

global\_x\_to\_map\_x(double x) 全局坐标系（米）到格栅矩阵（列）

global\_y\_to\_map\_y(double y) 全局坐标系（米）到格栅矩阵（行）

map\_xy\_to\_array\_index(int x, int y) 格栅矩阵（列，行）到格栅序号

array\_index\_to\_map\_xy(int index, int \*px, int \*py) 格栅序号到格栅矩阵（列，行）

map\_x\_to\_global\_x(int x) 格栅矩阵（列）到全局坐标系（米）

map\_y\_to\_global\_y(int y) 格栅矩阵（行）到全局坐标系（米）

## GlobalCoordinateMap (Abstract)

定义可以通过全局坐标系访问的地图接口。

**Members**

get(double x, double y) 获取全局坐标系的坐标对应的地图数据，（0-1）

get\_info() 获取地图元数据

# common/simple\_grid\_map.h

## SimpleGridMap

基本的格栅地图数据结构，3.12版本后包含了反射板地图（LandmarkMap）

**Members**

SimpleGridMap(MapInfo & info) 根据地图元数据初始化空白地图

SimpleGridMap(char \* data\_buf, int buf\_size) 根据二进制数据读取地图，数据需来自其他地图或地图文件或网络流

get(double x, double y) 获取全局坐标系坐标（x，y）对应的格栅概率值（0-1）

get\_info() 获取地图元数据

get\_data(std::vector<char> \*output) 编码地图数据并写入output指向的列表尾部

get\_landmark\_map() 获取包含的反射板地图指针

# common/landmark\_map.h

## LandmarkMap

反射板地图数据结构。以稀疏矩阵形式储存的二值化格栅地图。

**Members**

LandmarkMap() 不建议使用

LandmarkMap(MapInfo &info) 不建议使用

insert(double x, double y) 将全局坐标系坐标（x，y）所在位置标记为反射板

insert(int index) 将格栅序号为index的格栅标记为反射板

set\_mapinfo(MapInfo &i) 不建议使用

set\_data(std::vector<char> &datas\_with\_info) 不建议使用，由SimpleGridMap调用

gridding() 格栅化稀疏矩阵，将其转为数组形式加速访问，经过修改后需要重新调用

get(double x, double y) 获取全局坐标系坐标（x，y）是否被标记为反射板

get\_info() 获取地图元数据

get\_data(std::vector<char> \*output) 不建议使用，由SimpleGridMap调用

## LandmarkMapGenerator

构造反射板地图时的中间数据结构。负责统计激光点云数据，计算地图大小，构造地图坐标系，二值化地图。

**Members**

LandmarkMapGenerator(double resolution) 初始化目标地图的分辨率，单位米

add\_point(double x, double y) 添加全局坐标系下反射板的观测点坐标，单位米

to\_landmark\_map() 根据当前输入的观测点构建反射板地图

# common/sensor\_data.h

## ImuData2D

表示一帧IMU数据

**Members**

timestamp\_us 数据帧上传到本机时的系统时间戳，单位微秒

linear\_acceleration\_x x轴方向（前方）的加速度，单位米每秒平方

linear\_acceleration\_y y轴方向（右方）的加速度，单位米每秒平方

angular\_velocity\_z z轴逆时针方向旋转角速度，单位弧度每秒

angle\_z z轴旋转角，单位弧度

to\_char\_array(std::vector<char> \*output) 编码为二进制数据并写到output列表尾部

from\_char\_array(char \*buf, int size) 解码并读取二进制数据

## PointCloudData

一帧激光雷达数据

**Members**

timestamp 数据帧上传到本机时的系统时间戳，单位微秒

points 点云相对于激光雷达局部的坐标列表，单位米

intensities 点云的激光反射率列表，（0-1）

add\_point(double x, double y, double intensity) 插入一个激光反射点

to\_char\_array(std::vector<char> \*output) 编码为二进制数据并写到output列表尾部

from\_char\_array(char \*buf, int size) 解码并读取二进制数据

## WheelSensorData

一帧轮子编码器数据

**Members**

timestamp 数据帧上传到本机时的系统时间戳，单位微秒

type 左右轮信息，取WHEEL\_DATA\_TYPE\_LEFT或WHEEL\_DATA\_TYPE\_RIGHT

data 该编码器当前的累计数值

to\_char\_array(std::vector<char> \*output) 编码为二进制数据并写到output列表尾部

from\_char\_array(char \*buf, int size) 解码并读取二进制数据

# common/time.h

## get\_current\_time\_us()

获取系统当前时间戳，单位微秒

# common/pose.h

## Position

表示某个时间点的二维位姿。以二维刚体运动形式表示

**Members**

timestamp 时间戳，单位微秒

x 全局坐标系的x轴坐标

y 全局坐标系的y轴坐标

theta 相对于x轴逆时针旋转的角度，单位弧度，（0 – 2pi）

## operator-(const Position& a, const Position& b)

计算两个位姿的向量差

## operator+(const Position& a, const Position& b)

计算两个位姿的向量和

## operator\*(const Position& a, const Position& b)

使用位姿a变换位姿b得到的位姿

# common/huffman.h

## huffman\_compress(std::vector<char> \*, std::vector<char> \*)

使用huffman编码压缩input指向的列表中的内容，结果写入（不是追加）到output指向的列表

## huffman\_decompress(std::vector<char> \*, std::vector<char> \*)

解码input指向的列表中的huffman编码，结果写入（不是追加）到output指向的列表

# common/thpool.h

线程池库（C风格面向对象）

**Members**

thpool\_init(int num\_threads) 初始化线程池并指定其中的最大线程数

thpool\_destroy(threadpool) 释放线程池

thpool\_add\_work(threadpool, void (\*function\_p)(void\*), void\* arg\_p) 向任务序列中添加任务，格式为含有一个void\*参数的函数指针

# msg\_queue.h

程序内通讯机制，分频道发送/监听消息

## MsgQueueListener (Abstract)

抽象类，定义监听者的接口，所有监听者都需要实现该接口

**Members**

recieve\_message(const int channel, char \*buf, const int size) 当监听的频道有新消息时消息队列会调用该函数，参数为包含消息内容的内存块信息，传入的内存为只读共享内存，在该函数返回前保证有效

## MsgQueue

消息队列主体，在程序中全局单例。监听者将自身注册到队列的某个频道上，发送者选择频道发送信息。同一频道的多个监听者的消息处理异步进行。消息为二进制内存块，格式由频道的发送者与接收者协商。

**Members**

MsgQueue(int thread\_num) 初始化消息队列，指定使用的最大线程数

add\_listener(const int channel, MsgQueueListener \* p\_listener) 将某个监听者绑定到给定的频道编号上

remove\_listener(const int listener\_id) 根据绑定listener时分配的id移除该绑定

remove\_listener(const int channel, MsgQueueListener \* p\_listener) 移除该绑定

clear() 清空所有绑定信息

send(int channel, char \* buf, int size) 向某个频道发送消息

print\_debug\_string() 向标准输出打印消息队列当前的频道信息

# imu.h

## IMUModule

与IMU的串口通信，接受IMU数据并发到消息队列中

**Members**

IMUModule() 初始化串口，启动IMU线程，启动IMU

~IMUModule() 停止IMU，关闭串口

set\_target(MsgQueue \* target, int channel)

# radar.h

## RadarModule

激光雷达通信。同上

# wheel.h

## WheelSensorModule

轮子传感器通信。同上

# move.h

## MoveInstruction

表示一帧移动命令的数据结构。指令有两种类型：（1）指定前进速度和转向速度 （2）指定当前位姿和目标位姿

**Members**

type 取MOVE\_INSTRUCTION\_TYPE\_SPEED或MOVE\_INSTRUCTION\_TYPE\_TARGET

speed 是速度指令时有效

target 是目标点指令时有效

## MoveModule

与控制板通信，监听消息队列中的移动指令，接收并下发移动指令。

**Members**

MoveModule(MsgQueue \*msg\_queue, int channel) 初始化监听

handle\_movement\_message(int speed, int turn\_left) 接收手动输入的移动命令

handle\_movement\_message(struct movement\_instruction\_target target) 同上

# odometer.h

## Odometer

里程计。负责处理IMU数据与编码器数据。里程计会接收并保存最近的可信位姿（来自激光雷达匹配），以及时间戳在最近的可信位姿之后的IMU数据和编码器数据，从最近的可信位姿开始估计当前位姿。

**Members**

get\_last\_pose\_time() 最近的可信位姿的时间戳

get\_last\_data\_time() 最近的传感器数据的时间戳

insert\_odom\_data(long timestamp, long data, int type) 插入一帧轮子传感器数据

insert\_imu\_data(long timestamp, double w) 插入一帧IMU数据

insert\_position(Position &position) 插入一帧可信位姿

expolate\_pose(long timestamp, Position \* output\_p) 估计某个时间戳的位姿。目标时间戳应大于最近的可信位姿

# scan\_matching.h

执行scan matching点云匹配算法

scan\_matching(Position &, GlobalCoordinateMap \*, PointCloudData &, ScanMatchingOptions &)

以init为起始参数，根据地图和点云进行匹配优化，返回优化后的位姿。

# main.cpp

## TCPDataPackage

数据包类。处理与上位机的通信协议。

**Members**

TCPDataPackage(int func, int arg1, int arg2, int data\_len) 初始化数据包

TCPDataPackage(char \*head, int \*check\_result) 解析二进制数据并初始化数据包

reserve\_buf(int size) 预分配额外数据空间

to\_bytes(char \*buf) 获取二进制格式的包头

## handle\_package(TCPDataPackage \*, TCPDataPackage \*)

读取并处理输入数据包，将回复数据写入输出数据包。函数返回后输出数据包将被发回上位机。

## handle\_socket(void \*)

用于处理一次上位机连接请求。线程池中的线程函数。负责建立连接，接收数据，发送回答，关闭连接。

## running\_network()

循环监听网络端口，将新连接加入线程池处理。

## main(int, char \*[])

main函数。读取参数，初始化配置文件，初始化AGV，进入网络循环。