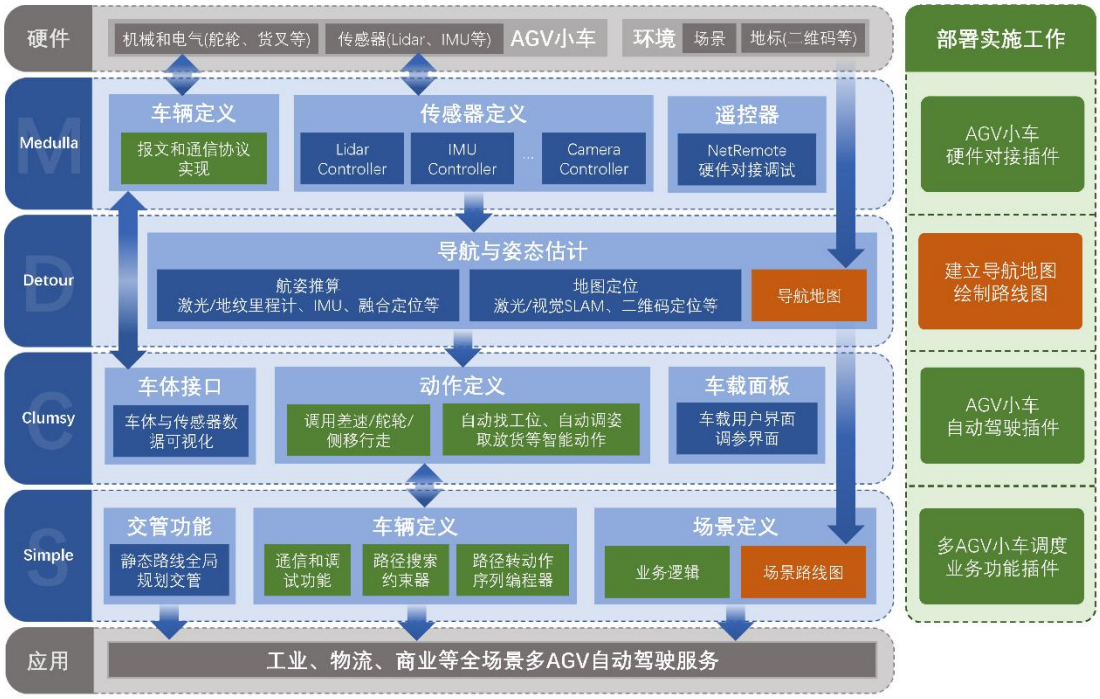
目录

第一部分 MDCS 部署使用指南

1 初识 MDCS



MDCS（Medulla，Detour，Clumsy，Simple）是一套完整的 AGV（Automated Guided Vehicle）小车自动驾驶算法与控制架构，它由以下 4 个软件组成：

- Medulla：硬件适配软件，提供通信抽象，通过插件对接各类硬件；
- Detour：定位软件，包含成熟的 SLAM 导航、二维码导航、地面纹理导航算法，通过调整参数即可进行部署，得到车体运行环境和车的位姿；
- Clumsy：自动驾驶软件，定义 AGV 的运动模型抽象形式，如差速轮、单舵轮、万向轮等；定义可用动作，如前进行走、后退行走、自动检测并行驶至工位或托盘等；
- Simple：AGV 调度软件，完备的静态调度系统。

Medulla、Detour 和 Clumsy 为车载程序，支持 Windows/Linux；RCS 调度系统 Simple 目前仅支持 Windows 部署。

需要注意：在 Windows 上 Medulla、Detour、Clumsy、Simple 均需以管理员权限运行；在 Linux 上，Medulla、Detour、Clumsy 均需以 root 运行。本 cookbook 中所有操作均默认以 Windows 管理员或 Linux root 用户进行。

2 Windows

2.1 目录结构和环境配置

车载：

```
MDCS/
    Medulla/
        plugins/
            CartActivator.dll
            LidarController.dll
            LessokajiDemo_m.dll
        Medulla.exe
        NetRemote.exe
        startup.iocmd
    Detour/
        Detour.exe
        conf.json
        mainmap.2dlm
    Clumsy/
        ClumsyConsole.exe
        LessokajiDemo_c.dll
        clumsy.json
```

调度服务器：

```
Simple/
    plugins/
        LessokajiDemo_Scene.dll
    SimpleComposer.exe
    simple.json
```

在上述文件目录结构中，MDCS 文件夹包括了部署在车端的 3 个软件，Simple 一般部署于场景中另外一台与所有小车在同一局域网内的调度电脑或服务器上，故单独放在 Simple/文件夹中进行说明。

在 Windows 系统下，无需安装额外的运行时环境。

2.2 Medulla

双击 MDCS/Medulla/Medulla.exe 即可启动。

2.2.1 插件加载器

Medulla 本质上是一个插件加载器。通常来说，车体会作为单独的一个插件进行加载，车上的各个传感器分别作为插件进行加载。每一个插件为一个 DLL（动态链接库），规定所有的 DLL 均放在 Medulla/plugins/中。

Medulla 本身使用类似命令行的方式，在运行时构建各插件的对象。一般来说，将 Medulla 需要运行的命令写在文本文件 startup.iocmd 中，Medulla 在启动时将自动加载运行该脚本并调用相关的插件。示例如下：

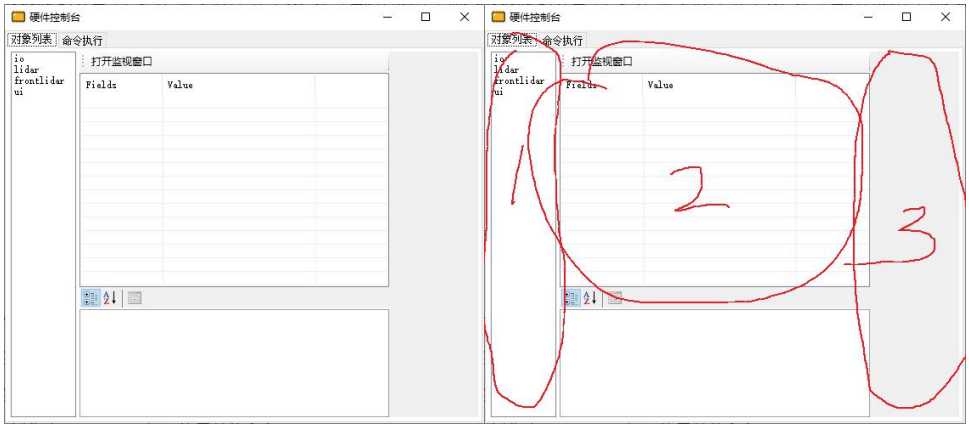
```
// CartActivator.dll 为车体插件的加载器
loader = io load plugins\CartActivator.dll
// 下面这一步是可选的，对插件中支持的一些初始化参数进行赋值，这里以给最大速度 MaxSpeed 赋值为例。这一步必须放在车体插件被加载之前进行
loader setp MaxSpeed 3000
// 用该加载器实例 loader 加载车体插件 ProjName_m.dll，得到车体类实例 cart
cart = loader load plugins\ProjName_m.dll
// 使用 WLR716Lidar.dll 插件实例化万集 716 雷达作为导航雷达 frontlidar
frontlidar = io load plugins\WLR716Lidar.dll
// 在 IP 地址 192.168.137.180 进行连接
frontlidar start 192.168.137.180
// 加载 WinMedulla.dll 插件并调用 Show 方法，显示 Medulla 的用户界面
ui = io load plugins\WinMedulla.dll
ui Show
```

Medulla 启动之后，在 Medulla 的用户界面中，选择“命令执行”标签页，即可在运行时输入新的指令。所有的单线激光雷达有一组通用的操作命令，示例中的 start 为通用的启动命令。其余通用命令详见附录 A。

需要注意：雷达倒置安装时，需要设置镜像：frontlidar setMirror true

2.2.2 用户界面

在 startup.iocmd 中加载 WinMedulla.dll 插件并使用 Show 方法后，Medulla 的用户界面如上图所示。区域 1 中显示所有已加载的插件的实例化对象。选择某一对象后，区域 2 中将显示其属性和相应的值。区域 3 中为当前选择对象支持的功能，例如：cart 对象的 remote 按钮会调出 Medulla 自带的简易遥控器；单线雷达对象的 view 按钮会调出雷达点云的可视化页面。

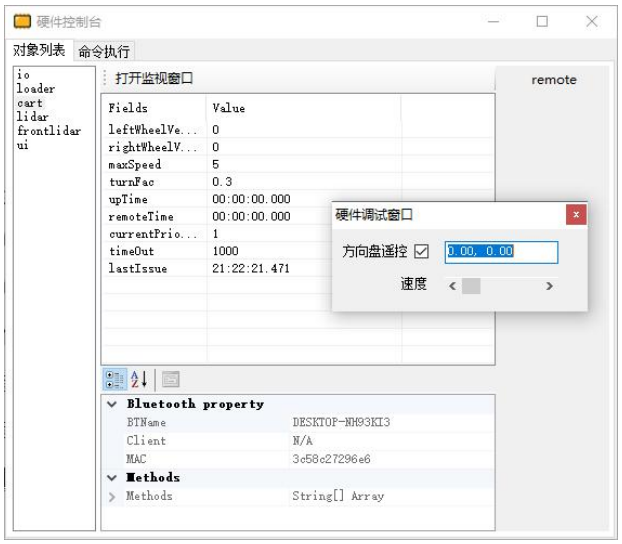


2.2.3 遥控器

Windows 下 Medulla 常用 2 种形式的遥控器。

需要注意：各种遥控器与自动驾驶（Clumsy）对车体信号的控制有优先级顺序：简易遥控器优先级为 0，http 连接的遥控器（包括 NetRemote.exe 和 3.X 中提到的 Web 页面遥控器）优先级为 1，蓝牙遥控器优先级为 2，Clumsy 自动驾驶优先级为 4。优先级数字越小，代表优先程度越高，高优先级的控制方式将屏蔽低优先级的控制方式。例如开启遥控器时，Clumsy 下发的命令无效。

第一种是 Medulla 内置的简易遥控器，通过 cart 对象的 remote 按钮可直接打开该遥控器。该简易遥控器勾选后有效。使用时自动绑定键盘↑、↓、←、→控制车体前后左右移动。下方速度滑钮控制速度大小。



第二种是 NetRemote.exe 遥控器应用程序。如下图所示。在左上方输入小车 IP 后，点击“确认”进行连接。“点击激活遥控”选中后遥控器生效。默认加载同目录下 remoteconf.json 配置文件。当没有配置文件时，打开 NetRemote.exe 会使用默认布局，点击“编辑”后可对小组件进行编辑：鼠标左键按住拖动位置；右键可修改名称、删除组件或设定键位绑定；点击“插入”

可增加新的组件；点击“保存”会覆盖掉同目录下的配置文件。绑定名称见 4 使用规则和约定。

第三种是网页访问的遥控器。如下图所示。目前该遥控器只能通过 Clumsy 的页面跳转进行访问。具体访问方法见 2.3。使用时需将遥控器配置文件放在 Medulla/htdocs/remoteconf.json。界面操作方法与 NetRemote.exe 相同。

第四种是在安卓设备上安装 NetRemote.apk。可通过 WiFi 或蓝牙连接小车进行遥控。操作界面操作方法与 NetRemote.exe 相同。

需要注意：遥控器对车体控制具有最高优先级的控制权限。开启遥控器时，Clumsy 下发的自动驾驶控制信号将被屏蔽。

可以选择 Medulla UI 上的 cart 组件，查看 currentPriority 判断目前遥控器控制来源：

- currentPriority==−1 表示无控制。
- currentPriority==0 表示 Clumsy 自动驾驶控制。
- currentPriority==1 表示 Medulla 内置简易遥控器控制。
- currentPriority==2 表示 Web 遥控器或 NetRemote。
- currentPriority==3 表示蓝牙遥控器。
- currentPriority==4 表示特殊种类遥控器，如实体遥控器（若有）。

currentPriority 数值越大的选项表示优先级越高，即，如果数值更大的控制源存在，其他控制源将被 Medulla 屏蔽。

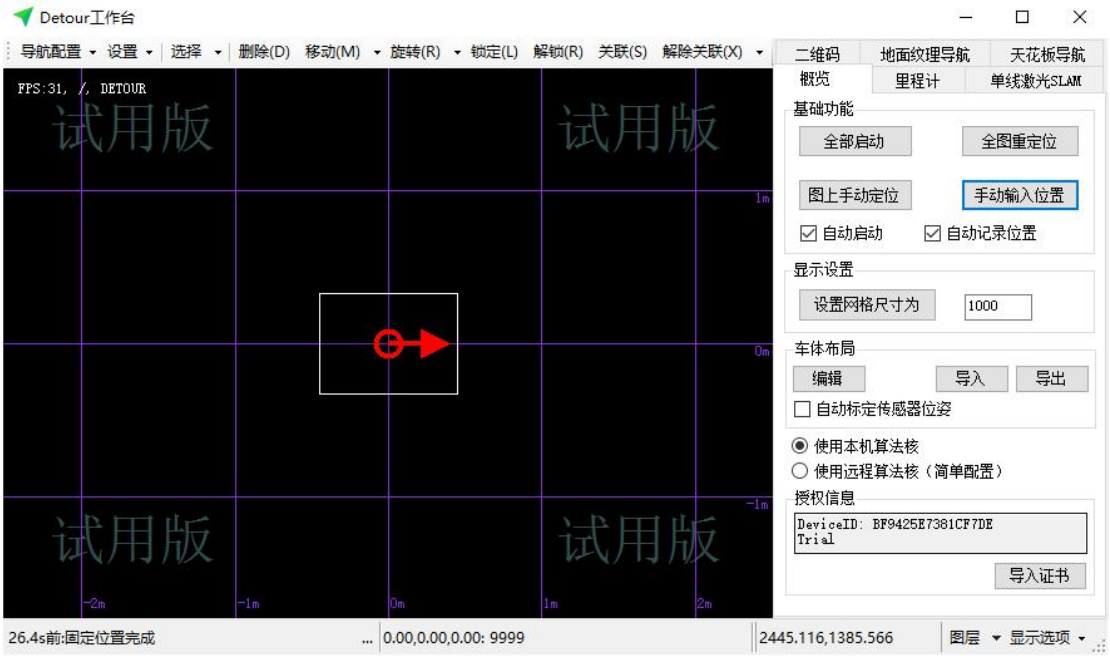
2.3 Detour

打开 Detour 之前应启动 Medulla。双击 MDCS/Detour/Detour.exe 即可启动。以下用单线激光雷达为例，说明如何使用 Detour 进行建图和定位。

2.3.1 车体布局编辑器

开始导航前务必需要确保车体的轮廓、雷达位置角度、扫描参数等设置已经正确。如果没有正确设置这些参数，会导致以下问题：无法有效更新地图；地图上有车体自身导致的杂点和拖影；运动控制不精确甚至失灵。

在概览页面，车体布局一栏中，点击编辑按钮即可打开车体布局编辑器。

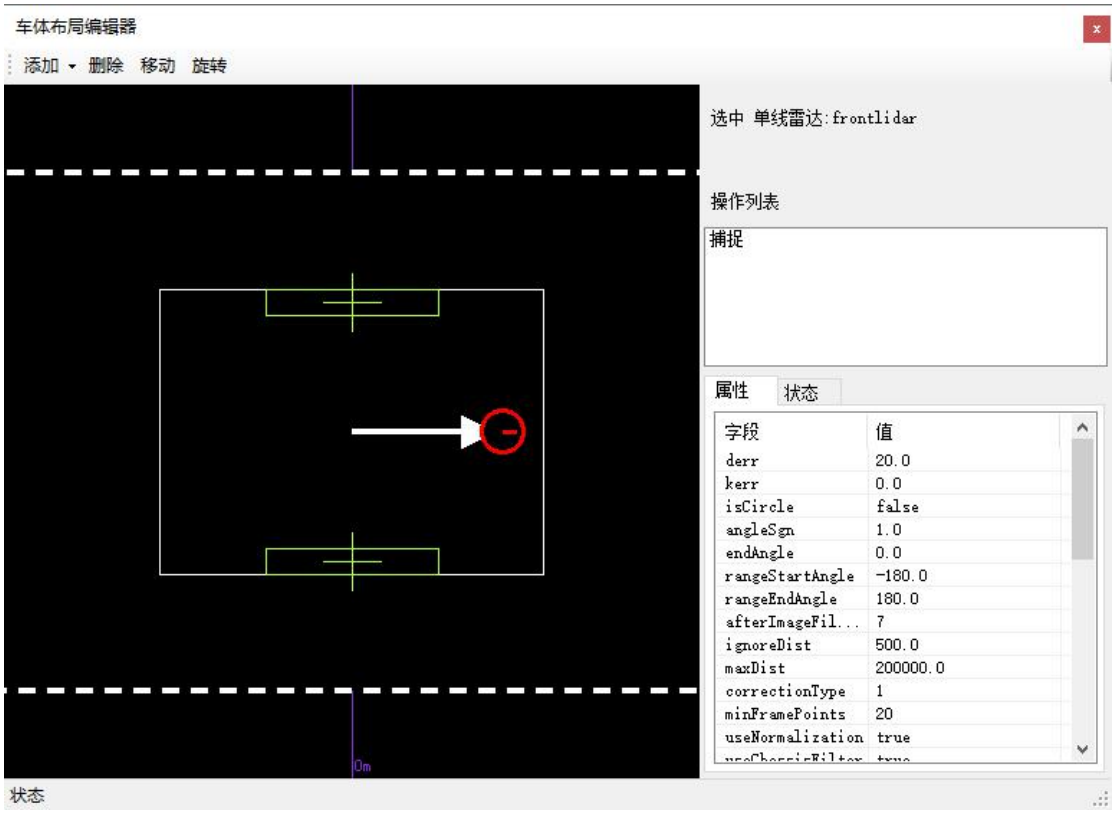


编辑车体轮廓。鼠标选中车体边框（选中时为红色）。双击操作列表中的重绘制轮廓。在左侧界面上依次闭绘制闭包的折线段（按下鼠标左键开始新增点，若单击后按住拖动可调整当前点的位置。右键结束绘制）。车体轮廓应当覆盖雷达扫描到车体自身所产生的杂点。

编辑单线激光雷达。选中左侧界面中的激光雷达后，需要修改右侧属性栏中的以下参数：

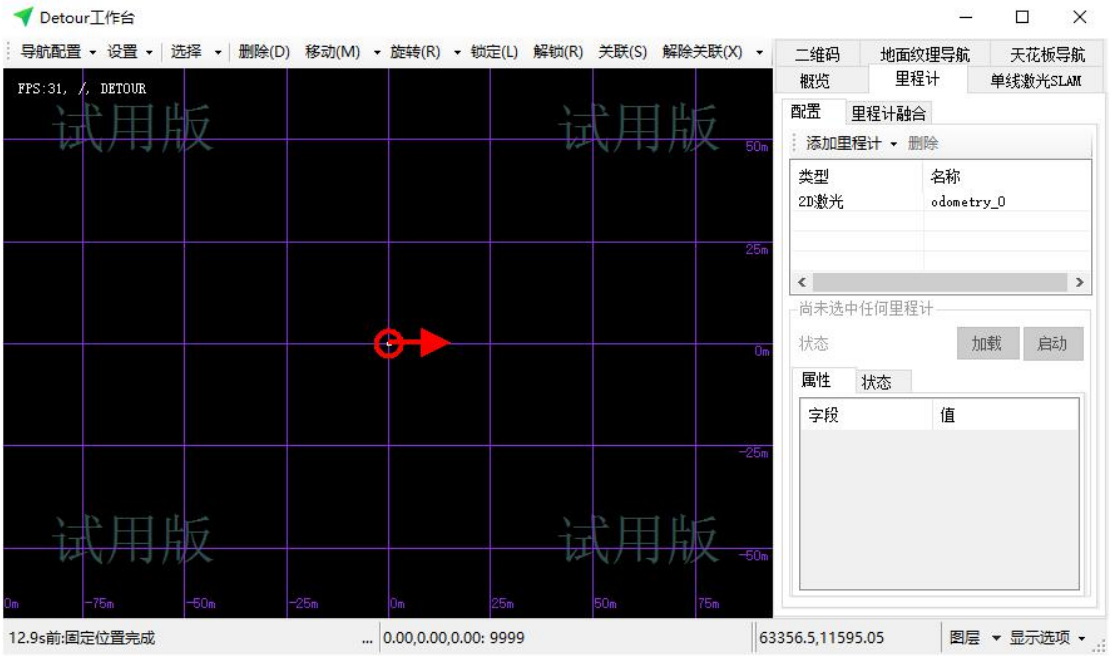
- x，y，th：雷达安装位姿相对驱动中心（舵轮或差速轮轴中心）位置，单位 mm；
- angleSgn：雷达扫描方向，1 为逆时针，−1 为顺时针。此项为雷达本身特性，不随雷达安装方式变化而改变；
- endAngle：雷达每帧扫描结束的角度。此项为雷达本身特性，不随雷达安装方式变化而改变，也不受 rangeStartAngle 和 rangeEndAngle 影响；
- rangeStartAngle：雷达实际有效扫描范围开始的角度；
- rangeEndAngle：雷达实际有效扫描范围结束的角度；
- ignoreDist：雷达扫描开始距离，用于屏蔽激光周围杂点；
- maxDist：雷达扫描最大半径。

其中，MDCS 已支持的雷达型号对应的 angleSgn、endAngle 可参见附录 B。



2.3.2 里程计和图层

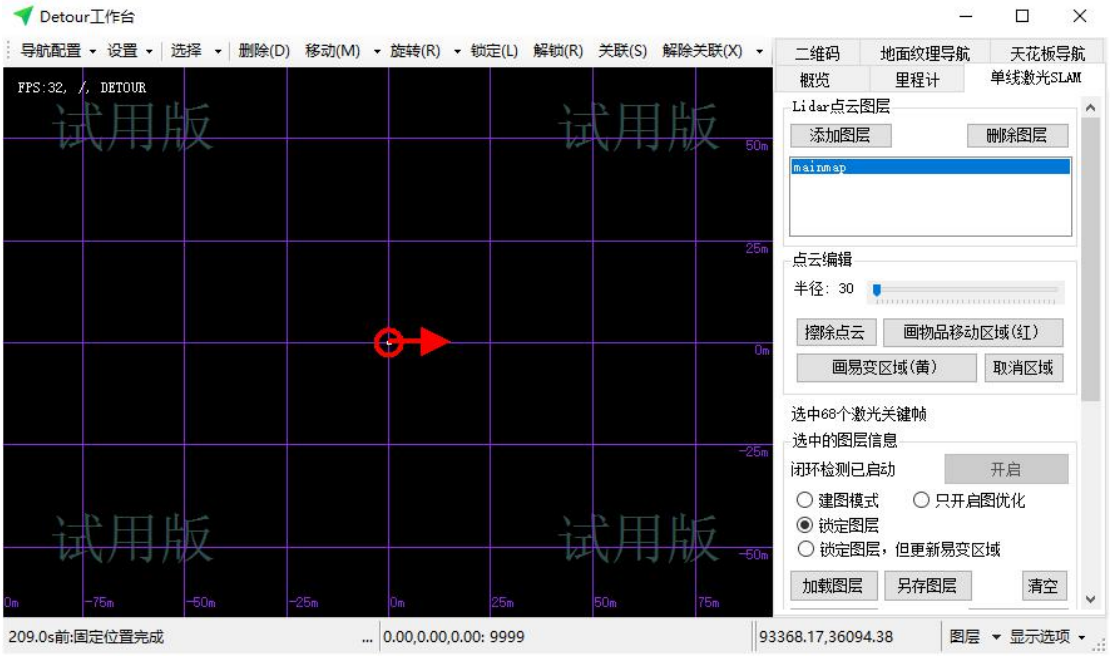
添加单线激光雷达里程计：选择里程计标签页，点击添加里程计下拉菜单，选择 2D 激光。一般使用默认名称 odometry_0。可以调节属性中的 switchingDist 字段，该属性表示最多靠激光里程计走多少距离必须发生一次回环检测，一般设置为 1000−7000。2D 激光里程计其余属性一般保持默认即可。



添加单线激光 SLAM 图层。选择单线激光 SLAM 标签页，点击添加图层。一般使用默认名称 mainmap。选中添加的图层后，点选列表中的 mainmap，可在下方切换模式：

- 建图模式：地图自动更新；
- 只开启图优化：不向地图添加帧，仅调整现有地图帧的位置；
- 锁定图层：不更新地图；
- 锁定图层，但更新易变区域：只更新地图上标记为易变区域的部分。

一般来说，只需在建图前选择建图模式，完成建图后选择锁定图层即可。

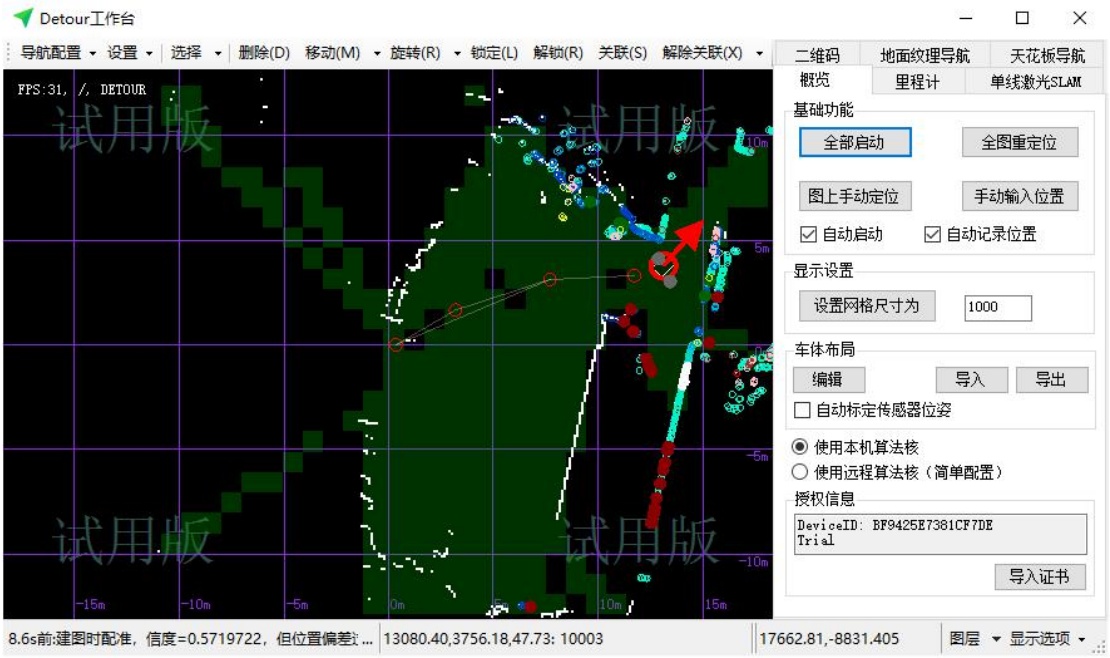


2.3.3 定位与建图

在进行这一步之前，应当确保导航使用的单线激光雷达完成了校准。校准的方法见 5.3。

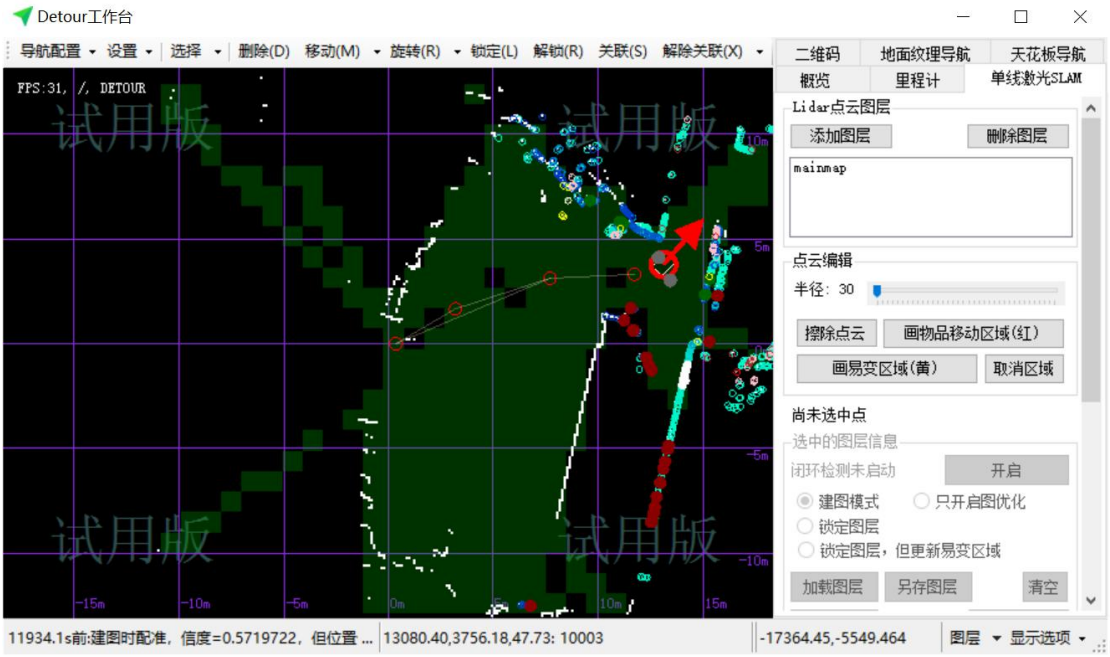
录制地图

点击概览标签页中的全部启动按钮后，请现场的人员将 AGV 在整个业务涉及到的区域中行驶一圈。确保最后回到起点附近，且最后一个关键帧与第一个关键帧之间完成了回环（关键帧是图上的红色圆圈，红色圆圈间有黄色细线代表有回环关系）。

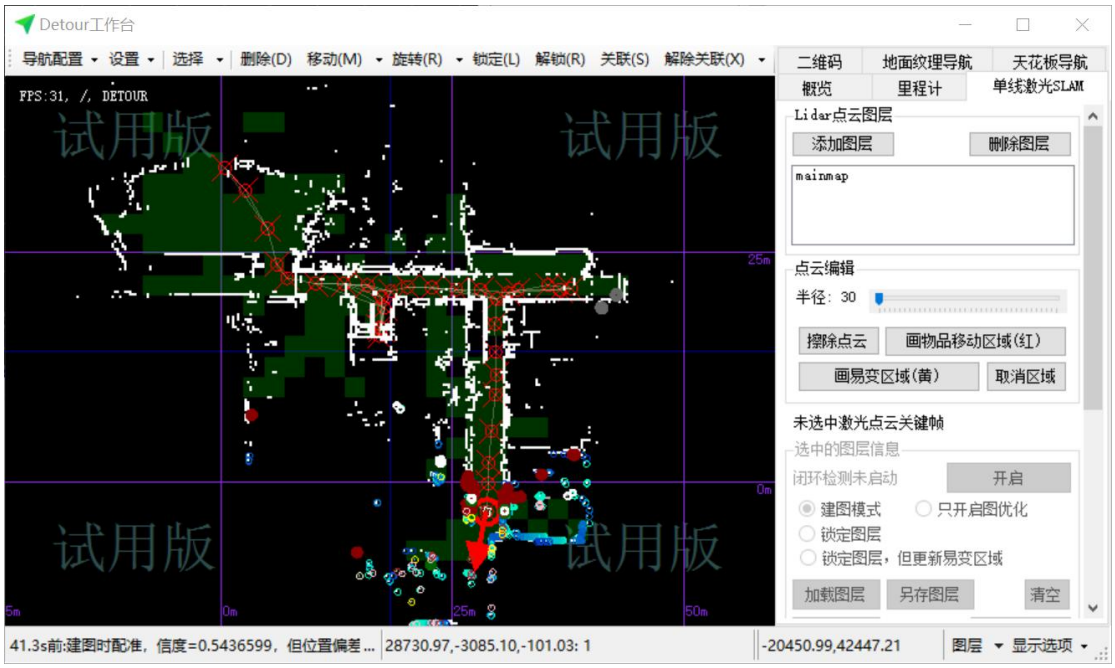


编辑与保存地图

在单线激光 SLAM 标签页，可通过擦除点云、画物品移动区域、画易变区域等，编辑已经建立的地图。可通过拖动上方的滑动条来改变工具的大小。



此外，可通过选中关键帧，再点击上方菜单栏的移动、旋转，来修正一帧或多帧关键帧。可框选所有帧，整体移动、旋转整个地图。被移动、旋转的关键帧点将被锁定（被锁定的关键帧会有个 X 号）。可点击“解锁”，使得 Detour 可以继续自行优化地图。



需要注意：应当在地图扫完之后将整个地图“转正”，使得后续 Simple 中建立站点和路径更加方便。（“转正”是个相对的概念）

建完图后，点击单线激光 SLAM 标签页，选中相应的图层，点击另存图层，即可将地图保存至硬盘。通常，mainmap 保存时文件名取为 mainmap.2dlm。

其他常用的操作有：

- 勾选概览标签页的自动启动和自动记录位置，Detour 将在每次运行后自动启动定位功能并恢复上次关闭软件时的位置；
- 顶部菜单栏，导航配置，保存，将当前程序配置保存为 conf.json，Detour 将自动加载该配置。

2.3.4 配置文件

2.3.2 中提到的导航配置，保存所生成的文件中包含所有 Detour 的配置信息，其中常用、重要的参数为：

- angleSgn、endAngle、rangeStartAngle、rangeEndAngle：雷达参数，含义见 2.3.1；
- mode：建图参数，0 为自动更新地图，1 为锁定地图；
- autoStart：软件参数，应设为 true，自动启动里程计和建图。

配置文件可手动修改，并直接拷贝至其他 Linux 或 Windows 上的 Detour 使用。

2.4 Clumsy

打开 ClumsyConsole.exe 之前应先启动 Medulla、Detour。在 Clumsy 中需要完成运动参数调试、智能识别调试、智能流程调试的工作。

2.4.1 配置文件和初次加载

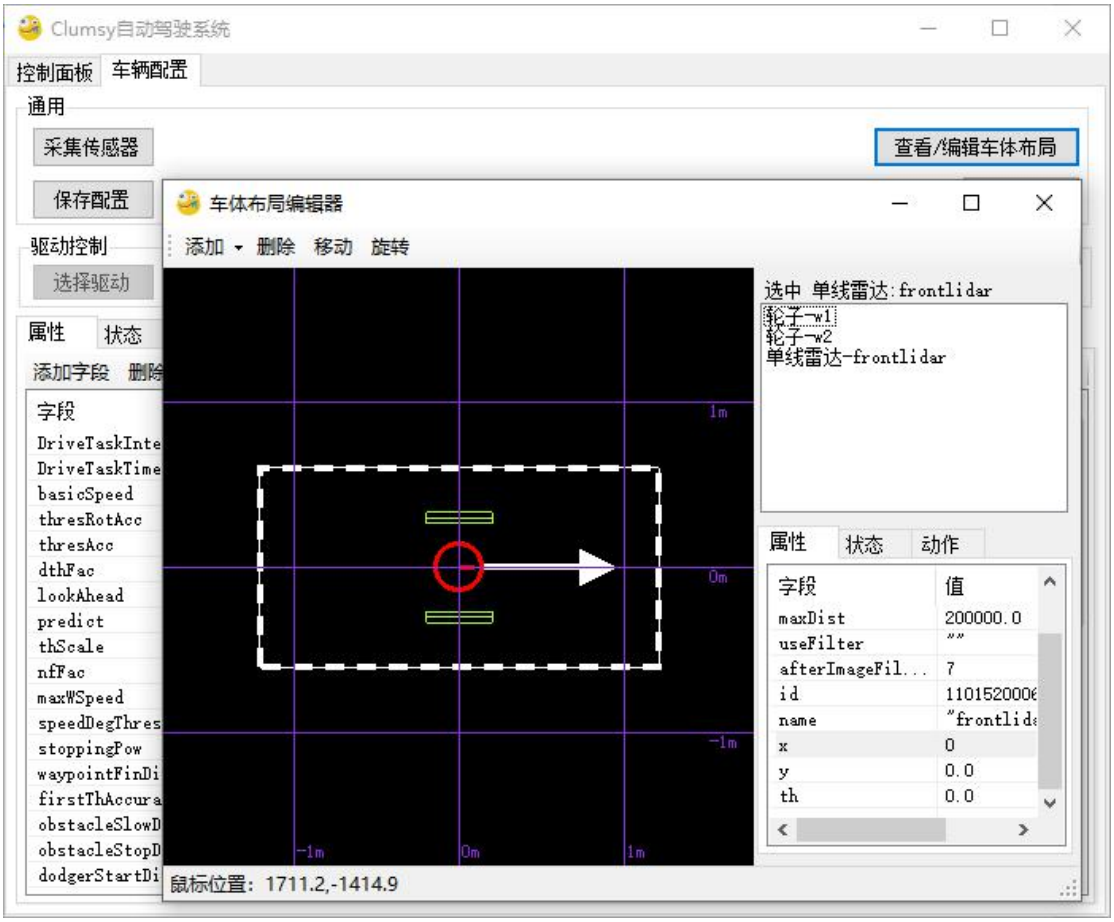
Clumsy 需加载配置文件和针对车型的 DLL 文件。初次加载时无配置文件，通过以下方式生成：如图 2.4.1—1 所示，点击“车辆配置”选项卡，点击“选择驱动”按钮，加载所需的 DLL 文件；点“保存配置”按钮，即可自动生成配置文件 conf.json；重启 Clumsy，将自动根据配置文件加载所需的 DLL 和参数值。

图 2.4.1—1

2.4.2 车体布局编辑器

选择“车辆配置”选项卡，点击“查看／编辑车体布局”按钮可打开车体布局编辑器窗口。在该窗口中，需对所有与智能动作相关的传感器在车体上的位置进行标定。例如，在 Medulla 中添加了名为 forklidar 的叉尖雷达用来检测托盘，则应在车体布局编辑器中，点击左上角的“添

加”，选择”单线雷达“，并左键单击界面将元件放置在车体上。选中相应的元件，在”属性“界面中将 name 更改为 Medulla 中对应的名称，例如”forklidar“。



需要对雷达在车体坐标系下的位姿(x,y,th)进行标定。具体的标定方法可参照第 5 章标定与校准。

2.4.3 动作测试

Clumsy 中需要对每个独立动作进行测试、调参，确保在可能的工况中，每种动作本身达到需要的一致性、精度、稳定性要求。

动作测试大致上分为以下 4 类：

a. **辅助标定的动作。**例如“舵轮到位测试”、“舵轮直线度校正”按钮，是用来对标定情况进行检测的。具体使用参照第 5 章标定与校准。

b. **基础动作。**包括前进和后退，以及叉车、举升车的上升下降等等。这两个动作是构成一切其他智能动作的基石。调试前进和后退时需要达到“小车从任意姿态能够尽可能快速的贴合路径”、“贴合路径后小车尽量稳定保持在路径上不抖动”。有时“贴合路径”与“贴合后稳定”是鱼和熊掌不可兼得：贴合路径越快可能导致贴合后越不稳定。可以在业务中的不同阶段使用不同的参数，例如在取货倒车速度比较低时，使用“贴合路径”更快的参数，而长距离倒车时，使用“贴合度”更好的参数。

涉及到运动的基础动作，通常是点击按钮后，在 UI 上点击拖拽出一个箭头，小车即会将箭头视为路径进行追踪。如图 2.4.3-1 所示。

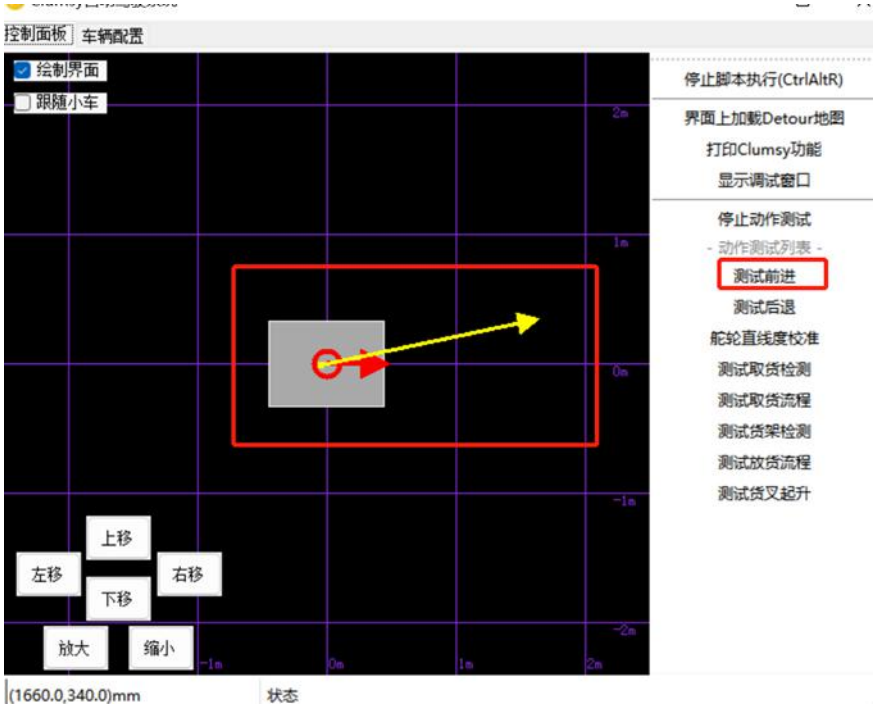


图 2.4.2-1

c. **智能检测算法。**例如“三腿托盘检测”、“二腿料架检测”等等。根据具体的技术要求，在不同角度、距离下测试智能检测算法。某些检测算法涉及到根据实际情况进行参数的调整。智能检测算法的调参请参照相应的手册。

智能检测算法往往是对某一特定模式进行识别，例如识别托盘、料架的轮廓结构。通常是点击按钮后，从被检测物大致位置触发，朝被检测物的正前方画一条箭头，算法会寻找复合指定位置和朝向的被检测物。如图 2.4.3-2 所示。

图 2.4.3-2

d. **智能动作流程。**智能动作流程是基础动作、智能检测算法、以及其他信号或交互的复合动作。例如智能取托盘流程：小车先沿指定路径倒车，并在倒车过程中同时调用托盘检测算法，在检测到托盘后，做减速/停车，对托盘位置进行确认后重新规划取托盘的精确路径，并沿新路径倒车直到触发撞板或离托盘达到指定距离，停车，举升，结束动作。

2.4.4 属性表、状态表

选择“车辆配置”选项卡，可以看到车辆的属性和状态标签页，如图 2.4.3-3 所示。

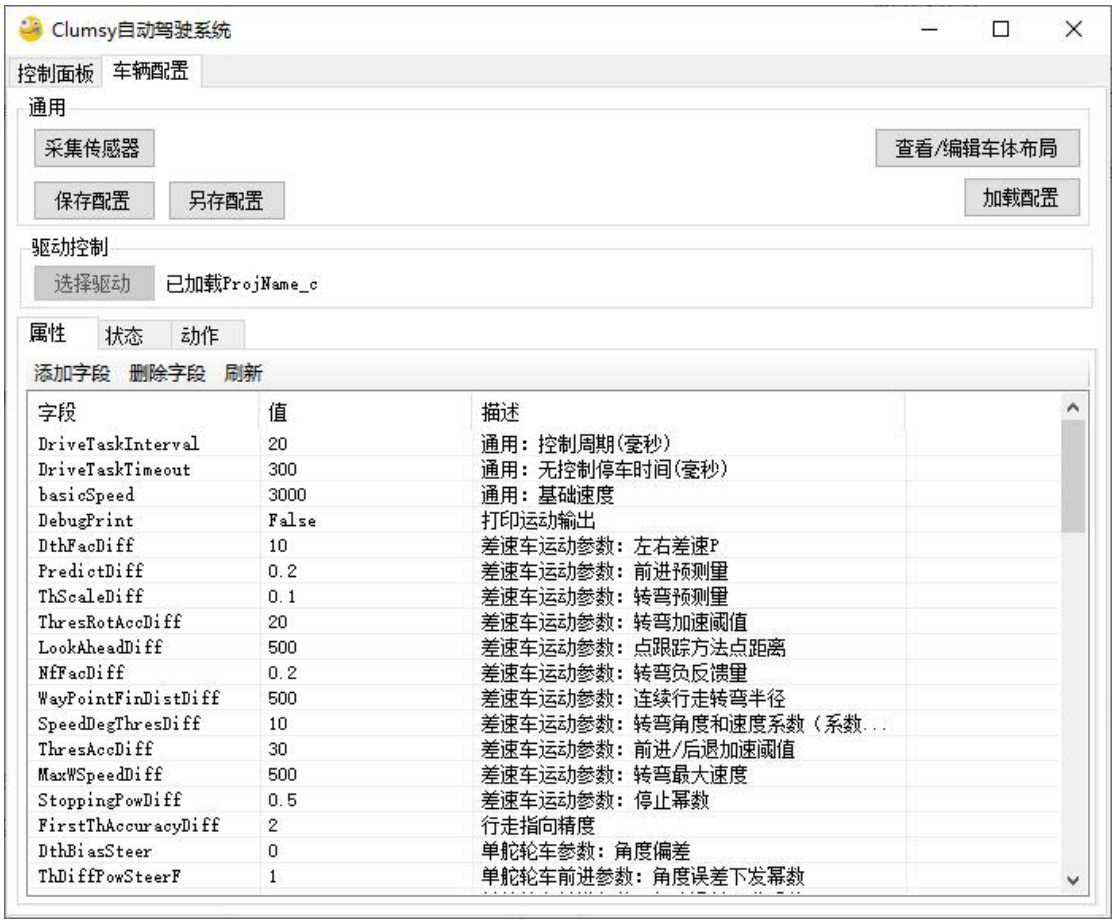


图 2.4.3-3

属性表：表示小车的配置参数。C#的基础数据类型（int、float、double、bool、string）均可作为属性值。属性有以下几种：

a. 车辆基本属性。例如小车运动速度（basicSpeed）、运动控制超时时间（DriveTaskTimeout）等。

b. 运动参数。运动控制相关的参数一般显示暴露在属性表中，方便使用人员调整以达到满足条件的运动状态。不同运动模型的调参方式请参见《MDCS 二次开发指南》附录 B。

c. 临时调试所用的变量。例如，车体上有两个雷达，可以使用一个变量来指定当前 Clumsy 调试时调用哪一颗。

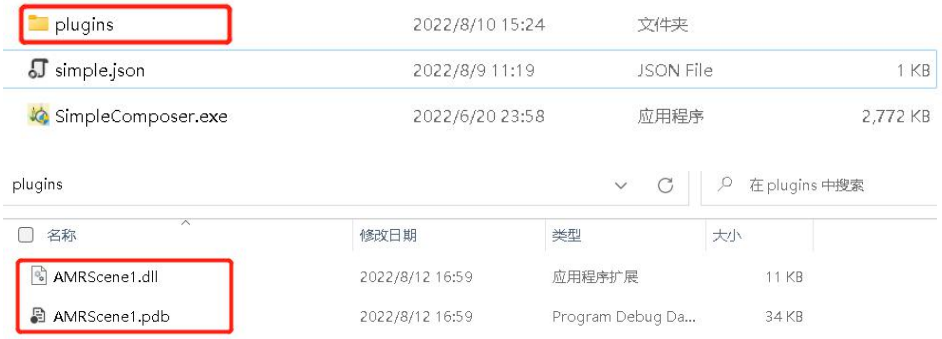
状态表：表示车辆状态的变量及其数值。即小车 ClumsyPilot 中定义的所有上下位信号。一般包括速度、角度、举升等等。

2.5 Simple

2.5.1 加载插件

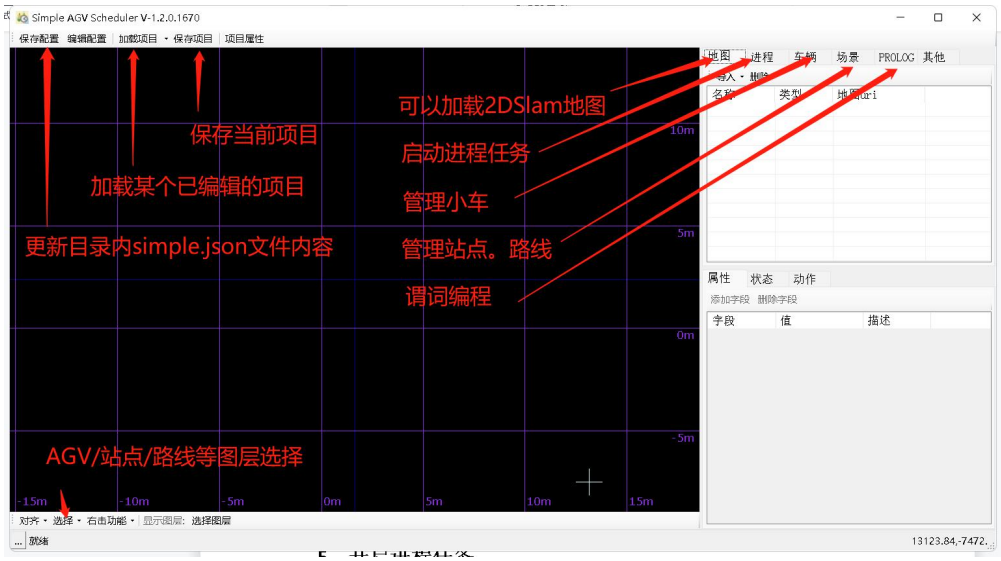
打开 Simple 文件夹，在 plugins/文件夹里面会有工程师提供的当前场景的插件（默认名称 AMRScene.dll），根据车型以及业务场景的不同，工程师会编写对应的插件，如果插件有更

新，替换 Simple/plugins 目录内 AMRScene.dll 即可

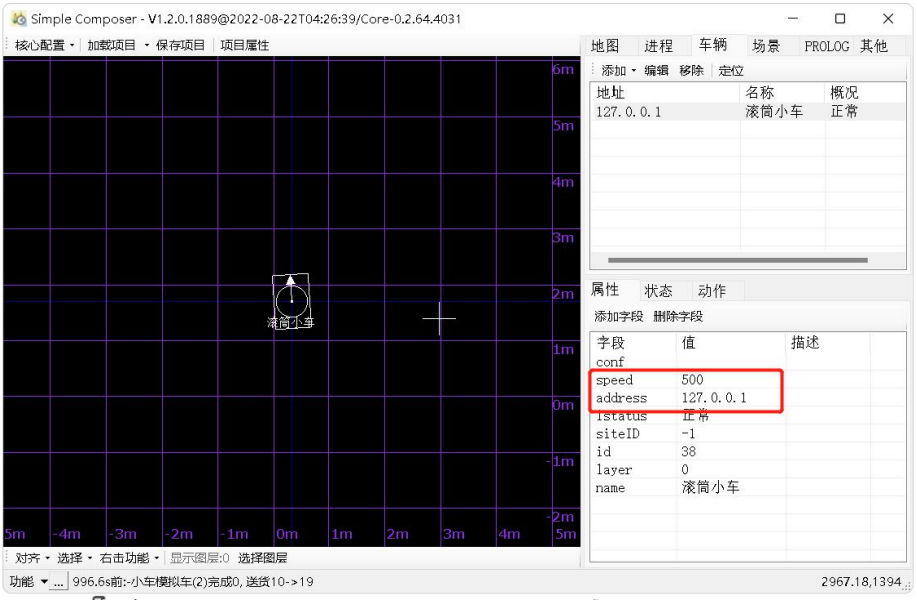


2.5.2 创建/配置小车

打开 SimpleComposer.exe，会看到一下界面



选择车辆—新建“xx 小车”，不要选择模拟车，然后配置该车的基本参数，选择小车，下面的属性里面填写 IP 地址(ipAddress)以及基础速度(speed)



2.5.3 创建站点/路径

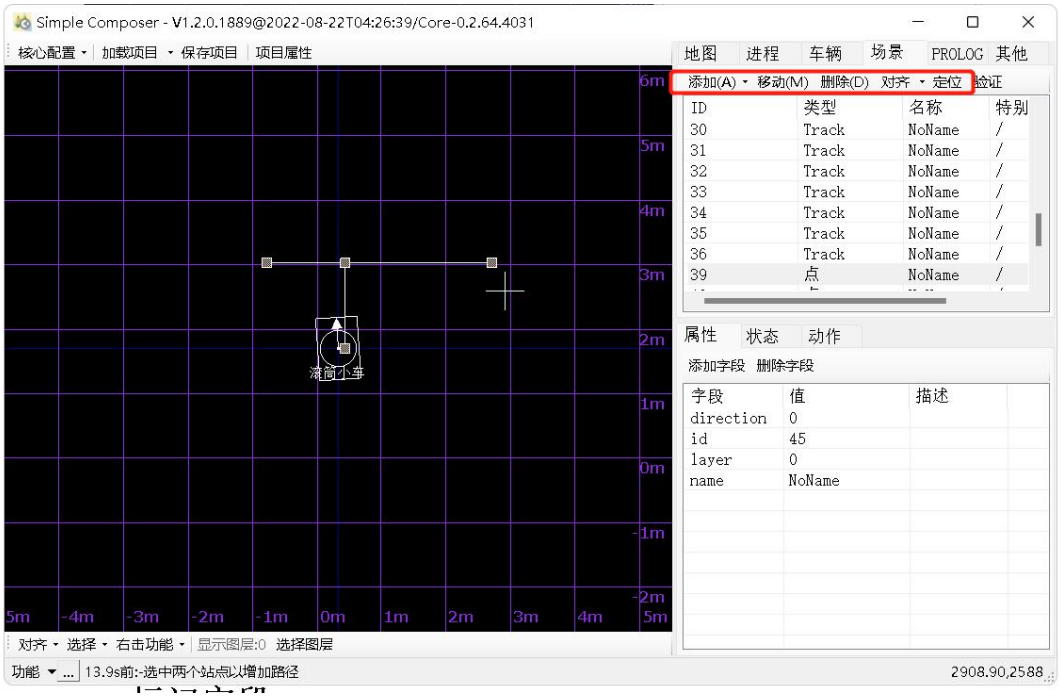
点击“场景”，添加“站点”创建工位，添加“路径”连接工位，也可使用快捷键 ALT+A 添加，然后按下 S 添加站点或 T 添加路径。

点选择可以看到点的属性，可以根据现场需求增加点的功能，选择线段可以看到线段属性，可以修改方向或者增加线段速度，点击添加字段，direction：1 表示正走，2 表示倒走，0 表示双向；speed：500 即可给线段附加 500r 的速度

若要使两点或多点平行/对齐，可以使用 CTRL 键或鼠标拖选两个或多个点，点击对齐→自动对齐

若要移动站点，选择站点，点击移动，鼠标左键按住拖动即可（可以在空的区域按住拖动，不需要点着需要移动的点进行拖动）

若要删除点，选择站点，点击删除即可，也可以选择多个点，删除

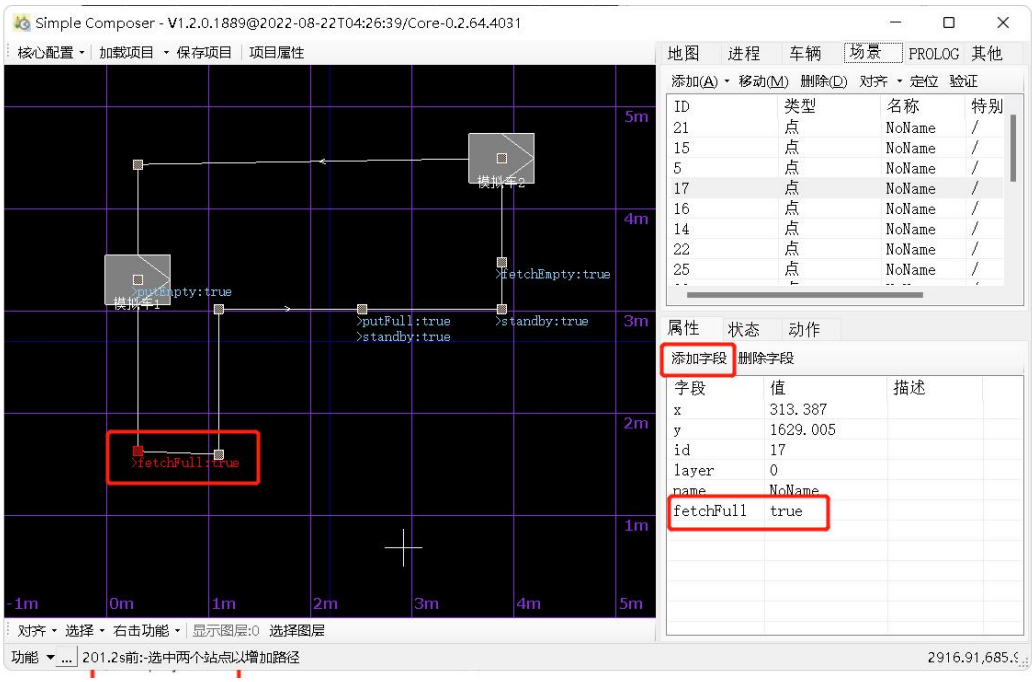


2.5.4 标记字段

可以根据现场需求增加点的功能，比如牵引上升下降，充电，辊筒取料，堆垛等等功能，只需要根据工程师提供的站点字段进行标记即可

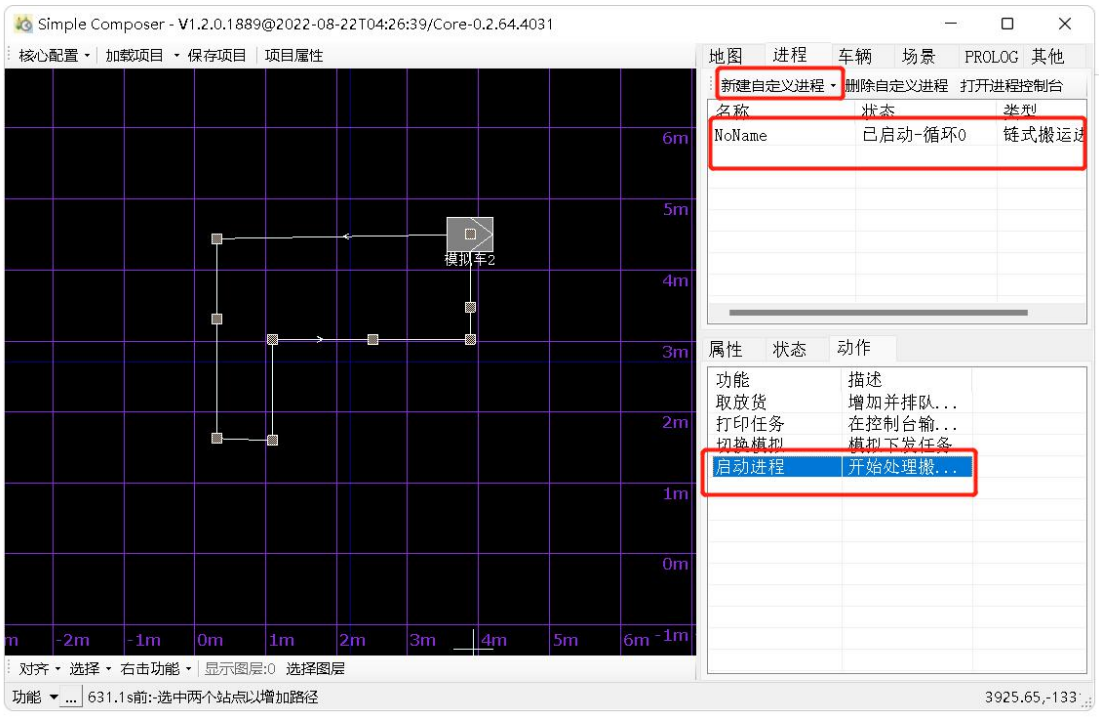
默认的调用小车功能方法（方法可以问工程师要）如：

- codeArrive：agv.Put();//到达点执行 Put 放料功能方法
- codeLevave：agv.Put();//离开点执行 Put 放料功能方法
- codeQueue：agv.Put();//立刻执行 Put 放料功能方法



2.5.5 开启进程任务

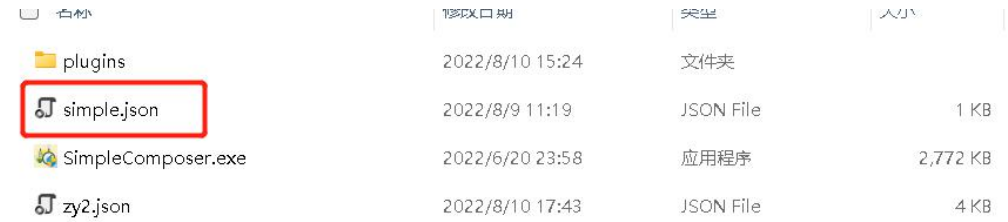
SimpleComposer 界面中选择进程，点击“新建自定义进程”添加需要开启的进程，双击启动进程即可



2.5.6 认识 simple.json 文件

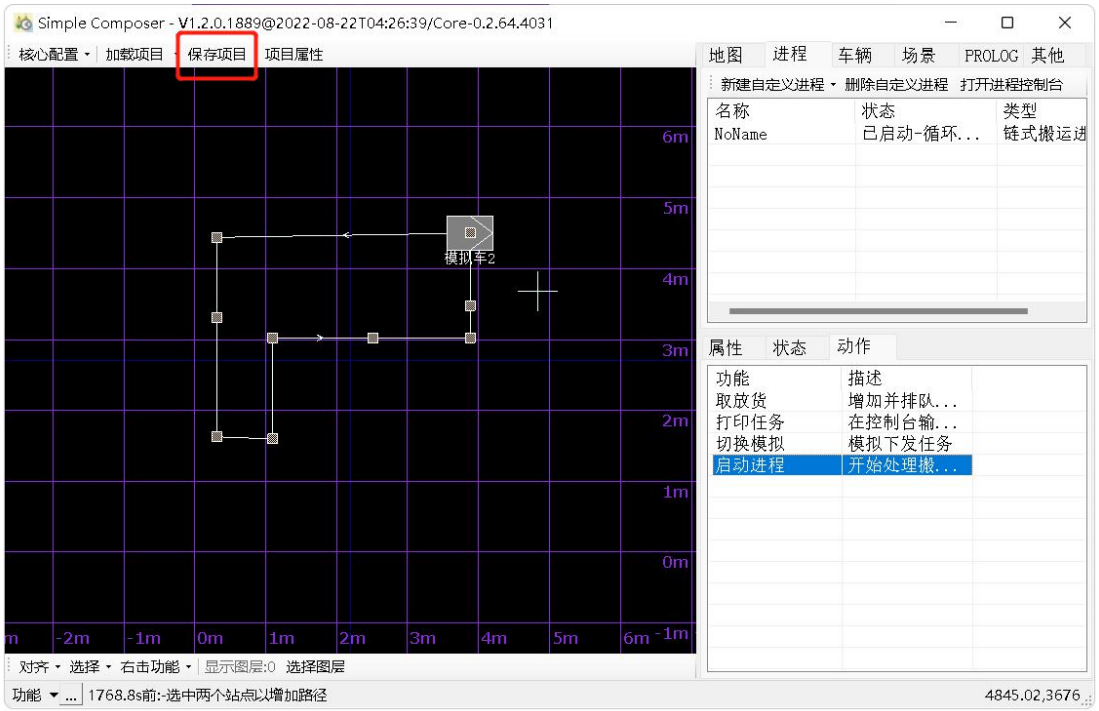
认识目录下的 simpel.json 配置文件（右击记事本打开），里面可以配置 Simple 自动加载的项目名称以及 Simple 所监听的端口和 IP，如果需要对某个项目进行自动加载，在 aotoload 字

段上填写对应的项目名称即可，注意保留后缀



2.5.7 保存项目

注意：保存项目和保持配置不一样，保存配置是保存 simple.json 配置文件，路径站点等信息需要新命名项目文件，如果需要加载已保存的项目，选择加载项目即可



3 Linux

本内容只写了区别于 windows 部分，其他内容与 windows 一样使用

3.1 目录结构和环境配置

环境配置：

在 Linux 系统，需要配置 mono 环境以运行 Medulla、Clumsy，需要配置 .Net 环境以运行 Detour。Linux 下配置环境的方式包括：使用懒书提供的预编译包直接部署、在线包管理器安装、以及源码编译三种方式。推荐使用第一种方式。

预编译包

$ mkdir MDCS && cd MDCS

将预编译包 env_setup.tar.gz 放入该文件夹

$ tar -zxvf env_setup.tar.gz

$ cd env_setup

$ chmod +x install.sh

根据系统架构选择部署的版本：

$ ./install.sh [x64/arm64]

出现以下提示说明安装成功：

$ dotnet successfully installed!

$ mono successfully installed!

在线包管理器安装与源码编译

这两种安装方式请参考以下链接：

.Net：在 Linux 发行版上安装 .NET - .NET | Microsoft Docs

Mono：Download - Stable | Mono (mono-project.com)

需要注意：.Net 应安装 5.0 版本，Mono 应安装 6.12 或更新的版本。

目录结构：

为了确保以下所有的配置和操作在任意 linux 系统中都适配，请按照以下要求构建文件夹保存 MDCS 文件。另外，Linux 系统中路径为“/”而非 Windows 下的”\”。

在任意位置创建 MDCS 文件夹，文件夹下创建 Medulla、Clumsy 文件夹，每个文件夹下分别存放对应软件的相关文件。MDCS 文件夹下创建 DetouLite 文件夹，DetourLite 文件下再创建 DetourLite 文件夹用以存放相关软件，同时创建 lib 文件夹用以存放 libOpenCvSharpExtern.so。

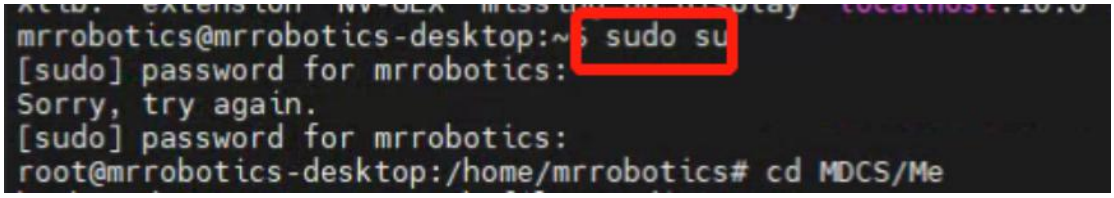
以下操作以文件夹创建在 home/username 为例，若创建在其他路径请自行修改对应路径。

MDCS/
- Medulla/
  - plugins/
    - CartActivator.dll
    - LidarController.dll
    - LessokajiDemo_m.dll
  - Medulla.exe
  - startup.iocmd
- DetourLite/
  - DetouLite/
    - DetourLite.dll
    - conf.json
    - mainmap.2dlm
  - lib/
    - libOpenCvSharpExtern.so
- Clumsy/
  - ClumsyConsole.exe
  - LessokajiDemo_c.dll
  - clumsy.json

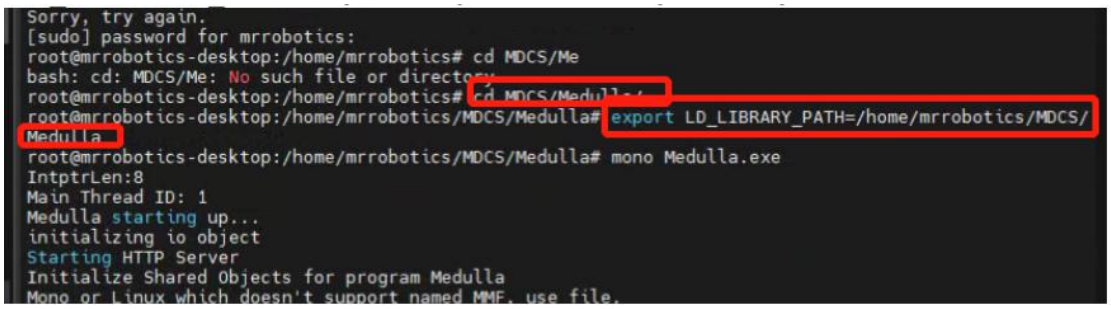
3.2 Medulla

startup.iocmd 的编写与 windows 下的一致，仅需修改路径的"\"为"/"。

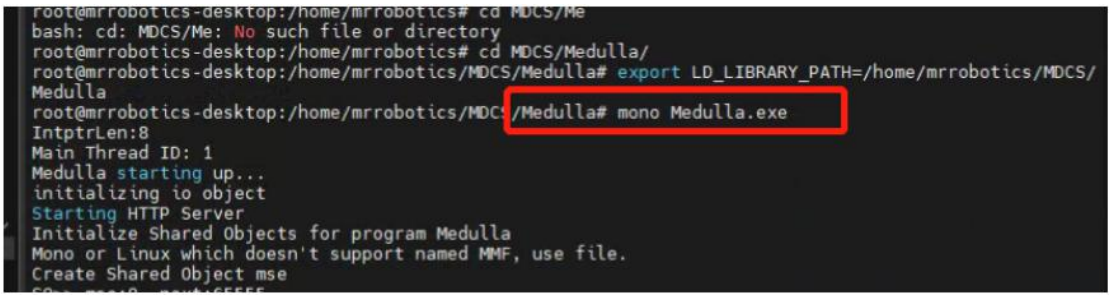
1．获取 root 权限



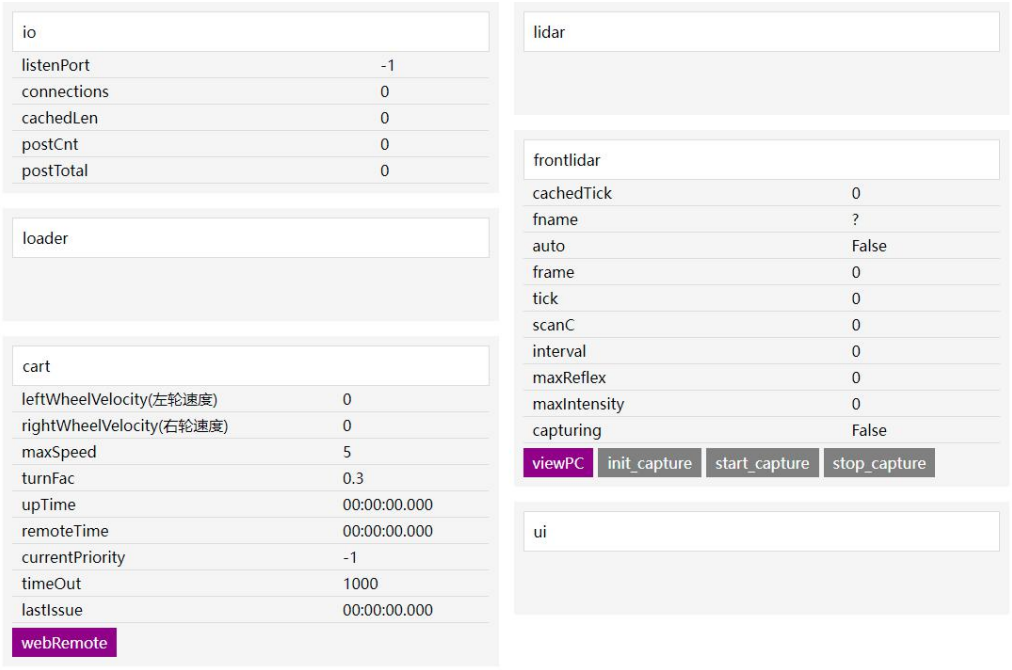
2．进入 MDCS/Medulla 目录 配置 Medulla 环境 export LD_LIBRARY_PATH=/home/username/MDCS/Medulla



3．mono 启动 Medulla



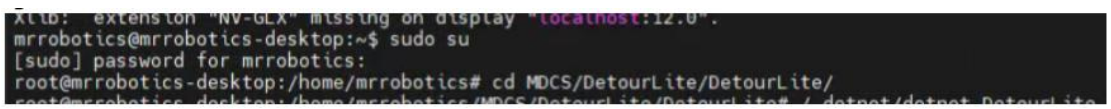
4．在 Linux 系统中不应该使用 WinMedulla.dll 插件，Linux 的用户界面通过浏览器访问。打开 Edge 浏览器，输入 http://IP:8007/simple.html 进入 medulla web 界面。（请确保使用 Edge 打开页面） 正确打开后将出现以下界面：



如上图所示，所有插件以卡片方式呈现，显示所有属性以及功能按钮。点击单线雷达的 viewPC 按钮可查看点云，在点云界面可以通过鼠标中键拖动点云，或是滚轮对点云进行放大缩小操作。

3.3 Detour

1．获取 root 权限，进入 cd MDCS/DetourLite 目录

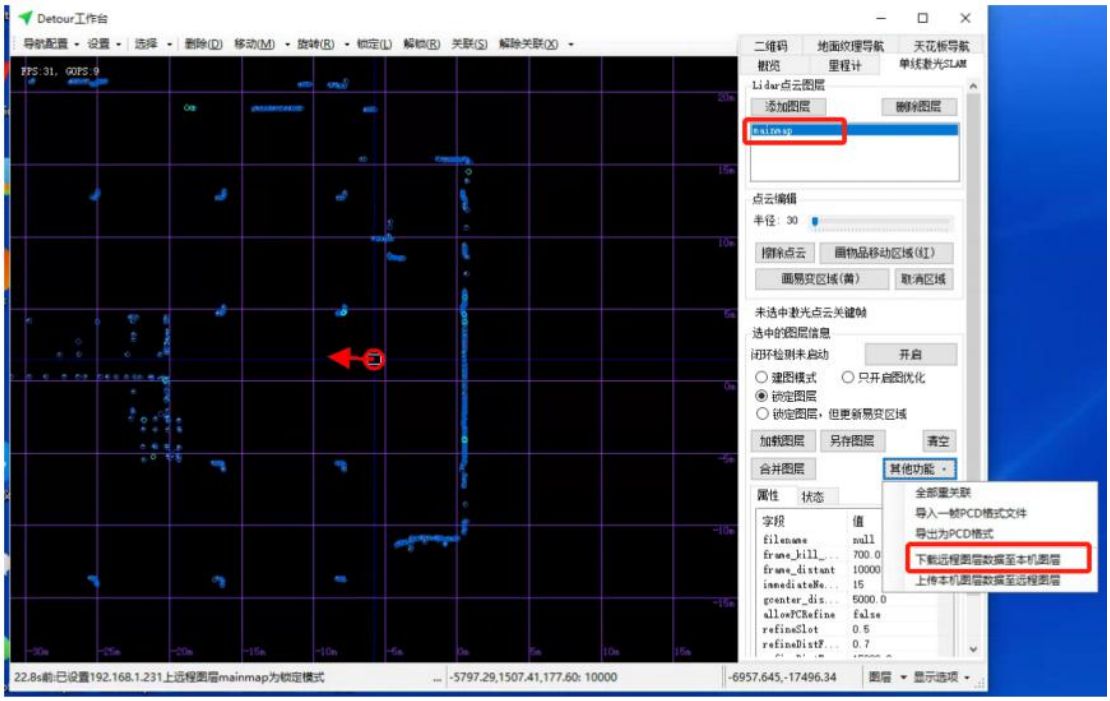


2．打开 detour 软件，输入~/.dotnet/dotnet DetourLite.dll

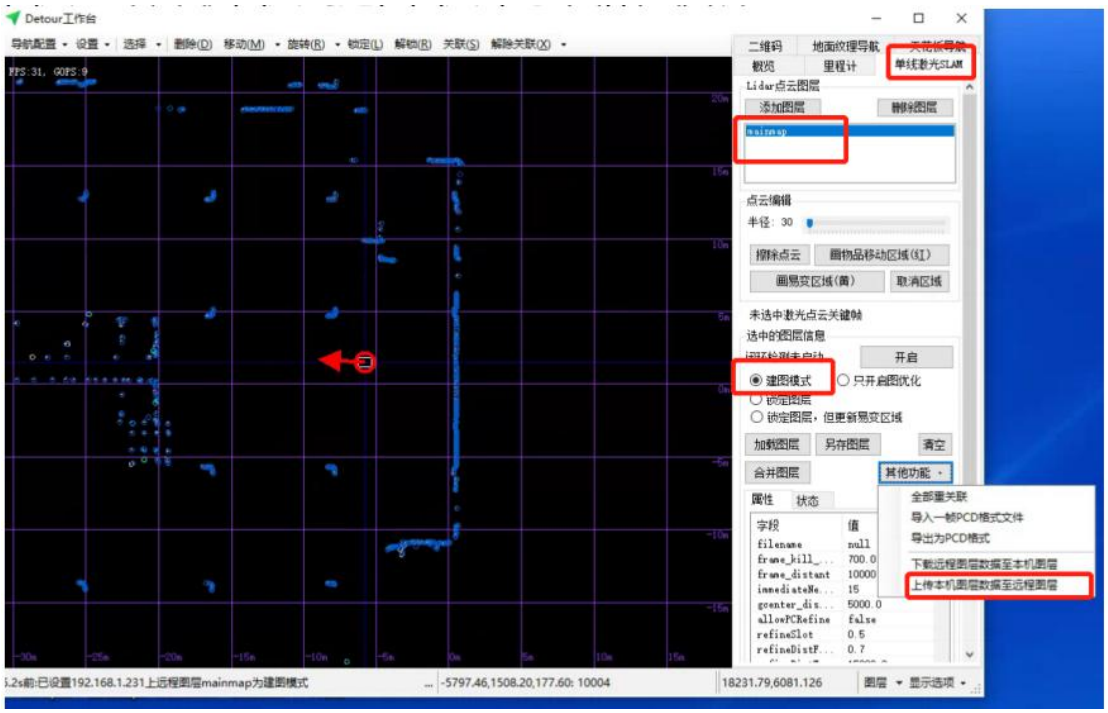
```
mrrobotics@mrrobotics-desktop:~$ sudo su
[sudo] password for mrrobotics:
root@mrrobotics-desktop:/home/mrrobotics# cd MDCS/DetourLite/DetourLite/
root@mrrobotics-desktop:/home/mrrobotics/MDCS/DetourLite/DetourLite# /.dotnet/dotnet DetourLite.dll
> Detour Lite Starting
> Running on linux core, preload library....
OpenCL platforms:

* OpenCL test failed, no OCL acceleration allowed
DetourCore by Lessokaji - 1.5.0.1018
Start time:20220317-093917
 * Device ID: BECF0A5BC3046B0B
init configuration...
Load config from conf.json
record last pos @ path =/home/mrrobotics/MDCS/DetourLite/DetourLite
 * Licence type: Test, activated by /, activate functions:DETOUR
Sensor [frontlidar] can capture
```

3．使用 windows 端的带可视化的 detour 软件远程连接车端 detour：选择概览—使用远程算法核—远程设备，输入 IP 点击连接

4．如果地图已经建好，请选择 单线激光 SIAM—选择图层—下载远程地图即可



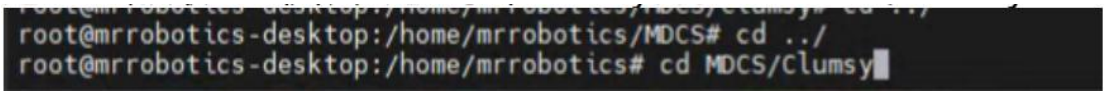
5．如果未建图，选择建图模式—遥控小车进行运行业务路线完成后，锁定图层—上传本机图层到远程图层即可



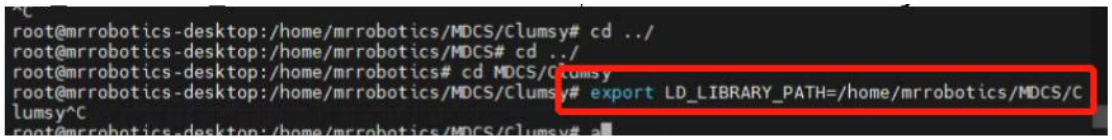
6．参考附录 C 中的 Detour 常用 URL 可以在车端的浏览器中调用 Detour 的各项接口，完成无可视化的 detour 的使用，具体参考附录 C。

3.4 Clumsy

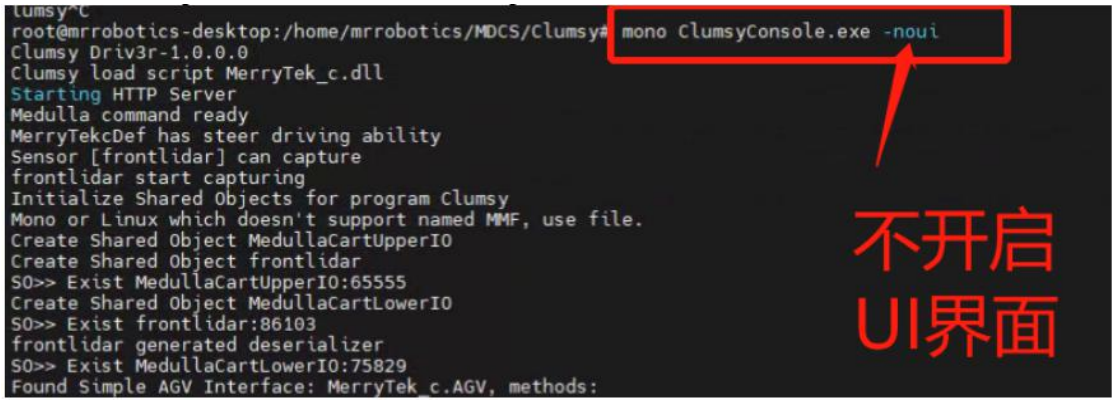
1．获取 root 权限，进入小车 Clumsy 目录 cd MDCS/Clumsy



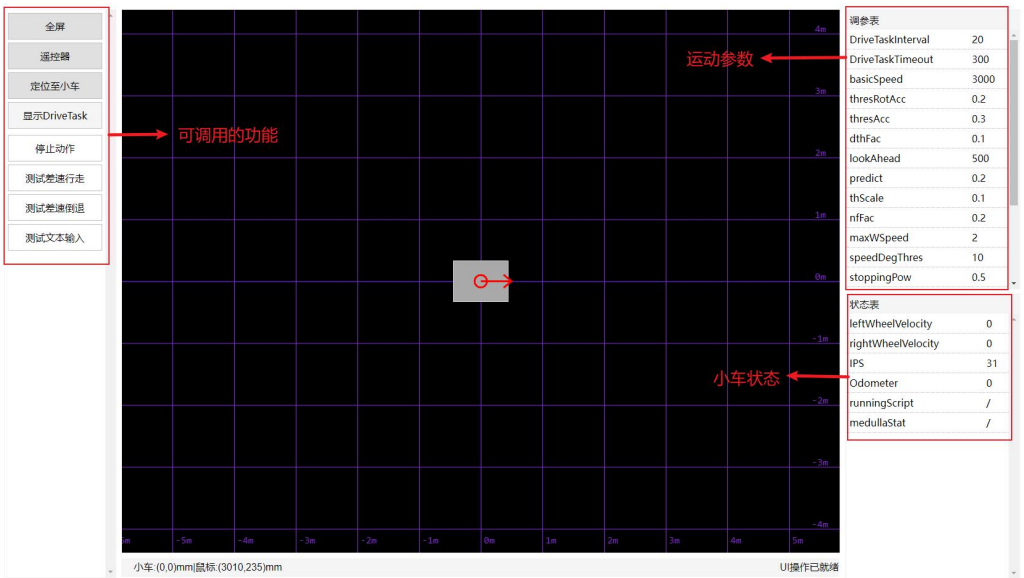
2．配置 Clumsy 运行环境 export LD_LIBRARY_PATH=/home/username/MDCS/Clumsy



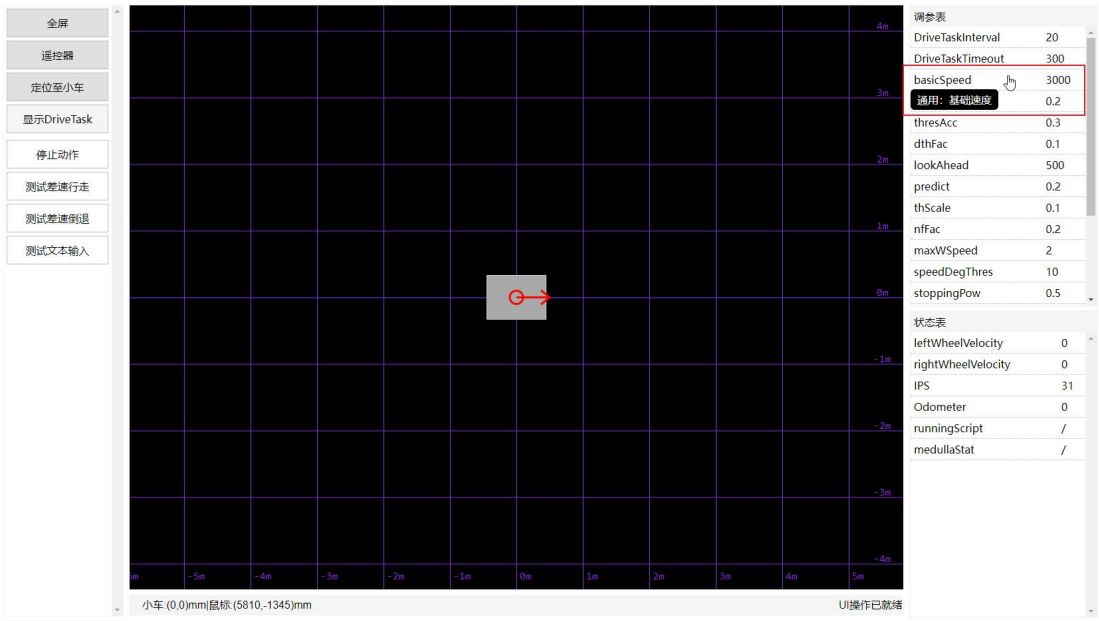
3．打开 Clumsy：输入 mono Clumsy.exe –noui



4．打开浏览器，输入 http://IP:8008/simple.html 进入 clusmy web 界面。正常打开将出现以下界面：



此处，运动参数和小车状态的具体条目可将鼠标移至每个条目上，会自动显示参数解释



4 使用规则和约定

a. MDCS：系统中所有输入输出角度均为角度制，范围约束在−180°到+180°，车体、雷达等局部坐标系下，正前方为 x 轴正方向，即 0 度线与 x 轴正半轴重合，角度逆时针方向增大；

b. Medulla：所有插件 DLL 文件放在 Medulla/plugins 中；

c. Medulla：车体插件应实例化为 cart，一般不使用其他命名；

d. Medulla：导航雷达插件应实例化为 frontlidar，一般不使用其他命名；

e. Medulla：货叉雷达插件应实例化为 forklidar，一般不使用其他命名；

f. Medulla：3D 相机插件应实例化为 cam3d，一般不使用其他命名；

g. Medulla：遥控器中控制车辆运动的**摇杆**或**四向键**名为 direction，一般不使用其他命名；

h. Medulla：遥控器中控制车辆运动速度的**滑钮**名为 speed，一般不使用其他命名；

i. Medulla：遥控器中控制货叉升降的**按钮**分别名为 up 和 down，一般不使用其他命名；

j. Medulla：遥控器中控制货叉升降速度的**滑钮**名为 liftSpeed，一般不使用其他命名。

5 标定与校准

5.1 单线雷达调平

车体上使用的所有雷达（包含导航雷达、货叉雷达等）均需要调至水平。首先，按下图方式，在小车四周立至少 4 个柱子，距离至少 5m，角度距离至少 60 度。

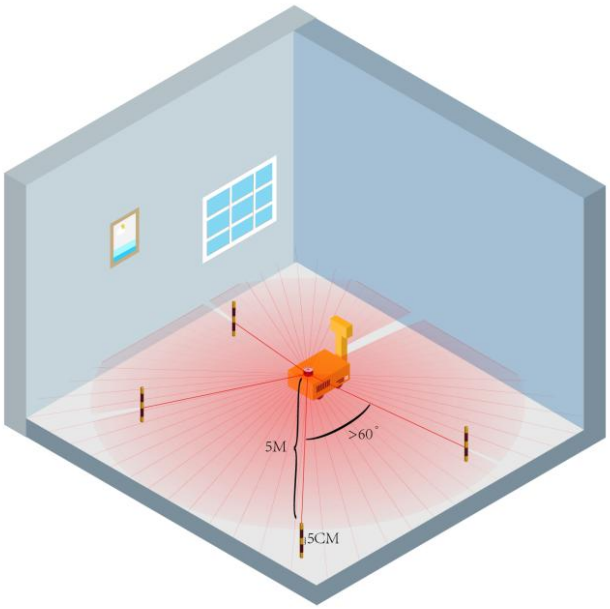


图 5.1-1

5.1.1 使用雷达的反射强度信息

在每个柱子等高位置贴上反光条，打开 Medulla 的 view 界面，能够同时看到所有柱子上的反光条（反光条处点云偏红色，其余反光度低处为白色点云），则雷达调平。

5.1.2 使用红外相机

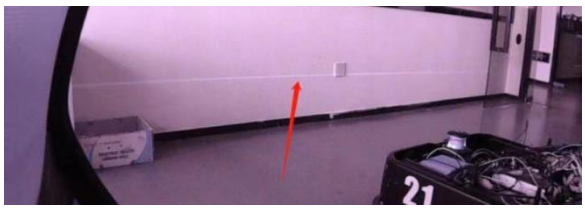


图 5.1.2-1

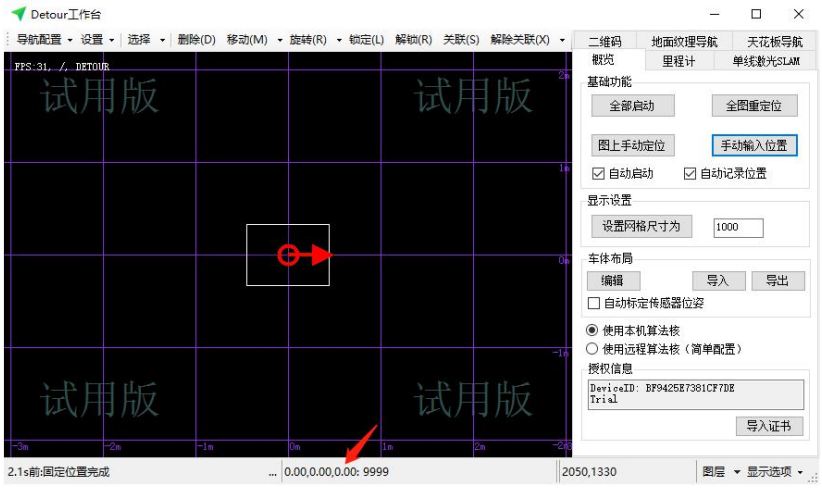
某些激光雷达不提供反射强度信息，可使用暗室和红外相机进行标定。在暗室中，激光雷达扫描到的位置可被红外相机观察到，如图 5.1.2-1 所示。在图 5.1-1 中每个柱子等高位置做标记，使用红外相机观察是否每个柱子上的标记被激光同时扫到，若能够同时扫到，则雷达调平。

5.2 舵轮角度校准

将车处于开阔且定位特征明显的环境中，确保有能够前进 10m 的空间。打开 Medulla、Detour、Clumsy 程序。若 Detour 已经建图，则将“单线激光 SLAM”标签页的图层删除。

确保校准舵轮角度之前，舵轮的 x、y 偏移量已经正确设置。

1. 在“概览”页面，点击“手动输入位置”，将定位设置为 0,0,0。
2. 在 Clumsy 上点击“舵轮直线度校正”按钮，车将依靠里程计前进 10m。
3. 应根据下图箭头处的第三个数值 th，对 Clumsy“车辆配置”标签页的 dthBias 进行调整。若 th 值为正，则应减小 dthBias；反之，增大 dthBias。
4. 重复上述步骤，直到 th 在 0 附近震荡，或 th 的偏差量在技术要求的范围内。



5.3 导航雷达角度校准

在校准导航雷达角度前，应先确保完成舵轮角度的校准。

将车处于开阔且定位特征明显的环境中，确保有能够前进 10m 的空间。打开 Medulla、Detour、Clumsy 程序。若 Detour 已经建图，则将“单线激光 SLAM”标签页的图层删除。

1. 在“概览”页面，点击“手动输入位置”，将定位设置为 0,0,0。
2. 在 Clumsy 上点击“舵轮直线度校正”按钮，车将依靠里程计前进 10m。
3. 应根据上图箭头处的第二个数值 y，对 Detour“车体布局”，“编辑”页面激光雷达的 th 进行调整。若 y 值为正，则应减小 th；反之，增大 th。
4. 重复上述步骤，直到 y 在 0 附近震荡，或 y 的偏差量在技术要求的范围内。

5.4 单线激光雷达取托盘标定

1. 首先将叉车的货叉放在托盘所在高度，并将托盘放置于货叉正前方的标定位置。可先将叉车完全叉入托盘中，再直线退出。

2. 将货叉高度调整到激光雷达能看到待检测固定参照物的高度。打开“车辆配置”标签页的“车体布局编辑器”页面，若发现点云残缺的情况，请检查叉尖雷达的安装是否歪斜，或者货叉是否没有下降到下限位，或者将托盘稍微远离叉车。点云完整后进入下一步。图 5.4-1 情况即为常见三腿托盘点云残缺的情况。

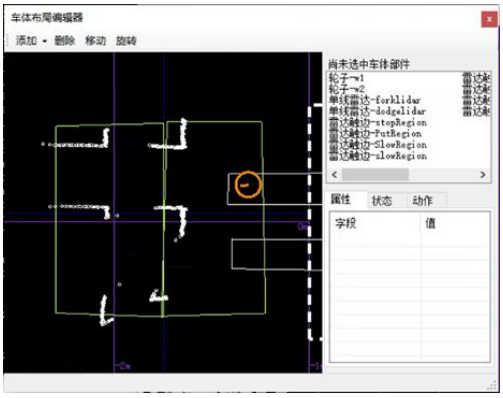


图 5.4-1

3. 在 Clumsy 界面上选择相应的检测测试按钮。当前以二腿料架为例。在界面上选择“检测测试:二腿料架”，并在 UI 界面上画一箭头开启检测。该箭头起点应在待检测物的对称中心位置附近，箭头方向从物体本身指向该物体面朝小车一面的正前方，如图 5.4-2 所示。

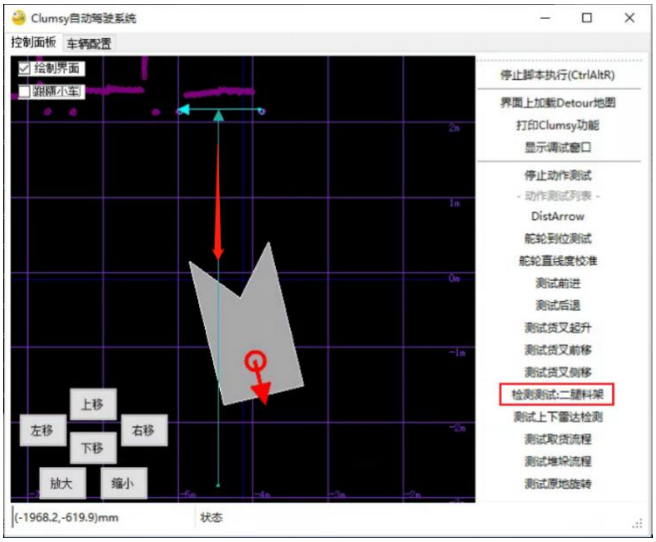


图 5.4-2

4. 若检测成功，应该在界面上看见指向该待检测物体的青色箭头，见图 5.4-2，且控制台上打印待检测物在车体坐标系下的位姿 Car(x,y,th)，以及世界坐标系下的位姿

World(x,y,th)，见图 5.4-3。检测成功则进入下一步，失败则请调整识别算法参数或咨询识别算法开发人员。

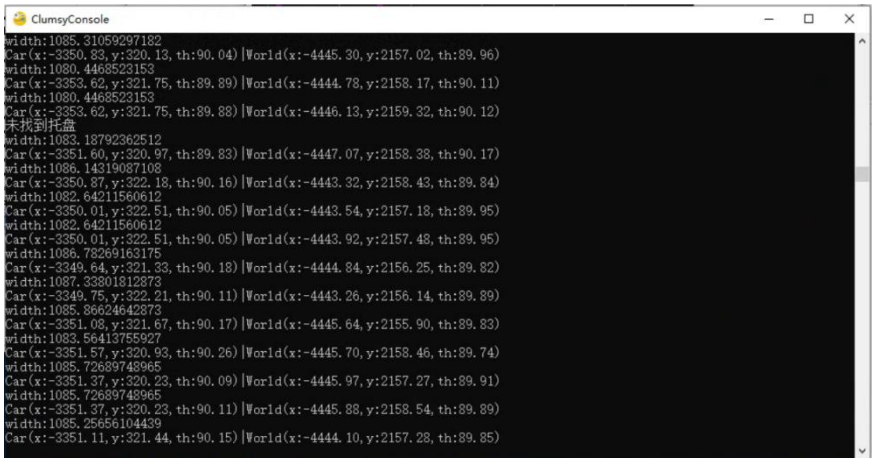


图 5.4-3

5. 标定雷达的 y 和 th。标定时需选定车体部件“单线雷达-forklidar”。双击属性标签页的 th 字段进行更改，使得步骤 4 中控制台打印的角度 Car.th 在 180 度附近震荡。后双击属性标签页的 y 字段进行更改，使得上一步中控制台打印的横向偏移量 Car.y 在 0 附近震荡。

6. 标定雷达的 x。雷达的 x 坐标仅影响侧取，若无侧取需求，可直接使用设计图纸上的数值，而无需对 x 进一步标定。标定前请先将车头方向朝向-90 度，且托盘放置于货叉的正前方，此时步骤 4 中显示的 World.x 应与小车的 x 坐标基本一致，将该 x 坐标值记为 δ。将小车调整至车头朝向 0 度的方向，此时调整雷达的 x 字段，使得步骤 4 中控制台的 World.x 显示在 δ 附近震荡，即完成 x 标定。

7. x，y，th 字段值调整完毕后，关闭“车体布局编辑器”，点击“保存配置”按钮。

8. 实车测试。返回 Clumsy 的“控制面板”标签页。点击“测试取货流程”，等待 UI 上出现紫色点云后，在地图上用鼠标左键拖曳出大致指向托盘的取货路径。松开鼠标后，车将自动完成取货操作。如图 5.4-4 所示。若观察到有偏差，请重复上述步骤进行标定。

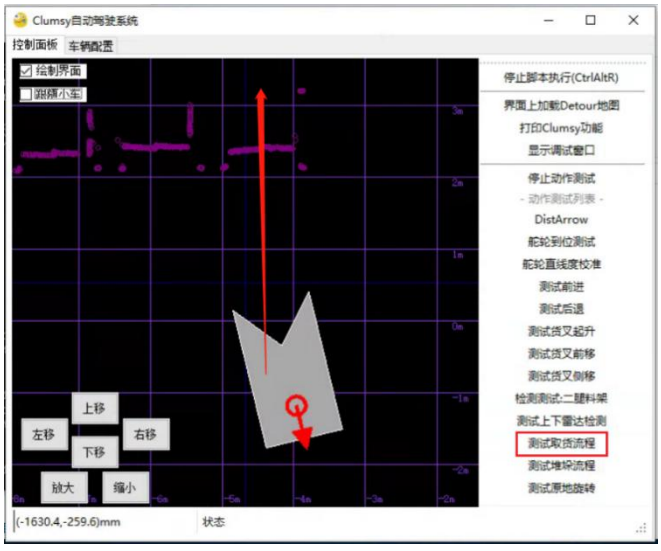


图 5.4-4

5.5 堆垛标定

1. 约定：下层料框检测所用的雷达为 forklidar，上层料框检测所用雷达为 forklidarUp。默认 forklidar 已经按照 5.4 所述流程完成了标定。
2. 将两层料框摆放至堆叠完成的状态，将货叉叉入上层料框，并将其举升至 forklidar 可看见下层料框固定参照物的位置。
3. 与 5.4 中步骤 3、4 所属方式相同，使用下雷达 forklidar 的检测算法，记控制台输出的 Car(x,y,th)为δx，δy，δth。
4. 使用上雷达 forklidarUp 的检测算法。调整 forklidarUp 的 th，使得当前控制台输出的 Car.th 在δth 附近震荡；再调整 forklidarUp 的 x、y，使得当前控制台输出的 Car.x、Car.y 分别在δx、δy 附近震荡。
5. 尝试进行堆垛。若有系统性堆垛不准，请调整 Clumsy 参数表中的 StackBiasX、StackBiasY、StackBiasTh。三者分别对应货叉前移、货叉横移、小车旋转角度的偏差的补偿量。

6 其他事项

6.1 Medulla 数据采集与回放

6.1.1 数据采集

6.1.2 数据回放

6.1.3 播放 rosbag 数据

目录结构

```
MDCS/
    Medulla/
        plugins/
            LidarController.dll
            RosLidar.dll
        Medulla.exe
        startup.iocmd
```

启动 rosbridge 服务：

在 Linux 上运行：

```
$ roslaunch rosbridge_server rosbridge_websocket.launch
```

配置 startup.iocmd 文件：

```
frontlidar = io load plugins/RosLidar.dll
frontlidar InitReadLidar ws://xxx.xxx.xxx.xxx:9090
```

其中 xxx.xxx.xxx.xxx 替换为 rosbridge 服务所在的 IP 地址，9090 为 rosbridge 服务默认端口。

若在 Windows 上打开 Medulla，可打开 UI 界面：

```
ui = io load plugins\WinMedulla.dll
ui Show
```

启动 Medulla：

可在 Windows 或 Linux 上启动 Medulla 程序。

播放 rosbag：

在 Linux 上运行：

```
$ rosbag play filename.bag --topic /scan_orig
```

其中 filename.bag 为待读取的数据包。

6.2 配置开机自启动

6.3 常见问题

使用指南附录

附录 A 单线雷达的通用命令

命令	参数	说明
start	string addr	启动雷达，addr 为 IP 地址或串口号等
setReflexRange	float rng	设置最大反光强度为 rng
setMaskDist	float minD，float maxD	设置雷达仅使用[minD，maxD]距离范围内的点云
setMask	int start，int end	设置雷达使用[start，end]范围内的点云
setBias	float bias	设置雷达点云整体逆时针偏转 bias 度
setMirror	bool m	若 m 为 true 则将点云镜像翻转

示例：

frontlidar setReflexRange 255

frontlidar setMaskDist 1000 5000

frontlidar setMirror true

附录 B 支持的单线雷达信息表

雷达角度为−180°~+180°，正前方为 x 轴正方向

angleSgn：扫描方向，1 为逆时针，−1 为顺时针

endAngle：一帧扫描的结束角度（−180°~+180°）

rangeStart：扫描起始角度（−180°~+180°）

rangeEnd：扫描结束角度（−180°~+180°）

型号	angleSgn	endAngle	rangeStart	rangeEnd
Se1035	1	135	−135	135
LDS−E310−E	1	0	−180	180
WLR716MiniLidar	1	135	−135	135

附录 C Detour 常用 URL

Detour接口　http://{ip}:4321/

接口	参数	说明	返回
/pause		里程计暂停运行	json，格式为{success: true}
/resume		里程计恢复运行	json，格式为{success: true}
/setLidar2DOdometryMask	odometry：string，所设置的里程计名称 字符串：一串 XY 坐标点构成的 json 字符串	设置激光雷达的无效范围，在一系列点构成的闭包范围内的激光点将被忽略，这些点的坐标为车体坐标系下的值	json，格式为 设置成功：{success: true} 设置失败：{error="No odometry named {指定里程计名称}"} **示例：** http://127.0.0.1:4321/setLidar2DOdometryMask?odometry=odometry_1&[{"X":123,"Y":456},{"X":123,"Y":456},{"X":123,"Y":456}]
/debug		将程序设置为 debug 模式运行，将输出 debug 日志文件	json，格式为{success: true}
/getSensors		获取激光雷达当前的扫描数据，包括对应设备在世界坐标系下的位姿	byte[]，格式参见示例程序
/getPos		获取当前车体在世界坐标系下的位姿	json，格式为{x: float,y: float,th: float,l_step: int,tick: long}
/getConf		获取程序的配置参数	json，格式参见示例程序
/switchSLAMMode	update：bool，是否更新地图（前建图） name：string，指定地图名称。不指定该参数则对所有图层生效 dynamic：bool，是否图优化，仅在 update 为 false 时生效。地面纹理导航暂不支持该项	切换 SLAM 的运行模式	json，格式为{performed: true}
/switchPosMatch	disabled：bool，是否关闭地图匹配 name：string，指定地图名称。不指定该参数则对所有图层生效	切换是否进行地图匹配	json，格式为{performed: true}
/switchPosMatch	disabled：bool，是否关闭地图匹配 name：string，指定地图名称。不指定该参数则对所有图层生效	切换是否进行地图匹配	json，格式为{performed: true}
/relocalize		进行一次全局匹配	json，格式为{performed: true}
/getStat		获取程序运行状态，包括里程计状态、地图状态等	json，格式参见示例程序
/loadMap	name：string，指定的图层名称，默认为 mainmapfn：string，指定加载的地图文件名称，默认为 mainmap.2dim	使远端算法核加载远端已保存的某个地图	json，格式为{performed: true}
/saveMap	name：string，指定的图层名称，默认为 mainmap fn：string，指定保存的地图文件名称，默认为 mainmap.2dim	使远端算法核将某指定图层当前的建图结果保存至文件	json，格式为{performed: true}
/uploadRes	fn：string，上传到远端后文件的命名 bytes：byte[]，待上传数据	以字节数组的方式，向远端算法核上传文件	json，格式为{performed: true}
/downloadRes	fn：string，需下载文件的文件名	以字节数组的方式，将远端算法核的文件下载至本地	byte[]
/setLocation	x：float，x 坐标值 y：float，y 坐标值 th：float，角度值	手动设定车体当前在世界坐标系下的位姿	json，内容为重新定位后的车体位姿，格式与/getPos 相同

附录 D Clumsy 常用 URL

第二部分 MDCS 二次开发指南

0 预备知识

本章节中将介绍与移动机器人开发相关的计算机知识、数学知识、和移动机器人开发know-how。包括：计算机数据表示，平面坐标系变换，一般开发流程介绍。

对相关概念熟悉的读者可跳过本章。

0.1 计算机数据表示

请自行百度以下知识：

- 大小端
- 字节表示
- 位运算
- 移位运算

0.2 平面坐标系变换

0.2.1 坐标系定义

不论是移动机器人算法开发，还是项目落地，坐标系变换是解决所有问题的先决要素之一。整个 MDCS 系统统一使用**二维右手坐标系，即 x 轴向右，y 轴向上，角度 th 逆时针增大（0 度方向与 x 轴正方向重合，取值范围为-180 到 180 度）**，如图 0.2-1 所示。MDCS 的坐标系距离单位是毫米，旋转单位为角度制表示。

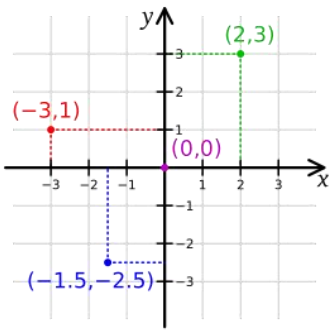


图 0.2-1

我们定义：

- 传感器坐标系：以传感器所在位置为坐标原点、传感器正前方为 x 轴正方向的右手坐标系（Medulla 传感器的 view 界面即为传感器坐标系）。
- 车体坐标系：以车体所在位置为坐标原点、车体正前方为 x 轴正方向的右手坐标系（Clumsy 车体编辑器界面即为车体坐标系）。
- 世界坐标系：在地图上的二维右手坐标系。

0.2.2 坐标系转换

坐标系转换要解决的问题本质上非常简单：同一个物体在不同坐标系下的坐标是什么关系；或者，已知观测的坐标，求坐标系的位置。

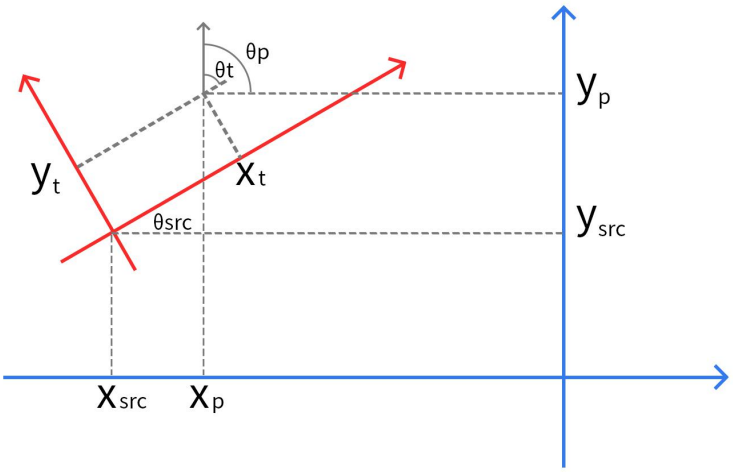


图 0.2.2-1

常见的坐标系转换问题有以下 3 种。

问题 I：如图 0.2.2-1 所示，已知某点 p 在 A 坐标系下的位姿为 Tpa=(xt,yt,θt)，A 坐标系在 B 坐标系下的位姿为 Tab=(xsrc,ysrc,θsrc)。求点 p 在 B 坐标系下的位姿 Tpb。

在 B 坐标系下，p 的位姿为：

xp=xt*cos(θsrc)−yt*sin(θsrc)+xsrc

yp=xt*sin(θsrc)+yt*cos(θsrc)+ysrc

θp=θsrc+θt

Tpb=(xp,yp,θp)

这个抽象问题的常见实例是：我们已知某个托盘在雷达坐标系下位姿为 Tpa，雷达在车体上的位姿为 Tab，Tpb 即为托盘在车体坐标系下的位姿。

代码中可使用 LessMath.Transform2D(src，t)求解该问题，其中 src 为 Tab，t 为 Tpa，返回值为 Tpb。

问题 II：如图 0.2.2-1 所示，已知某点 p 在 B 坐标系下的位姿为 Tpb=(xp,yp,θp)，A 坐标系在 B 坐标系下的位姿为 Tab=(xsrc,ysrc,θsrc)。求点 p 在 A 坐标系下的位姿 Tpa。

在 A 坐标系下，p 的位姿为：

xt=(xp−xsrc)*cos(θsrc)+(yp−ysrc)*sin(θsrc)

yt=−(xp−xsrc)*sin(θsrc)+(yp−ysrc)*cos(θsrc)

θt=θp−θsrc

Tpa=(xt,yt,θt)

这个抽象问题的常见实例是：我们已知某个托盘在世界坐标系下的位姿为 Tpb，车体在世界坐标系下的位姿为 Tab，Tpa 即为雷达在车体坐标系下的位姿。

代码中可使用 LessMath.SolveTransform2D(src，dest)求解该问题，其中 src 为 Tab，t 为 Tpa，返回值为 Tpb。

问题 III：如图 0.2.2-1 所示，已知某点 p 在 A 坐标系下的位姿为 Tpa=(xt,yt,θt)，点 p 在 B 坐标系下的位姿为 Tpb=(xp,yp,θp)。求 A 坐标系在 B 坐标系下的位姿为 Tab。

在 B 坐标系下，A 坐标系的位姿为：

θsrc=θp-θt

xsrc=xp-xt*cos(θsrc)+yt*sin(θsrc)

ysrc=yp-xt*sin(θsrc)-yt*cos(θsrc)

Tab=(xsrc,ysrc,θsrc)

这个抽象问题的常见实例是：我们已知托盘在雷达坐标系下位姿为 Tpa，托盘在车体坐标系下的位姿为 Tpb，Tab 即为雷达在车体坐标系下的位姿。

代码中可使用 LessMath.ReverseTransform(dest，t)求解该问题，其中 dest 为 Tpb，t 为 Tpa，返回值为 Tab。

0.3 一般开发流程

一般来说，MDCS 部署之前要求提供完全具备软件调试条件的硬件。测试前需要具备的条件指的是：

- 所有的报文经过了检验，确保地址、大小端等均正确。一般认为电气部门或客户提供的接口文档已经确认过。如发现报文收发无效的问题，请根据协议的种类，选择使用串口助手、网络助手、或者官方提供的上位机程序进行检验。若该类助手能正确使用，则说明是软件层面的问题；反之，应请电气部门或客户进行检查。
- 所有雷达经过了调平。具体要求见《MDCS 使用指南》5.1。一般认为机械部门或客户已做过确认。一般通过肉眼观察点云，可以看出是否调平。发现未调平好的情况，请机械部门或客户进行整改。

从零开始的开发步骤大致如下：

1. 安装 Visual Studio 2022，安装时选择 C#桌面开发环境。
2. 安装 ReSharper（该步可选）。
3. 将引用文件和新建的 VS 项目按照下一章中的目录结构进行排布，添加 MedullaAdapter、ClumsyPilot、AMRScene 的开发模板，并将 ProjName 改为实际的项目编号或车型代号。
4. 在 MedullaAdapter 的 ProjName.Cart 中按照需求增删各种信号变量，并在 Init()函数中初始化需要使用的通讯类（常用的通信类已封装在 MDCSToolBox.dll）。
5. 在 MedullaAdapter 的 MotorRoutine 中对接驱动报文的实际收发。
6. 在 MedullaAdapter 的 IORoutine 中对接 IO 报文的实际收发。
7. 若有其他报文种类，在其他 Routine 中分别进行对接。
8. 用遥控器检查报文适配是否实现正确，小车行走是否正常。
9. 将要使用的信号从 MedullaAdapter 中复制到 ClumsyPilot 的 ProjNamecDef.cs 中。
10. 根据需要，在 ClumsyPilot 的 Movements 中实现各种基础的动作，例如行走、举升、自

旋等，并用 MovementTest 按钮进行测试。此步骤中可复查舵轮、雷达的角度是否有偏，具体参考《使用指南》第 5 章。

11. 根据需要，在在 ClumsyPilot 的 Movements 中实现各种智能动作，例如自动取托盘、自动进工位等，并用 MovementTest 按钮进行测试。注意，涉及检测算法的，应该先测试检测算法本身，然后对相关传感器调平，再测试智能动作流程。

12. 将业务涉及到的动作（包括站点和路径动作）封装在 ClumsyPilot 的 AGV.cs 中。

13. 在 AMRScene 的 ProjName.Car 中实现 TemplateTrackCoder 和 TemplateSiteCoder，实现“点线”路径序列向实际动作的转换。

14. 可以在 AMRScene 的 ProjNameCar.cs 中用 MethodMember 编写一个简单的 Demo 按钮。

1 开发规范

需要注意！！！《MDCS 二次开发指南》中所有“ProjName”在实际部署时应替换为具体的项目名称或者车型代号，例如将“ProjName”替换为“FG22XXXXX”。

开发环境

必装：visual stuido 2022 C#桌面开发包

选装：resharper

目录结构

```
CartAdapterProjects/
    ref/
        Medulla/
            Medulla.exe
            NetRemote.exe
            CartActivator.dll
        Clumsy/
            ClumsyConsole.exe
            ClumsyCore.dll
        Simple/
            SimpleComposer.exe
            SimpleCore.dll
        MDCSToolBox.dll
    ProjName/                                   // 某项目代号作为文件夹名
        ProjName.sln                            // 解决方案文件
        CartAdapters/
            01/                                 // 车型 01 的车载插件
                MedullaAdapter/                 // Medulla 插件目录
                    MedullaAdapter.csproj       // C#项目文件
                    ProjNameCart.cs             // Medulla 车体
                    ProjNameRemote.cs           // UI 遥控器
                    MotorRoutine.cs             // 驱动报文处理线程
                    IORoutine.cs                // IO 报文处理线程
                ClumsyPilot/                    // Clumsy 插件目录
                    ProjNamecDef.cs             // Clumsy 车体定义
                    ProjNameConfigs.cs          // Clumsy 参数表
                    Movements.cs                // 动作定义
                    AGV.cs                      // 调度接口函数
            ......                              // 该项目中的其余车型（若有）
        Scenes/
            AMRScene1/
                ProjNameCar.cs                  // 业务场景中的车型
                WebApi.cs                       // 网络接口文件
```

```
        Mission.cs                          // 任务进程文件
    ......                                  // 该项目中的其余业务场景（若有）
```

需要注意以下开发规范或者常见问题：

- 解决方案以车型号或项目代号命名，各文件和类以车型号或项目代号为前缀命名。即，将上述目录结构中的 ProjName 更换为具体的车型号或项目代号。
- AMRScene 中的 Car.cs 与 CartAdapters 中的一组 M、C 插件相对应。若一个项目中有多款车混合调度，则在 CartAdapters 中添加 02、03 等，各文件也添加后缀“_02”。例如“FGXXXX001_02Cart.cs”。
- MedullaAdpater 需引用 Medulla.exe、NetRemote.exe、CartActivator.dll。
- ClumsyPilot 需引用 ClumsyConsole.exe、ClumsyCore.dll。
- AMRScene 需引用 SimpleComposer.exe、SimpleCore.dll。
- MedullaAdapter 中一般情况下包含 MotorRoutine 和 IORoutine，某些车型可能还包括 BatteryRoutine 等，应按照实际情况增删。
- AMRScene 中的 Mission 应按照实际情况增删。
- 在从别处拷贝项目后，可能需要将引用删除再重新添加；若某个项目中所有引用都有感叹号，请删除.csproj 中最后一个<Target></Target>标签中的内容。

2 Medulla 传感器插件

【待完善】

3 Medulla 车体插件 MedullaAdapter

MedullaAdapter 生成的 ProjName.dll 插件，由 Medulla 进行加载（见《MDCS 使用指南》2.2.1），使得 Medulla 能够读取 CAN、PLC 等所有车体相关的报文，从而将车体硬件转化为一组能够被上层软件结构使用的“抽象数据”（例如，一个舵轮会被抽象为“角度数值”+“速度数值”，同时可能还有“角度反馈数值”和“速度反馈数值”）。

在本教程中，把 MedullaAdapter 生成的 ProjName.dll 插件简称为 m.dll。

3.1 ProjNameCart

ProjNameCart.cs 文件中包含类 ProjNameCart ： CartDefinition。显然，这就是我们的 Medulla 车体定义开始的地方。

首先，我们在 ProjNameCart 类中需要定义若干的成员变量来表示车体信号。这些成员分为以下 4 类：

- AsUpperIO：上位信号。由 Medulla 上层的软件下发同名同 Attribute（直接复制黏贴在 ClumsyPilot 的 Cart.cs 中）的变量至 Medulla，再由 Medulla 转化为实际的报文。例如下发的电机速度、下发的舵轮角度、下发的货叉举升速度，等等。
- AsLowerIO：下位信号。由 Medulla 读取报文，再由发送至上层同名同 Attribute（直接复制黏贴在 ClumsyPilot 的 Cart.cs 中）的信号。一般是反馈信号或者各类 IO 传感器。例如，电机速度的反馈值、避障传感器触发等信号。
- AsInitParam：表示初始化参数的变量。这些变量可以在 Medulla 加载 m.dll 之前，让用户方便的进行赋值，而无需重新更改代码中的数值再重新编译 dll 文件。使用方法见《MDCS 使用指南》2.2.1。
- IOObjectMonitor：相当于普通的变量。与普通的成员变量唯一不同的点在于，被设定为 IOObjectMonitor 的变量会在 UI 上显示。需要注意，上面 3 种成员变量也会自动显示在 Medulla 的 UI 上（图 3.1-1 的区域 2）。

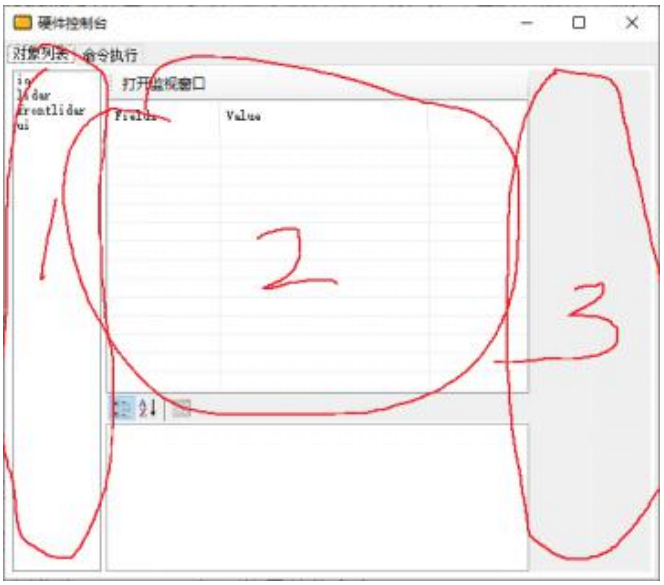


图 3.1−1

上述4种成员变量的声明方式非常简单，只需要在普通的成员变量前添加“特性”（即 Attribute，成员变量前面的中括号里的内容就是 Attribute。为了避免“特性”二字在中文语境下与别的含义冲突，后文全部用 Attribute 而不用“特性”来形容 C#的这个语言特性。可以看到这里已经造成迷惑的表述了）。下方为几个示例。

```
[AsUpperIO(desc = "下发速度", timeOutReset= true)] public float velocity;
[AsLowerIO(desc = "实际速度", timeOutReset= true)] public float actualV;
[AsInitParam(desc = "最大速度")] public float MaxSpeed;
[IOObjectMonitor(desc = "速度原始报文")] public float rawVelocity;
```

全部的 4 种 Attribute 均有 desc 属性，desc 内容的字符串会显示在 Medulla 图 3.1−1 的区域 2 中。另外，AsUpperIO 与 AsLowerIO 还具有 timeOutReset 属性，当设为 true 时，若相关的变量没有被更新，则会被置零。

ProjNameCart.cs 中还需要关注 Init()函数。Init()函数在 c.dll 插件被加载后首先运行。因此，对各种通信协议的初始化应放在该函数中。

各类常用通讯协议的规范与函数接口请参考开发指南附录 A。

3.2 Routine

各种信号与报文在各种 Routine ：LadderLogic<ProjNameCart>类中进行处理，包括读取（下位信号）或者下发（上位信号）。常见的有 MotorRoutine、IORoutine、BatteryRoutine 等。每个 Routine 可被视为一个梯形图，或是一个线程。线程不断以某个固定周期运行，每个周期调用 Operation()函数，对各种报文和信号进行赋值和读取操作（相当于一个 while true）。Operation()可视作 while 内执行的内容。启用 Routine 的方法是，在类 ProjNameCart 前添加 Attribute：

```
[UseLadderLogic(logic = typeof(MotorRoutine), scanInterval = 10)]
```

scanInterval 即线程运行的周期，单位毫秒。UseLadderLogic Attribute 可添加多个。

原则上来讲，Medulla 层面对上位信号只读不写（读取后将值赋给报文），对于下位信号可写可读(将读取的报文信息赋值给下位信号变量,或者读下位信号的值用来给别的变量赋值)。需要注意，一定要避免在 Routine 内对上位信号进行赋值，同时原则上应避免使用报文以外的数值对下位信号进行赋值。

在多数情况下，信号变量与报文一一对应，比如，3.1 中的 velocity 对应速度报文。但有时，报文与信号变量存在多对一、一对多、或多对多关系。例如，车上的转向灯、柱灯状态可能同时与车辆的前进速度和转向角度有关（通常，各种灯应作为 IOObjectMonitor，因为这些信号往往在 Medulla 层面根据别的变量就可以得到。但也要灵活变通！如果某个灯与自动驾驶动作有关，它也可能是 AsUpperIO）。

各类常用通讯协议的规范与函数接口请参考开发指南附录 A。

3.2 ProjNameRemote

类 ProjNameRemote ： ManualController<ProjNameCart>是实现 Medulla 遥控器的接口。遥控器的核心作用是验证报文对接是否正确。

尽管这个类放在 Medulla 插件中，仍应把它想象成 Medulla 的“上层”。这里提到的“上层”与本章第一段种提到的“上层”是同一个概念，我们的自动驾驶软件 Clumsy、简易遥控器、遥控器软件 NetRemote 都属于这一范畴。在 Medulla 的视角里，遥控器或者 Clumsy 其实是等价的（它们都是给 Medulla 下发上位信号、读取 Medulla 的下位信号的控制源）。

首先在 ProjNameRemote 类中定义若干遥控器控件变量。支持的控件类型如下：

类名	功能	包含成员	返回值范围
Throttle	滑钮	float val	0 到 1
DThrottle	双向滑钮	float dval	−1 到 1
Switch	开关	bool on	true/false
Button	按钮	bool pressed	true/false
Stick	摇杆	float x，y	−1 到 1
Knob	旋钮	byte val，int lastDir	0 到 255，−1/0/1
Dpad	四向键	float x，y	−1 到 1

在 ProjNameRemote 中定义控件变量前，也要添加 AsControlItem 这一 Attribute。如下：

```
[AsControlItem(name = "速度控制")] public Throttle velocity;
[AsControlItem(name = "方向控制")] public Dpad direction;
```

ProjNameRemote 的主体部分是 Operation()函数。与 Routine 的 Operation()类似，遥控器的 Operation()也是一个线程，其中执行上位变量的赋值，从而实现车体控制。启用遥控器的方法是，在类 ProjNameCart 前添加 Attribute：

```
[UseManualController(maunalController = typeof(ProjNameRemote))]
```

3.3 开发规范

4 Clumsy 自动驾驶插件 ClumsyPilot

5 Simple 业务调度插件 AMRScene

5.1 场景对象介绍

Simple 场景中主要是以下 3 个主要操作对象，小车，站点，路径，在 xxCar.cs 里面定义了小车的基础属性以及动作，可以初始化小车参数，增加动作等等

```
[CarType("XX叉车", editor = typeof(XQE_Car))]
10 个引用|zw-huang, 8 天前|1 名作者，3 项更改
public class XQE_Car : ClumsyCar
{
    public static Logger logger;
    0 个引用|zw-huang, 29 天前|1 名作者，2 项更改
    static XQE_Car()...

    0 个引用|zw-huang, 36 天前|1 名作者，1 项更改
    public static async Task<XQE_Car> Create() // boilerplate
    {
        var fl = new XQE_Car()
        {
            lstatus = "连接中",
            address = "127.0.0.1",
            name = $"XX叉车",
            haveCoordination = true,
            speed = 500,
        };

        return fl;
    }
```

5.1.1 小车对象

可以通 SimpleLib.GetAllCars()获取当前场景所有小车，然后使用 linq 查询满足我们条件的小车，也可以 SimpleLib.GetCar(int cartId)获取某个 ID 的小车，获取到小车对象后，可以获取这个车的名称，字段，statu 状态，status.enums 里面可以获取 clusmy 标记 upload 的变量以及小车的 tag 和交管字段，其他对象属性可以直接转到定义查看即可

例如：

```
//获取所有在线小车
var carsOnline = SimpleLib.GetAllCars().ToList().FindAll(c => c.siteID != -1);
//获取某个小车的电量
var soc= car.status.enums.ContainsKey("soc") ? float.Parse(car.status.enums["soc"]) : 0;
```

Common.cs 方法里面封装了一些常用的方法，比如 SelectCar,选择可用车，就判断了小车 siteID 以及 tags 是否满足条件

```
public static bool SelectCar(AbstractCar car)
{
    return !car.tags.Contains("occupied") &&
           car.siteID != -1 &&
           !car.tags.Contains("priority");
}
```

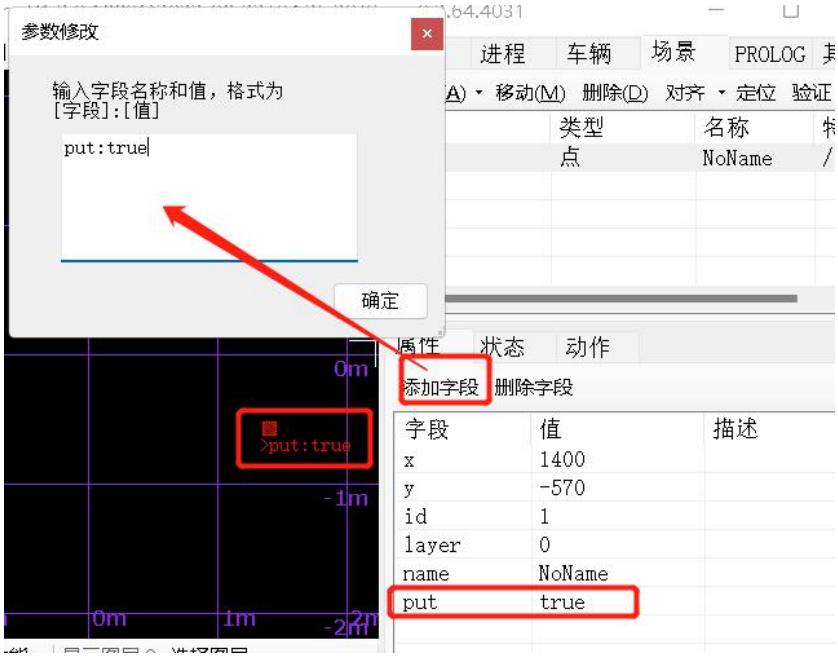
5.1.2 站点对象

1，可通过 SimpleLib.GetAllSites()获取当前场景所有站点集合，也可以

2，SimpleLib.GetSite(site.id))来获取单个站点对象，站点主要使用的是 field 字段属性，我们可以在某个站点标记一个字段，比如 put:true 那么程序中可以直接

3，var putSite = SimpleLib.GetAllSites().First(site => site.name == "put");

4，就可以获取这个站点对象，站点的其他对象属性可以直接转到定义查看即可

5， 

5.1.3 路径对象

可通过 SimpleLib.GetAllTracks()获取当前场景所有线段，每个线段都有 2 个端点位 SiteA 和 SiteB,线段同理 Site 也有 Field 属性，可以通过标记线段的 field 程序就可以 linq 获取到其对象进行操作

5.2 创建点对点移动任务

Simple 中的路线编译器实现小车移动以及功能的脚本下发，先创建一个计划 SegmentPlan，然后对这个计划进行寻路 FindRoute，会返回一个路线脚本 code（包含 A—B 路线所有站点以及路线功能），然后 SendScript(code)即可下发给 Clumsy，Clumsy 解析然后进行移动或功能的实现。

了解以上我们开始编写一个简单的 move 动作，这一块我们在 Common.cs 里面编写，这个方法和选中小车右击去某个点类似，具体流程如下方法：

```
public static Task<bool> move(AbstractCar car, int dstId)
{
    //创建一个小车的移动任务
    var mplan = new SegmentPlan() { usingCar = car, };
    //根据小车当前位置以及目标点进行寻路
    var weight = mplan.FindRoute(
        SimpleLib.GetSite(car.siteID),
        SimpleLib.GetSite(dstId));
    //给小车标记已占用
    car.tags.Add("occupied", $"go{dstId}");
    return Task.Run(() =>
    {
        try
        {
            var code = "";
            code += $"{mplan.Code()};" +
                    $"agv.Wait();"
                ;
            //下发路径给对应小车clumsy
            car.SendScript(code).Wait();
            //任务完成，siteId更新
            car.siteID = dstId;
            //移除已占用标记
            car.tags.Remove("occupied");
            return true;
        }
        catch (Exception ex)
        {
            car.tags.Remove("occupied");
            return false;
        }
    });
}
```

根据这个流程就，传入需要移动的小车以及目标点即可实现小车的单点移动任务

5.3 创建动作按钮

如果需要把所有线段都增加一个的 switchBarrier(切换避障区域)标记，对于小场景可以手动添加，对于大场景，我们可以动态的进行添加。
对于小车动作，我们带上 MethodMember 的标记，会显示在界面-车辆-选择小车-动作里面

```
[MethodMember(name = "增加 switchBarrier 标记", desc = "双击增加 switchBarrier 标记")]
public void AddSwitchBarrier()
{
```

```
    //获取场景中所有的路线进行遍历
    foreach (var tracks in SimpleLib.GetAllTracks())
     {
        //判断路线是否有 swichBarrier 字段，如果有不添加
        if (!tracks.fields.ContainsKey("switchBarrier"))
        {
            //给所有的线段加上 switchBarrier:1 的字段，当前也可以根据业务逻辑等灵活使用
           tracks.fields.Add("switchBarrier","1");
        }
    }
}
```

写好后生成 dll，到 Simple 目录下，打开 simpeComposer，添加小车车，选中小车就可以看到增加增加 switchBarrier 标记的方法，双击就可以把所有线段增加避障切换标记

这个方法的好处是我们可以根据现场业务逻辑灵活使用，如果现场都是重复性工位，每个工位都需要增加一个 switchBarrier 字段，那么我们就可以动态的给全场景增加和删除字段了，来代替手动一个个添加

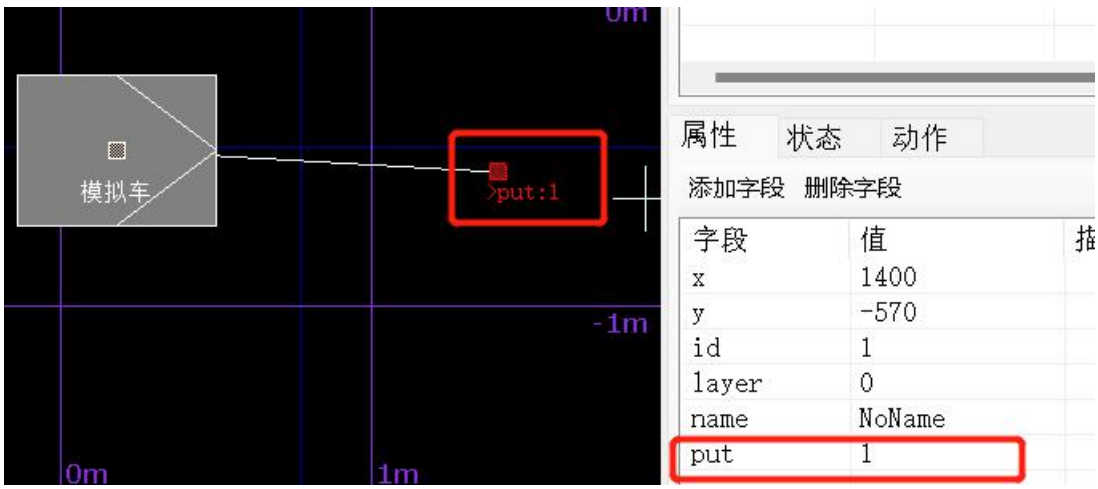
5.4 TempleteCoder 使用

编写 coder 可以根据业务场景灵活的调用 AGV.cs 方法，我们先在 clumsy 插件代码中 AGV.cs 里面写好接口方法，然后 Simple 这边根据一定规则进行调用，具体如下：

5.4.1 siteCoder

示范一个让小车到达站点执行 Clumsy AGV.cs 里面的取料方法案例

首先我们站点标记字段 put：1



然后程序中，需要在 SiteFields 类里面增加 put 属性

```
Class SiteFields
{
        public int put= -1;
```

```
}
```

在 useVerb 里面判断"dst.put!=-1" 如果条件满足，就执行 templateString 内容，调用 AGV.cs 方法里面的 Put 方法。

```
[TemplateSiteCoderSettings(
        priority = 3,
        useVerb = "dst.put!=-1",
        templateString = "agv.Put(${dst.put});",
        blockVerb = "true",
        siteFields = typeof(SiteFields),
        trackFields = typeof(TrackFields),
        planFields = typeof(PlanField))]
```

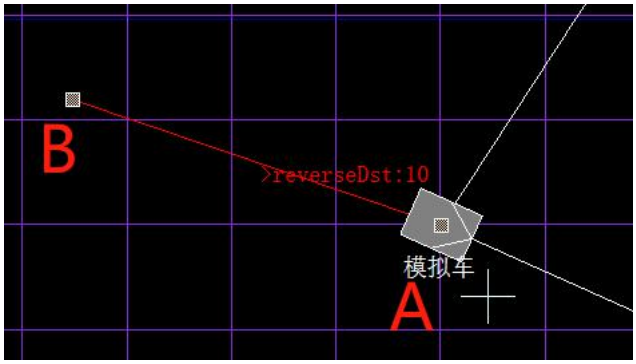
如果站点同时标记了 2 个方法，会根据调用 priority 值更大的那个方法，如果想调用 2 个可以在 templateString ="agv.Put(${dst.put});agv.Wait();agv.xx();" 后添加 agv.xx 方法即可

当小车到达标记 put 字段的站点后调度会自动解析 TemplateSiteCoderSettings 从而调用 templateString 的方法。

5.4.2 trackCoder

示范一个可以让叉车 A—B 倒走，B—A 正走的案例

首先我们线段标记 reverseDst：10 代表需要倒走的目标点



然后程序中，需要在 TrackFields 类里面增加 reverseDst 属性

```
class TrackFields
{
    public int speed = -1;
    public int reverseDst = -1;
}
```

在 useVerb 里面判断"track.reverseDst == dst.id" 如果条件满足，就执行 templateString 内容，调用 AGV.cs 方法里面的 ReverseGo 方法。

```
[TemplateTrackCoderSettings(
        priority = 5,
        useVerb = "track.reverseDst == dst.id",
        blockVerb = "true",
        templateString =
"agv.ReverseGo(${src.x},${src.y},${src.id},${dst.x},${dst.y},${dst.id},${track.id}," +
                        "${track.speed});",
        siteFields = typeof(SiteFields),
        trackFields = typeof(TrackFields))]
```

这样当需要小车从 A-B 走的时候就会自动将 ReverseGo 方法打包到 code 里面了

5.4.3 planCoder

planCoder 用法也类似和 SiteCoder 和 TrackCoder 但是 Plancoder 不需要在场景上作标记，需要在程序里面作标记

我们示范一个下取料的 plan

首先我们先声明 paln 的字段 action

```
1，class PlanFields
2，{
3，        public string action = "/";
4，}
```

5，在 useVerb 里面判断"plan.action=='put'" 如果条件满足，就执行 templateString 内容，调用 AGV.cs 方法里面的 Put 方法。

```
6，[TemplateTrackCoderSettings(
7，            priority = 10,
8，            useVerb = "(plan.action=='put')",
9，            templateString =
"agv.Put(${src.x},${src.y},${src.id},${dst.x},${dst.y},${dst.id},${track.id},${track.speed});",
10，            blockVerb = "true",
11，            siteFields = typeof(SiteFields),
12，            trackFields = typeof(TrackFields),
13，            planFields = typeof(PlanFields))]
14，
```

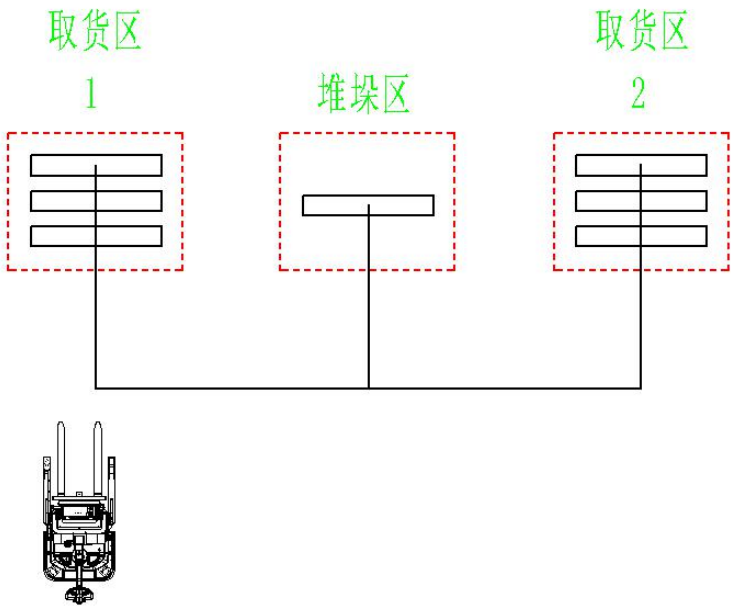
15，程序中创建一个移动方法,当执行这个移动任务的时候如果 plan.field 里的 action 不为空，则调用对应 templateString 的方法

```
16，   //创建一个移动任务
17，    var plan = new SegmentPlan() { usingCar = this };
18，   //给任务计划添加一个"put"的 action
19，  plan.fields["action"] = "put";
20，   //寻路
21，   plan.FindRoute(startSite, endSite);
22，   var code = $"{plan.Code()};";
```

```
23,     Console.WriteLine(code);
24,     //下发任务
25,     var tsk = SendScript(code);
26,     await tsk;
27,     //任务完成
28,  siteID = fetch.id;
```

5.5 创建 Demo 路线任务

了解以上知识后，我们综合做一个 Demo 路线任务，业务需求是这样的



目前计划使用 3 个木箱，放在取货区 1
Demo 流程为：
①车辆从取货区 1 依次取三个木箱，在堆垛区完成三次堆垛
②车辆在堆垛区对木箱进行拆垛，依次运送到取货区 2

准备工作：
任务拆分：
1，首先我们写一个堆垛任务 Task，用于实现①任务进行 3 次取料->放料堆垛

```
//创建一个堆垛任务
//1，先从起点(startSite)去取货区1（fetch）取物料
//2，取到料后从取料点把物料放到堆垛区(endSite)根据(action)提供的动作进行多层堆叠
3 个引用 | zw-huang，36 天前 | 1 名作者，1 项更改
public async Task FetchStack(Site startSite, Site fetch, Site endSite, string action)
{

    var plan = new SegmentPlan() { usingCar = this };
    //plan.fields["action"] = "fetchDwon";
    plan.FindRoute(startSite, fetch);
    var code = $"{plan.Code()};";
    Console.WriteLine(code);
    var tsk = SendScript(code);
    await tsk;
    siteID = fetch.id;

    plan = new SegmentPlan() { usingCar = this };
    //plan.fields["action"] = action;
    plan.FindRoute(fetch, endSite);
    code = $"{plan.Code()};";
    Console.WriteLine(code);
    tsk = SendScript(code);
    await tsk;
    siteID = endSite.id;
}
```

29，再写一个用于拆垛的任务 Task，用于实现②任务进行 3 次拆垛->放料

```
//创建一个拆垛任务
//1，先从起点去堆垛区1 按照层数取物料
//2，取到料后从取料点把物料放到取货区2
6 个引用 | zw-huang，36 天前 | 1 名作者，1 项更改
public async Task FetchPut(Site startSite, Site fetch, Site endSite, string action)
{
    SegmentPlan plan;
    string code;
    Task tsk;

    plan = new SegmentPlan() { usingCar = this };
    plan.fields["action"] = action;
    plan.FindRoute(startSite, fetch);
    code = $"{plan.Code()};";
    Console.WriteLine(code);
    tsk = SendScript(code);
    await tsk;
    siteID = fetch.id;

    plan = new SegmentPlan() { usingCar = this };
    plan.fields["action"] = "put";
    plan.FindRoute(fetch, endSite);
    code = $"{plan.Code()};";
    Console.WriteLine(code);
    tsk = SendScript(code);
    await tsk;
    siteID = endSite.id;
}
```

3，编写一个可以根据站点名称获取站点对象的方法

```
//根据站点name名称来获取站点对象
19 个引用|zw-huang, 36 天前|1 名作者, 1 项更改
public Site GetSiteByName(string name)
{
    return SimpleLib.GetAllSites().First(site => site.name == name);
}
```

以上都准备好后，开始进行 Demo 编写

1，获起点/堆垛的站点对象

```
//1获取起点/堆垛区的站点对象
var startSite = SimpleLib.GetAllSites().First(site => site.name == "startSite");
var stackSite1 = SimpleLib.GetAllSites().First(site => site.name == "stack1");
```

2，判断小车是否在起点，如果不在先调度到起点位置

```
//2判断小车是否在起点，如果不在先调度到起点位置
if (car.siteID != startSite.id)
{
    Console.WriteLine("** Going to start site");
    await Commons.move(car, startSite.id);
}
```

3,开始①进行堆垛任务

```
//3先从起点(startSite)去取货区1（fetch1）1站点取物料，取完后放到堆垛区 stackSite1
await FetchStack(GetSiteByName("startSite"), GetSiteByName($"fetch1"), stackSite1, "put");
//4去取货区1（fetch2）2站点取物料，取完后放到堆垛区 stackSite1 进行二层堆垛
await FetchStack(stackSite1, GetSiteByName($"fetch2"), stackSite1, "stack");
//5去取货区1（fetch3）3站点取物料，取完后放到堆垛区 stackSite1 进行三层堆垛
await FetchStack(stackSite1, GetSiteByName($"fetch3"), stackSite1, "stack3");
```

4，开始②进行拆垛任务

```
//6堆垛完成回到起点
Commons.move(this, GetSiteByName("startSite").id).Wait();
// 7先从起点(startSite)去堆垛区stackSite1取第三层物料，取完后放到放料点 put1
await FetchPut(GetSiteByName("startSite"), stackSite1, GetSiteByName($"put1"), "fetch3");
// 8放料点(put1)去堆垛区stackSite1取第二层物料，取完后放到放料点 put2
await FetchPut(GetSiteByName($"put1"), stackSite1, GetSiteByName($"put2"), "fetch");
// 9放料点(put2)去堆垛区stackSite1取第一层物料，取完后放到放料点 put3
await FetchPut(GetSiteByName($"put2"), stackSite1, GetSiteByName($"put3"), "fetchDown");
```

5，整个 Demo 如下，即可完成上述的需求

```
[MethodMember(name = "Demo")]
0 个引用|0 项更改|0 名作者，0 项更改
public async void Demo()
{
    Console.WriteLine("** Start Demo");

    //1获取起点/堆垛区的站点对象
    var startSite = SimpleLib.GetAllSites().First(site => site.name == "startSite");
    var stackSite1 = SimpleLib.GetAllSites().First(site => site.name == "stack1");

    //2判断小车是否在起点，如果不在先调度到起点位置
    if (siteID != startSite.id)
    {
        Console.WriteLine("** Going to start site");
        var plan = new SegmentPlan() { usingCar = this };
        plan.FindRoute(SimpleLib.GetSite(siteID), startSite);
        var code = $"{plan.Code()}; agv.Wait();";
        Console.WriteLine($"**code: {code}");
        var tsk = SendScript(code);
        await tsk;
        siteID = startSite.id;
    }
        //3先从起点(startSite)去取货区1（fetch1）1站点取物料，取完后放到堆垛区 stackSite1
        await FetchStack(GetSiteByName("startSite"), GetSiteByName($"fetch1"), stackSite1, "put");
        //4去取货区1（fetch2）2站点取物料，取完后放到堆垛区 stackSite1 进行二层堆垛
        await FetchStack(stackSite1, GetSiteByName($"fetch2"), stackSite1, "stack");
        //5去取货区1（fetch3）3站点取物料，取完后放到堆垛区 stackSite1 进行三层堆垛
        await FetchStack(stackSite1, GetSiteByName($"fetch3"), stackSite1, "stack3");
        //6堆垛完成回到起点
        Commons.move(this, GetSiteByName("startSite").id).Wait();
        // 7先从起点(startSite)去堆垛区stackSite1取第三层物料，取完后放到放料点 put1
        await FetchPut(GetSiteByName("startSite"), stackSite1, GetSiteByName($"put1"), "fetch3");
        // 8放料点(put1)去堆垛区stackSite1取第二层物料，取完后放到放料点 put2
        await FetchPut(GetSiteByName($"put1"), stackSite1, GetSiteByName($"put2"), "fetch");
        // 9放料点(put2)去堆垛区stackSite1取第一层物料，取完后放到放料点 put3
        await FetchPut(GetSiteByName($"put2"), stackSite1, GetSiteByName($"put3"), "fetchDown");

        Console.WriteLine("task finished");
}
```

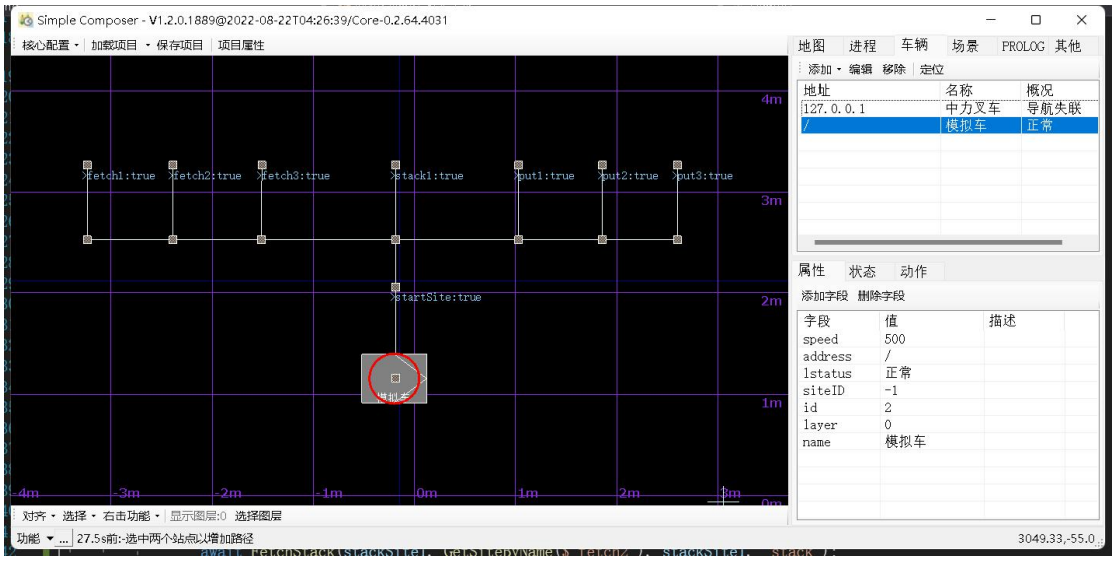
6，下一步就是进行实车测试，，我们需要绘制好路径和站点，并且站点名称标记好即可进行测试

5.6 模拟车测试

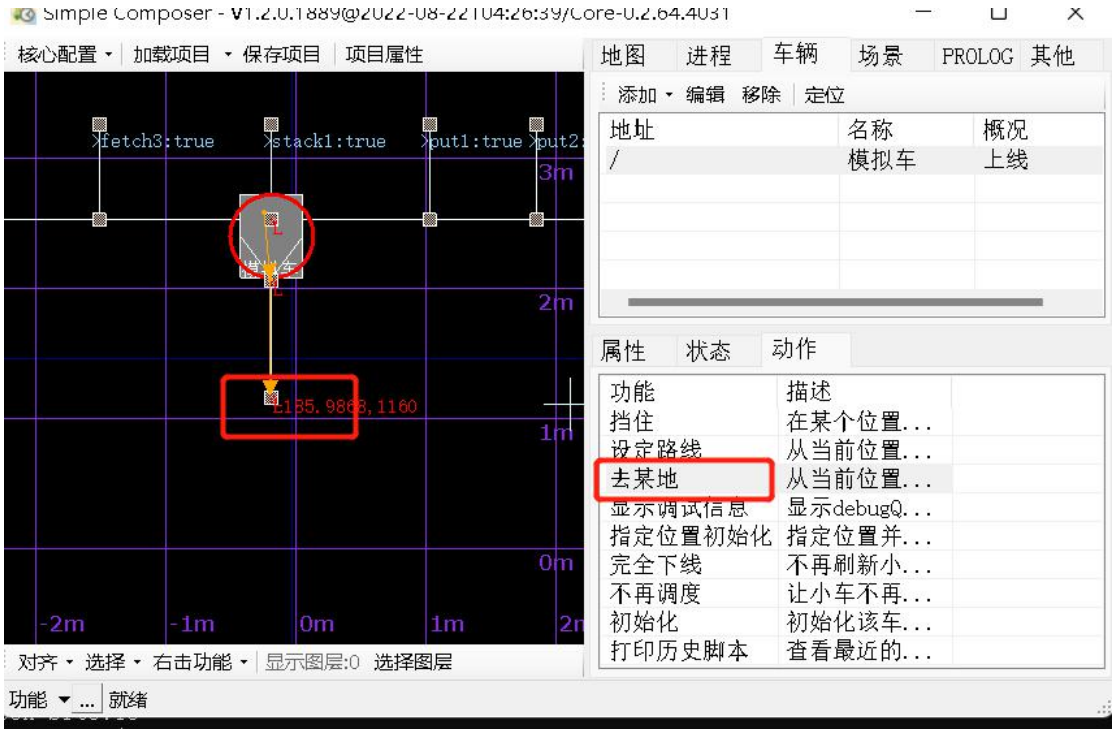
模拟车主要用于业务模拟，交管模拟，项目最好都经过模拟车验证逻辑后再跑物理车，因为物理车跑起来效率低，并且短时间大量模拟测试。

1，将业务插件编译，放到 SimpleComposer/plugins 目录内，然后打开 SimpleComposer，创建模拟车，站点，路径，（模拟车只能进行路径模拟）

2，配置模拟车（速度 speed 一定要配置）/站点/路径。



3，选择模拟车，动作，双击“前往站点”，在鼠标左键要去的站点，模拟车就会移动过去了



4，这是手动发送，我们也可以通过其他接口发送任务给小车，比如 webApi 或者其他接口

5.7 创建进程任务

如果想要某个业务逻辑一直进行工作，可以创建一个 mission 进程
例如，需要对和充电站做通讯，可以创建一个 RechargeMission，主要就是开个线程，然后在线程里面处理通讯连接以及充电逻辑

```
[MissionType("充电站通讯", editor = typeof(RechargeMission))]
2 个引用|0 项更改|0 名作者，0 项更改
public class RechargeMission : Mission
{
    0 个引用|0 项更改|0 名作者，0 项更改
    public static Mission Create()
    {
        return new RechargeMission();
    }
    public bool started = false;
    public byte[] sendByte = new byte[32];
    public byte startCharge = 0;
    public  float chargeVoltage = 29.20F;
    public  float chargeElectricCurrent = 30.00F;
    public ushort chargeTimeSpan;
    public  float carVoltage;
    public ushort carSoc;
    public  float carElectricCurrent;
    [MethodMember(name = "启动进程", desc = "启动进程")]
    0 个引用|0 项更改|0 名作者，0 项更改
    public override void Execute()
    {
        if (started) return;
        started = true;
        status.status = "已启动";
        new Thread(() =>
        {
            while (isExist())...

        }).Start();


    }
```

1，创建好进程后，更新插件，可以在界面进程-新建自定义进程 里面看到已添加的进程

2，给进程任务添加按钮，创建一个方法，给这个方法加上 MethodMember 的特性即可，name 里面填写这个按钮的名称，desc 里面填写描述即可

```
}
[MethodMember(name = "打印任务", desc = "打印任务")]
0 个引用|zw-huang，16 天前|1 名作者，1 项更改
public void PrintTask()
{
    foreach (var item in taskList)
    {
        Console.WriteLine($"src:{item.src}|dst:{item.dst} statu:{item.taskStatu}\n");
    }
}
[MethodMember(name = "清除任务", desc = "清除任务")]
0 个引用|zw-huang，16 天前|1 名作者，1 项更改
public void ClearTask()
{
    taskList.Clear();
}
```

3，如何打开软件自动启动进程

在 XXCar.cs 里面，创建一个线程，里面用于开启自动项目加载，具体方法如下

```
new Thread(() =>
{
    ChainedDeliveryMission DeliveryMissionn = null;
    RechargeMission rechargeMission = null;
    while (true)
    {
        Thread.Sleep(1000);
        foreach (var tasks in SimpleProject.proj.Missions)//从当前项目中遍历Missions
        {
            Console.WriteLine(tasks.name);
            if (DeliveryMissionn == null && tasks.name == "chainedDelivery")//判断进程名称
            {
                DeliveryMissionn = (ChainedDeliveryMission)tasks;
                DeliveryMissionn.Execute();//启动进程
            }
            if (rechargeMission == null && tasks.name == "rechargeMission")
            {
                rechargeMission = (RechargeMission)tasks;
                rechargeMission.Execute();
            }
        }

        if (DeliveryMissionn != null && rechargeMission != null) break;//进程都打开，结束

    }

}).Start();
```

开发指南附录

附录 A 常见通讯协议及以及接口

- CAN 总线通讯：CAN/CANOPEN
- 串口通讯（485/232/422）：modbus (RTU/ASCII)/自由口协议
- 以太网通讯：modbusTcp/tcpClient/tcpSever/udpClient/udpSever

A.1 CAN 总线通讯

主要用于电机/编码器/电池等硬件通讯，叉车通常用 can 通讯比如柯蒂斯驱动器的通讯属于自由定义的通讯协议，常规小车比如使用伺服驱动器（步科）使用的则是 CanOpen 协议，是一个标准协议，要根据协议规则来进行配置和使用,CanOpen 包含 4 个 TPDO 以及 4 个 RPDO，以及 SDO，伺服驱动器常用 CANOPEN 协议，通过 TPDO 以及 RPDO 携带的信息进行交互控制，一般 TPDO1：帧（181+id）为是接收数据 ID 地址， RPDO1:帧（201+id）是写入数据的地址以此类推。
我们常用的是 CAN 协议中的数据帧，数据帧一般由 7 个段构成，即：
（1） 帧起始。表示数据帧开始的段。
（2） 仲裁段。表示该帧优先级的段。
（3） 控制段。表示数据的字节数及保留位的段。
（4） 数据段。数据的内容，一帧可发送 0~8 个字节的数据。
（5） CRC 段。检查帧的传输错误的段。
（6） ACK 段。表示确认正常接收的段。
（7） 帧结束。表示数据帧结束的段

其中仲裁段也就是 CANID，越小优先级越高，可以这么理解我们 CAN/CANOPEN 通讯就是去获取对应仲裁段（CANID）的数据段。
还有一个是拓展帧以及远程帧某些项目会用到，基本使用到就这 3 个帧
更详细的 CAN 通讯请自行查阅，此处只做简单介绍

以下为封装好的 CANHelper 使用方法（CAN/CANOPEN 都可用）

步骤如下：
1，初始化 CAN/配置波特率等参数，确认有几路 CAN（一般最大 2 路 CAN）
2，确认设备报文是否主动上传，如果不是需要发送广播指令 帧 00 数据: 01 00 来启动下 CAN 总线下所有节点
3，确认发送和读取的报文 ID 以及数据偏移进行下发或者解析

主要分 2 类 TCP 转 CAN GCANTCP 以及 CAN ZLGCAN

ZLGCAN 在使用之前我们需要根据对应协议修改对应的通讯波特率，如下规则：

```
//10 Kbps 0x31 0x1C
//20 Kbps 0x18 0x1C
//40 Kbps 0x87 0xFF
//50 Kbps 0x09 0x1C
//80 Kbps 0x83 0xFF
//100 Kbps 0x04 0x1C
//125 Kbps 0x03 0x1C
//200 Kbps 0x81 0xFA
//250 Kbps 0x01 0x1C
//400 Kbps 0x80 0xFA
//500 Kbps 0x00 0x1C
//666 Kbps 0x80 0xB6
//800 Kbps 0x00 0x16
//1000 Kbps 0x00 0x14
//33.33 Kbps 0x09 0x6F
//66.66 Kbps 0x04 0x6F
//83.33 Kbps 0x03 0x6F
```

常用的是 00 1C 对应 500k 的波特率

```
VCI_INIT_CONFIG config = new VCI_INIT_CONFIG();
config.AccCode = 0;
config.AccMask = 0xFFFFFFFF;
config.Timing0 = 0x01;
config.Timing1 = 0x1C;
config.Filter = 0;
config.Mode = 0;
if (VCI_InitCAN(m_devtype, m_devind, 0, ref config) != STATUS_OK)
{
    Console.WriteLine("initialization can 1 failed");
    return;
};
```

使用：

GCANTCP：在 Init 里面初始化 tcpCan，传入通讯的 IP 以及 can1 路端口以及 can2 路端口即可

```
public GCANTCP can;

0 个引用 | zw-huang, 7 天前 | 1 名作者, 3 项更改
public override void Init()
{
    //电机
    can = new GCANTCP(motorIp,port1, port2);
    Console.WriteLine("init ok");
```

下发数据给设备:调用 sendMessage1(can1 路)sendMessage2(can2 路)此处 can1 用于和电机通讯,can2 用于和电池通讯,传入需要控制的 id 以及数据即可，一般数据都是 8 位，如没有请给 0，除非协议特意说明下发数据必须 2 位或其他否则一律 8 位

```
////PC发送到CURTIS（驱动数据） 0x226
cart.can.sendMessage1(0x226, new byte[8]
{
    (byte) ((int)cart.velocity & 0xff),
    (byte) (((int)cart.velocity >> 8) & 0xff),
    0,
    0,
    sendSolenoid,
    sendDownSpeed, 0, 0
});
```

读取设备数据：调用 canRecv.TryGetValue 方法传入需要读取的 id 以及输出的数据即可，canRecv 对应 can1 路 canRecv2 对应 can2 路

```
//CURTIS发送到PC（反馈数据） 0x1A6
if (cart.can.canRecv.TryGetValue(0x1A6, out var tup1A6))
{
    var br = tup1A6.Item1;
    cart.actualV = (short)(br[0] + (br[1] << 8));
    //cart.soc = br[4];
    cart.motorErrorCode = br[5];
    cart.steerErrorCode = br[6];
    cart.singleFeedback = br[7];
}
```

ZLGCAN：和 GCANTCP 几乎一样，就是初始化直接 new 就可以，GCANTCP 就是在 ZLGCAN 的基础上加了一个 TCP 转 CAN 模块

```
public ZLGCAN can;

0 个引用|zw-huang, 31 天前|1 名作者, 1 项更改
public override void Init()
{
    can = new ZLGCAN();
```

读和写与 GCANTCP 一样

###关于拓展帧使用方法：新增一个 sendMessage3 把 ExternFlag 改成 1 即可，非拓展帧必须为 0

```
//拓展帧
0 个引用|zw-huang, 48 天前|1 名作者, 1 项更改
public unsafe int sendMessage3(int id, byte[] data)
{
    VCI_CAN_OBJ sendobj = new VCI_CAN_OBJ();
    sendobj.RemoteFlag = 0;
    sendobj.ExternFlag = 1;
    sendobj.ID = (uint)id;
    sendobj.DataLen = 8;
    Marshal.Copy(data, 0, (IntPtr)sendobj.Data, 8);
    //Console.WriteLine($"Send to {id:X2}: {string.Join(" ", data.Select(p=>$"{p:X2}"))}");
    return (int)VCI_Transmit(m_devtype, m_devind, 1, ref sendobj, 1);
}
```

###关于远程帧使用方法：新增一个 sendMessage4 把 RemoteFlag 改成 1 即可，非远程帧必须为 0

```
0 个引用|zw-huang, 48 天前|1 名作者, 1 项更改
public unsafe int sendMessage4(int id, byte[] data)
{
    VCI_CAN_OBJ sendobj = new VCI_CAN_OBJ();
    sendobj.RemoteFlag = 1;
    sendobj.ExternFlag = 0;
    sendobj.ID = (uint)id;
    sendobj.DataLen = 8;
    Marshal.Copy(data, 0, (IntPtr)sendobj.Data, 8);
    //Console.WriteLine($"Send to {id:X2}: {string.Join(" ", data.Select(p=>$"{p:X2}"))}");
    return (int)VCI_Transmit(m_devtype, m_devind, 1, ref sendobj, 1);
}
```

A.2 串口通讯

串口通讯主要用于电机/PLC/声光/传感器等设备通讯，硬件接线采用 485/232/422 等接线方式通过设备协议来进行数据的收发，一般分为固定协议（RTU）或非固定协议（自由口），如某个灯设备支持 modbusRtu 协议，那么根据灯光厂家提供的控制协议，对不同地址的数据进行读写即可

简介下 modbusRtu 协议：

Modbus 的操作对象有四种：线圈、离散输入、保持寄存器、输入寄存器。

- 线圈:开关量，在 Modbus 中可读可写
- 离散量:开关量：在 Modbus 中只读
- 输入寄存器:只能从模拟量输入端改变的寄存器，在 Modbus 中只读
- 保持寄存器:输出模拟量信号的寄存器，在 Modbus 中可读可写

根据功能码我们读取不同的数据，如下表

代码	中文名称	英文名	位操作/字操作	操作数量
01	读线圈状态	READ COIL STATUS	位操作	单个或多个
02	读离散输入状态	READ INPUT STATUS	位操作	单个或多个
03	读保持寄存器	READ HOLDING REGISTER	字操作	单个或多个
04	读输入寄存器	READ INPUT REGISTER	字操作	单个或多个
05	写线圈状态	WRITE SINGLE COIL	位操作	单个
06	写单个保持寄存器	WRITE SINGLE REGISTER	字操作	单个
15	写多个线圈	WRITE MULTIPLE COIL	位操作	多个
16	写多个保持寄存器	WRITE MULTIPLE REGISTER	字操作	多个

目前可以使用已经封装好的 ModbusRtu 协议或者自由口的 SerialBase 类进行通讯

步骤如下：

1，确认设备的 COM 口，波特率，停止位，奇偶校验等参数

2，初始化串口

3，根据协议进行读写

ModbusRtu：调用 StartRtu()方法传入通讯参数

```
public ModbusSRtu RtuMagNav = new ModbusSRtu();

0 个引用 | zw-huang, 32 天前 | 1 名作者, 3 项更改
public override void Init()
{
    RtuMagNav.StartRtu("COM5", 115200, Parity.None, StopBits.One);
```

根据设备协议，确认功能码，调用对应的读写方法，如下控制灯颜色，灯设备站号是 9，写地址从 0 开始，功能码是 06 对应 writeSingleRegister()方法

06	写单个保持寄存器	WRITE SINGLE REGISTER	字操作	单个

```
public void lightWrite()
{
    cart.Light.writeSingleRegister(9,0,(ushort)cart.lightType);

}
```

读数据也如此，如：根据电池站号地址，数据地址，功能码，读取多少长度填入进去即可获取对应长度的数据，readRegisterBuffer 对应 03 功能码

03	读保持寄存器	READ HOLDING REGISTER	字操作	单个或多个

```
var data = cart.RtuBattery.readRegisterBuffer(1, 18, 3);
cart.electricCurrent = (((byte)data[2] << 8) | ((byte)data[3])) * 0.01f;
```

SerialBase:一般根据双方设定的协议进行交互（一发一回）

先调用 SerialPortInni()方法 传入参数初始化串口

```
serialBase.SerialPortInni(com, baudRate, parity, stopBits);
```

根据协议我们组合下发的数据，调用 ReadBase（sendData）返回的就是设备的数据了，正常串口通讯都是一发一回的。

```
/// <summary>
/// 读取串口的数据
/// </summary>
/// <param name="send">发送的原始字节数据</param>
/// <returns>带接收字节的结果对象</returns>
7 个引用 | zw-huang, 48 天前 | 1 名作者, 1 项更改
public byte[] ReadBase(byte[] send)
{
    hybirdLock.Enter();

    if (IsClearCacheBeforeRead) ClearSerialCache();

    var sendResult = SPSend(SP_ReadData, send);
    if (!sendResult)
    {
        hybirdLock.Leave();
    }

    var receiveResult = SPReceived(SP_ReadData, true);
    hybirdLock.Leave();

    return receiveResult;
}
```

A.3 以太网通讯

以太网的协议主要用 PLC/雷达传感器/第三方设备等通讯,常用的有 TCP/UDP 等协议，目前与 IO 通讯 (PLC)常用 ModbusTcp，用法与 modbusRtu 类似，可以参考上文串口通讯，TCP/UDP

主要用于和第三方设备通讯,对接其他系统或者设备
TCP 分为 TcpClient（客户端）以及 TcpListener（服务端）,引用系统 System.Net 即可使用,此处简单展示下如何进行客户端和服务端连接
客户端:

```
1 个引用|zw-huang，49 天前|1 名作者，1 项更改
private void  initTcpSocket(string IP, int Port)
{

    if (tcp==null)
    {
        tcp = new TcpClient(IP, Port);
        ns = tcp.GetStream();
        var sendbuffer = new byte[] {0x00,0x01,0x02 };
        ns.Write(sendbuffer, 0, sendbuffer.Length);//写数据
        byte[] readBuffer = new byte[11];
        ns.Read(readBuffer, 0, 11);//读数据
    }
```

服务端:

```
1 个引用|zw-huang，1 小时前|1 名作者，1 项更改
private void Acceptor(IAsyncResult o)
{
    TcpListener server = o.AsyncState as TcpListener;
    tcpClient = server.EndAcceptTcpClient(o);
    Console.WriteLine(tcpClient.Client.RemoteEndPoint.ToString());
    ns = tcpClient.GetStream();

}
1 个引用|zw-huang，1 小时前|1 名作者，1 项更改
private TcpListener InitMobus()
{
    try
    {
        tcpsever = new TcpListener(IPAddress.Parse(IOTcp), IOTcpPort);
        tcpsever.Start();//开启监听
        tcpsever.BeginAcceptTcpClient(new AsyncCallback(Acceptor), tcpsever);

    }
    catch (Exception ex)
    {
        Console.WriteLine($"初始化TcpListener失败，{ex.Message} ");
    }
    return (tcpsever);
}
```

UDP 通讯也分为 UdpClient（客户端）以及 UdpSever（服务端）,在 MDCS 系统中使用的很少简单用法如下:
服务端:

```
UdpSever.udpSever("127.0.0.1", 17800);
string Smessage = "udp data";
UdpSever.sendMsg("127.0.0.1", 17801, Smessage);
var revSUdp = UdpSever.ReciveMsg();
```

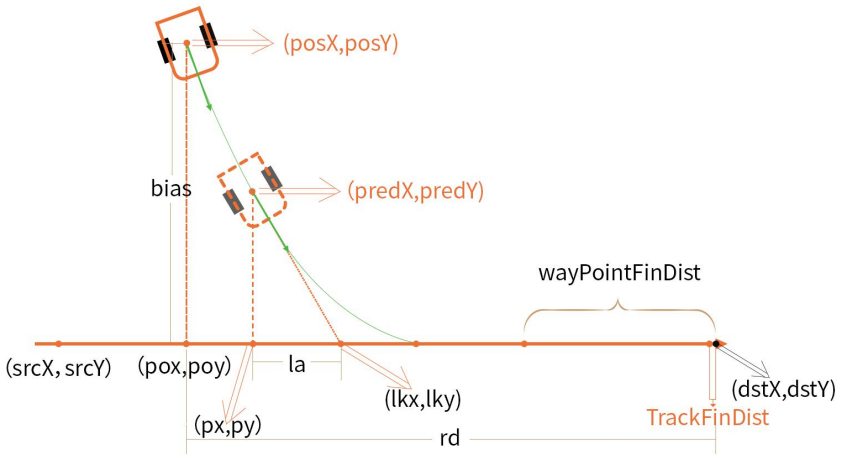
客户端:

```
UdpClient.udpClient("127.0.0.1", 17801);
string Cmessage = "udp data";
UdpClient.sendMsg("127.0.0.1", 17800, Cmessage);
var revCUdp = UdpClient.ReciveMsg();
```

附录 B Clumsy 内置运动模型及参数表

B.1 差速轮模型

差速轮



主方法 Track 运行逻辑：

1．当从 A 站点(srcX，srcY)开往 B 站点(dstX，dstY)时，利用贴近 AB 直线的算法让小车不断往 AB 直线靠近（具体影响贴近直线的运动参数将在下面"参数作用"部分详细说明）

(1) 以小车当前位置为起点，根据正向追踪点 la 参数计算出目标点（lkx，lky）

(2) 在迭代前进的过程中不断微调目标点的位置，并利用 pd 算法修正前进角度并下发速度前往目标点。

2．当小车行进至投影点(pox，poy)与目标点(dstX，dstY)之间的距离 rd 小于 wayPointFinDist 时，做以下判断：是否是最后一段需要行走的路程，即 AB 点后没有需要前往的 C 点。

(1) 如果是最后一段，则继续以较小速度前进，并在 rd 小于 trackFinDist 时停车。

(2) 如果后续还有需要前往的 C 点，且此路径设置为“需要直角转弯”时，根据 ABC 之间转向角度是否足够大判断是否需要精确到达该点后再前往下一站点。

① 如果不需要大角度转向（如 ABC 几乎在一条直线上），则直接开始从 B 点至 C 点的行走，并重复此运行逻辑。

② 如果需要大角度转向（如 ABC 三点之间夹角为 90 度），则继续以较小速度前进，并在 rd 小于 trackFinDist 时判断到达 B 点，然后再进行 B 点至 C 点的行走。

(3) 如果后续还有需要前往的 C 点，且路径设置为“圆弧过弯”，则和上一个条件下的“不需要大角度转向”类似，无需继续前往 B 点，直接从当前位置开始从 B 点至 C 点的行走（即使 ABC 之间的转弯角度较大）。

可调用方法：

1. void Init(double x, double y)：

初始化定义一个点用的 List<XY>pointsXY、List<Action>actionStart、List<Action>actionEnd、List<int> trackType。其中 pointsXY 是站点位置的列表，actionStart 是开始行走动作前需要执行的函数列表，actionEnd 是行走动作结束后需要执行的函数列表，trackType 为路径类型的列表。

2. bool AddPoint(double x, double y, Action start = null, Action end = null, int type = 0)

添加一个点。其中位置 x 和 y 为必选参数，行走前动作 start、行走后动作 end 为可选参数，默认为 null，路径类型 type 也是可选参数，默认为 0，即过弯时使用圆弧。1 为：直角转弯，需要精确到达该点后执行转弯动作。2 为：必须停车点。

3. IEnumerable<bool> TurnTo(float deg)

用于转至指定角度的函数，返回是否转向成功的迭代器。

4. IEnumerable<bool> Track(TrajectoryReplanningObstacleAvoidance avoidance=null)

具体说明见“主方法 Track 运行逻辑”，可选参数：TrajectoryReplanningObstacleAvoidance avoidance，用于实现避障。

参数说明：

参数名称	参数类型	参数说明	参数范围	具体起效步骤
baseSpeed	float	预设基础速度		1&2
isReverse	bool	是否是倒车	true/false	1&2
predict	float	控制预测小车位置变化量的比	0—1	1.1

		例，用于控制预测小车的位置，即 predX 和 predY 的大小		
thscale	double	控制预测小车所在的方向，用于控制预测小车的位置，即 predX 和 predY 的大小	0—1	1.1
lookAhead	float	车到路径垂足点向前的追踪量，可调节小车贴近路径的快慢，过小容易超调，过大则调整过慢，建议高速长直线适当增大，低速高精度适当调小	>0	1.1
lineSnappingDist	float	车体到路径的距离，控制 lookAhead 衰减	>0	1.1
stoppingLAFac	double	接近终点时 lookAhead 的衰减系数	0—1	1.1
maxSpeed	float	允许的最大速度		1.2
maxWSpeed	float	转弯最大速度		1.2
nffac	double	pid 控制中的 d，角度负反馈修正系数	0—1	1.2
dthfac	float	pid 中的 p，角度比例增益		1.2
thresAcc	double	最大加速度，调节小车加减速率	>0	1.2
thresWAcc	double	最大转弯加速度，调节小车加减速率	>0	1.2
wayPointFinSpeedFactor	float	该值控制“当速度越大时”连续行走半径的增大幅度	0—1	2
wayPointFinSpeedPow	float	该值控制“当速	>0	2

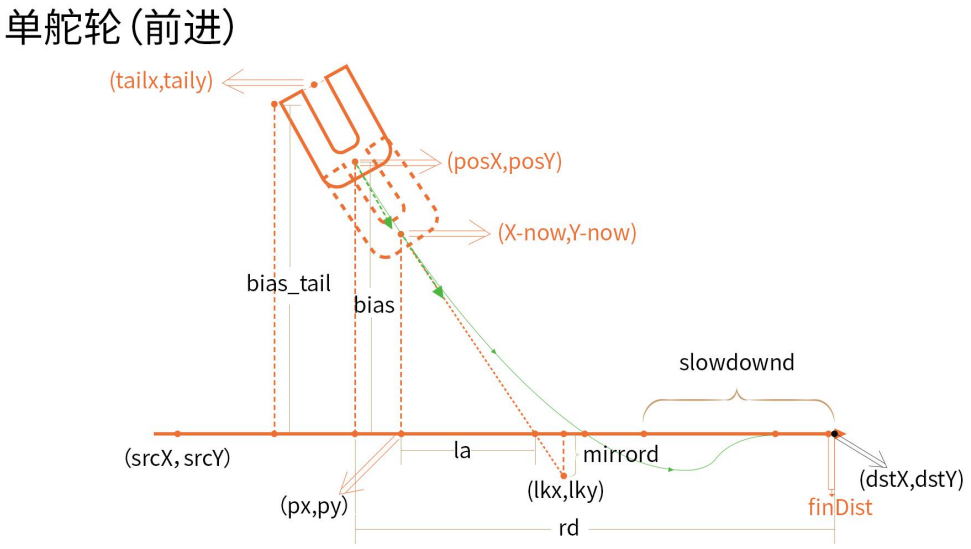
		度越大时”连续行走半径的增大幅度		
wayPointFinBase	float	该值控制“当速度越大时”连续行走半径的衰减幅度	>0	2
stoppingPow	double	接近终点时速度的衰减幅度，如果小车停不下来，超过目标点，可以适当调大	>0	2
finSpeed	float	到达终点时的停车速度		2.1&2.2
trackFinDist	float	判断“已到达终点”的阈值	>0	2.1&2.2.2
firstAccuracy	double	初始旋转的对齐精度		2.2
degPrecision	float	TurnTo 方法的精度，即停止 TurnTo 动作的阈值		TurnTo 方法

特殊说明：

1．小车转向时出现超调，如原地转向出现超调，请适当减小 maxWspeed 以及 nffac

2．参数 float wayPointFinDist 由以上 wayPointFinSpeedFactor 、 wayPointFinSpeedPow 、 wayPointFinBase 计算得出

B.2 单舵轮前进模型



主方法 Follow 运行逻辑：

1．当从 A 站点(srcX，srcY)开往 B 站点(dstX，dstY)时，利用贴近 AB 直线的算法让小车不断往 AB 直线靠近（具体影响贴近直线的运动参数将在下面“参数作用”部分详细说明）

(1) 以小车当前位置为起点，根据正向追踪点 lookahead 参数和正向点跟踪镜像距离 mirrord 计算出目标点（lkx，lky）

(2) 在迭代前进的过程中不断微调目标点的位置，并利用 pd 算法修正前进角度并下发速度前往目标点。

2．当小车行进至投影点(pox，poy)与目标点(dstX，dstY)之间的距离 rd 小于 slowndownd 时减速，以较小速度继续执行 1。

3．当小车行进至投影点(pox，poy)与目标点(dstX，dstY)之间的距离 rd 小于 findist 时停车。

可调用方法：

1．IEnumerable<bool> SpinToAbsolute(double theta，double finThres = 3，int readyTime = 1500，float slowdownTh = 45)

用于转至指定角度的函数，返回是否转向成功的迭代器。theta 为目标角度，finThres 为停止转向的阈值，readyTime 为开始转向前和转向结束后的暂停运动时长，单位为 ms，slowdownTh 为减速转向的阈值。

2．IEnumerable<bool> Follow(TrajectoryReplanningObstacleAvoidance avoidance = null)

具体说明见“主方法 Follow 运行逻辑”，可选参数：TrajectoryReplanningObstacleAvoidance avoidance，用于实现避障。

参数说明：

参数名称	参数类型	参数说明	参数范围	具体起效步骤
baseSpeed	double	预设基础速度		1&2
refAheadBase	double	控制预测小车位置偏移量基础值，用于控制预测小车的位置，即 X—now 和 Y—now 的大小	>0	1.1
refAheadScale	double	控制预测小车位置变化量的比例,用于控制预测小车的位置，即 X—now 和 Y—now 的大小	0—1	1.1
lookAhead	double	车到路径垂足点向前的追踪量基础值,可调节小车贴近路径的快慢,过小容易超调,过大则调整过慢,建议高速长直线适当增大,低速高精度适当调小	>0	1.1
lookAheadFactor	double	追踪点前向偏移量与速度大小的线性系数,速度越大，la 越大	0—1	1.1

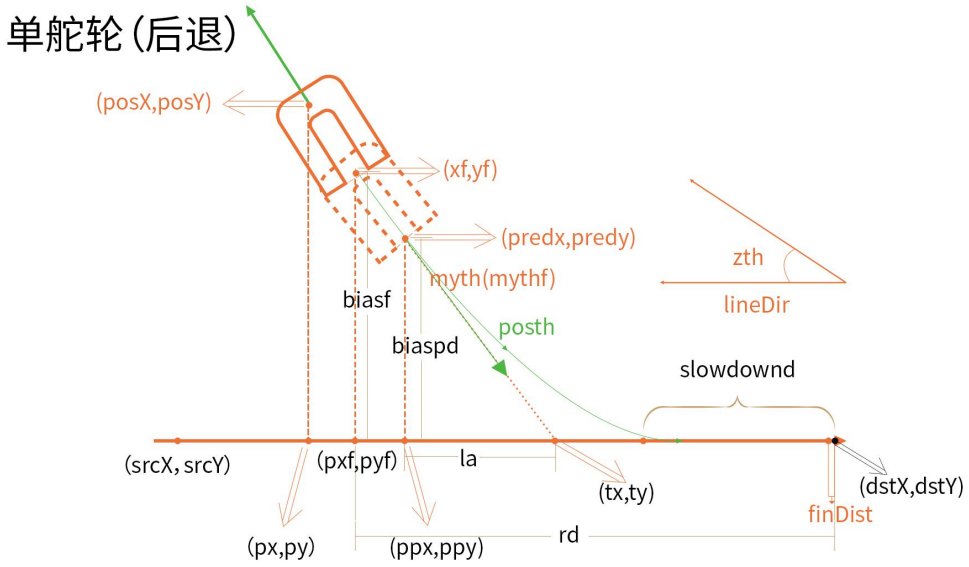
lookAheadPow	double	衰减速度对前向偏移量 la 的影响系数	0—1	1.1
mirrorThres	double	正向点跟踪镜像距离阈值	>0	1.1
mirrorScale	double	正向点跟踪镜像系数	0—1	1.1
mirrorPow	double	正向点跟踪镜像幂数	0—1	1.1
tail_l	double	舵轮中心到定轮的距离	>0	1.1
nffac	double	pid 控制中的 d，角度负反馈修正系数	0—1	1.2
turnSlowSigma	double	转弯减速角度范围		1.2
turnSlowWeight	double	转弯减速比例	0—1	1.2
thDiffPow	double	角度误差下发幂数	0—1	1.2
thDiffScale	double	角度误差下发倍数		1.2
angSigma	double	正向角度误差速度系数，控制同时转向和速度的，越大，则还没调好转向时就允许运动越小则表示必须完全调好转向再运动		1.2
dthBias	double	角度偏差，用于舵轮标定时调节舵轮直线度		1.2
dthScale	double	角度倍数		1.2
dthPow	double	角度幂数	0—1	1.2
angleThres	double	旋转角度最大值，限制舵轮最大转向角度	>0	1.2
rotAcc	double	加速度最小值	>0	1.2

dthfac	float	pid 中的 p，角度比例增益		1.2
thresAcc	double	最大加速度，调节小车加减速率	>0	1.2
thresWAcc	double	最大转弯加速度，调节小车加减速率	>0	1.2
slowdown	bool	是否启用减速逻辑	true/false	2
slowdownd	double	开启减速逻辑的距离阈值	>0	2
slowdownFac	double	减速幂数	0-1	2
findist	double	判断“已到达终点”的阈值	>0	3

特殊说明：

1. 参数 double mirrord 由 mirrorThres、mirrorScale、mirrorPow 计算得出

B.3 单舵轮后退模型



主方法 ReverseFollow 运行逻辑：

1．当从 A 站点(srcX，srcY)开往 B 站点(dstX，dstY)时，利用贴近 AB 直线的算法让小车不断往 AB 直线靠近（具体影响贴近直线的运动参数将在下面“参数作用”部分详细说明）

(1) 以小车当前位置为起点，根据正向追踪点 lookahead 参数计算出目标点（tx，ty）

(2) 在迭代前进的过程中不断微调目标点的位置，并利用 pd 算法修正前进角度并下发速度前往目标点。

2．当小车行进至投影点(pox，poy)与目标点(dstX，dstY)之间的距离 rd 小于 slowndownd 时减速，以较小速度继续执行 1。

3．当小车行进至投影点(pox，poy)与目标点(dstX，dstY)之间的距离 rd 小于 findist 时经过 stopWait 时间和 rdWait 时间的缓冲，期间不下发速度和角度，仅靠惯性稍稍滑行，最终实现停车 。

4．当 zth 大于 20 度时，即使小车已经到达停车前的减速或者最终停车距离，仍然使用算法调整角度。

可调用方法：

1．void ComputeBias()

用于计算小车行驶过程中的 bias、biasf 和 rd。

2．IEnumerable<bool> ReverseFollow()

具体说明见“主方法 Follow 运行逻辑”。

参数说明：

参数名称	参数类型	参数说明	参数范围	具体起效步骤
baseSpeed	double	预设基础速度		1&2&3
refAheadBase	double	控制预测小车位置偏移量基础值 ，用于控制预测小车的位置，即 predx 和 predy 的大小	>0	1.1
refAhead	double	控制预测小车位置变化量的比例，用于控制预测小车的位置，即 predx 和 predy 的大小	0—1	1.1
lookAhead	double	车到路径垂足点向前的追踪量基础值，可调节小车贴近路径的快慢，过小容易超调，过大则调整过慢，建议高速长直线适当增大，低速高精度适当调小	>0	1.1

lookAheadDamperDist	double	反向跟踪停车前若干距离，用于提高跟踪点距离从而使得小车不转向		1.1
forkLength	double	反向跟踪叉长，要略远于定轮位置	>0	1.1
thDiffPow	double	角度误差下发幂数	0—1	1.2
thDiffScale	double	角度误差下发倍数		1.2
angSigma	double	正向角度误差速度系数，控制同时转向和速度的，越大，则还没调好转向时就允许运动越小则表示必须完全调好转向再运动		1.2
dthBias	double	角度偏差，用于舵轮标定时调节舵轮直线度		1.2
dthPow	double	角度幂数	0—1	1.2
threshold	double	旋转角度最大值，限制舵轮最大转向角度	>0	1.2
thresAcc	double	加速度最大值	>0	1.2
slowdown	bool	是否启用减速逻辑	true/false	2
slowdownd	double	开启减速逻辑的距离阈值	>0	2
slowdownFac	double	减速幂数	0—1	2
finSpeed	double	减速时基础速度	>0	2
findist	double	判断“已到达终点”的阈值	>0	3
stopWait	int	停车前的等待滑行时间 1		3

rdWait	int	停车前的等待滑行时间 2		3

附录 C　仿真软件 CoppeliaSim 使用步骤

C.0 文件说明

以下为所需文件说明，具体其他车型仿真可以自行生成对应 Medulla 和 Clumsy 库文件进行仿

真

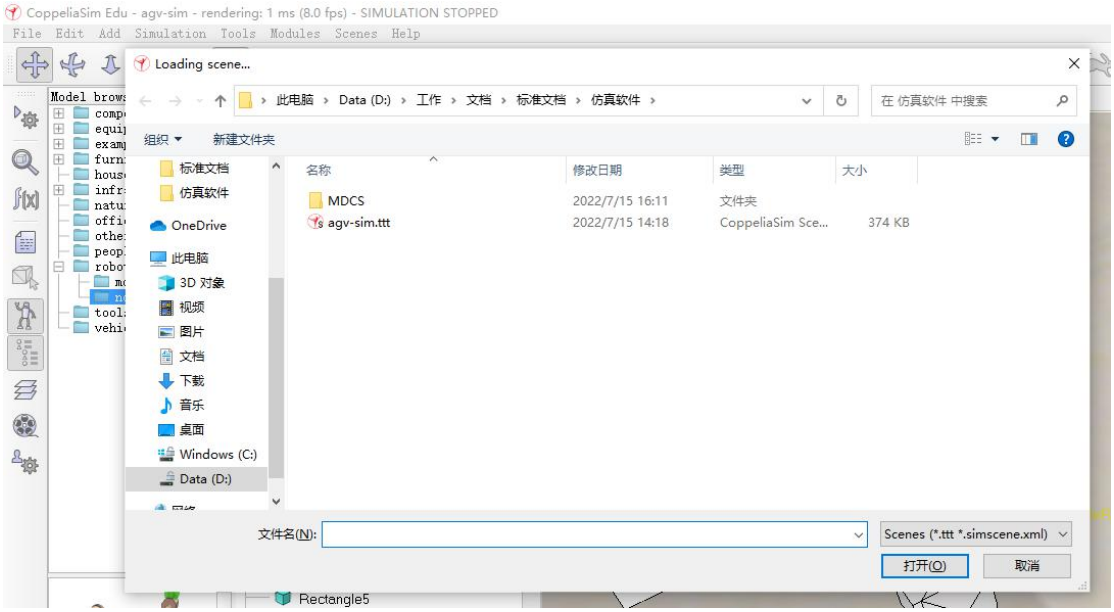
C.1 安装 CoppeliaSim 软件

1. 在 CoppeliaSim 官网下载软件： https://coppeliarobotics.com/downloads；
2. 选择版本进行下载（尽量选择最新版本）；

3. 下载完成后按步骤进行安装

C.2 导入文件

1. 导入 agv-sim.ttt 文件：打开 CoppeliaSim 软件，点击 File，选择 Open scene，选择 agv-sim.ttt 文件，点击导入。选择如下图所示



2. 在 Medulla 文件夹中导入 CoppeliaSim_m.dll 文件：将 CoppeliaSim_m.dll 放入 MDCS/Medulla/plugins 文件夹中
3. 在 Medulla 文件夹中导入 startup.iocmd 文件：将 startup.iocmd 文件放入 MDCS/Medulla 文件夹中

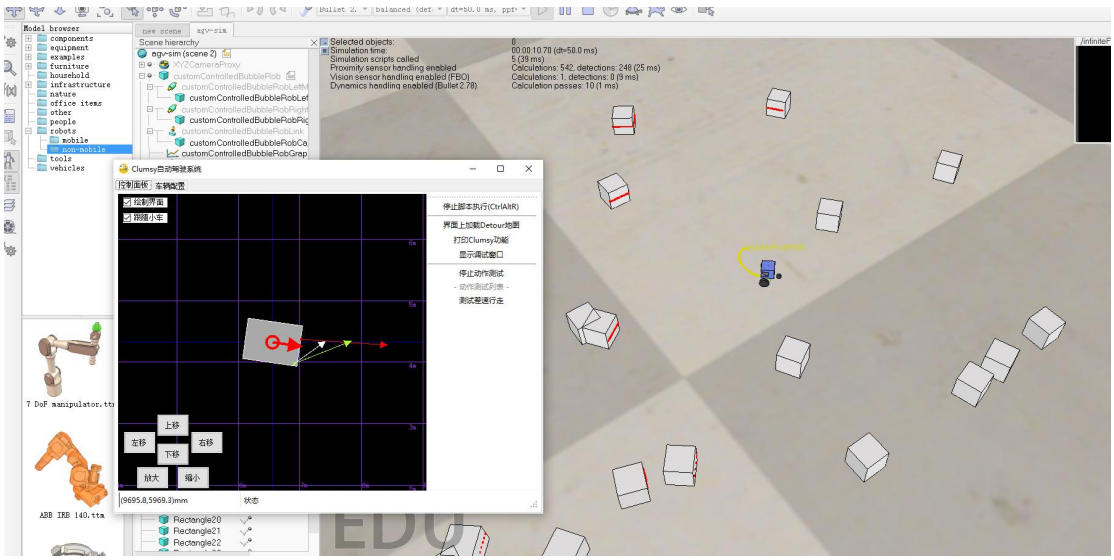
4. 在 Clumsy 文件夹中导入 CoppeliaSim_c.dll 文件：将 CoppeliaSim_c.dll 放入 MDCS/Clumsy 中

5. 修改 clumsy.json 文件配置：打开 clumsy.json 文件，将默认加载修改为 CoppeliaSim_c.dll；

6.

C.3 仿真测试

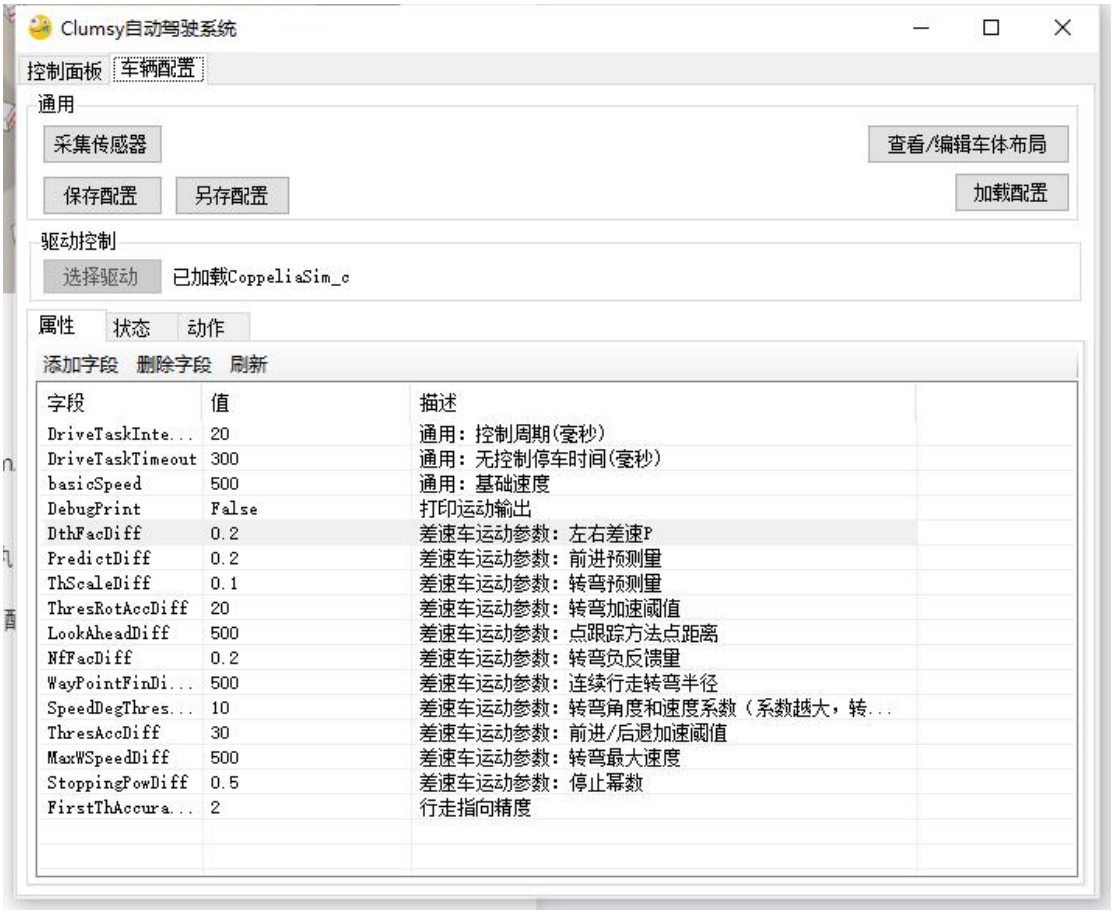
1. 启动 Medulla.exe 程序
2. 启动 ClumsyConsole.exe 程序
3. 点击 CoppliaSim 仿真软件中 start 按钮，然后在 Clumsy 中进行动作参数配置，仿真软件中的小车会按照动作参数进行运动。Medulla 示例中定义了差速左驱动右驱动 IO，Clumsy 示例中目前只定义了一个“测试差速行走”，如果需要其他动作测试请参考开发文档进行 Medulla 适配和 Clumsy 开发。



C4. 注意事项

1. 在 http://dl.lessokaji.com/上更新最新程序包进行测试
2. 如果启动后仿真程序后，在 Clumsy 中执行测试方法后，小车出现乱跑、原地旋转等现

象，请按照 MDCS 中运动参数配置说明进行参数调整，基础速度建议设置 10 左右，方便观察车体动作



当出现执行 Clumsy 动作测试方法后，仿真程序中车体没有对应动作，可以尝试 stop 当前仿真进程，重新点 start 进行加载