一、总体实现思路

本项目总共分为四个模块进行总体设计，分别为前端上位机，后端服务器，Matlab 模拟车，以及智能手机硬件平台的模拟车。

前端上位机基于 Vue 构建，这种框架的开发可以把前后台完全分离，前后端可以依据 rest 风格进行通信，前后台只在接口上有交集，代码上完全独立。

后端服务器基于 SpringBoot 构建，用于存储、记录工程中设计到的所有数据，同时进行部分路径的规划以及优化。

Matlab 模拟车采用 simulink 进行搭建，通过 UDP 网络通信方式与服务器进行信息交互，同时进行 PID 控制，平滑运行曲线。

智能手机硬件平台的模拟车采用 Android Stdio 进行开发，通过 UDP 网络通信方式向服务器传输位姿等信息。

二、实现细节

1.1 前端实现

1.1.1 Vue 框架

前端主要基于 Vue 构建。这种框架的开发可以把前后台完全分离，前端人员只负责前端开发，后端人员负责服务端开发。并且 Vue 有自己的请求处理方式，前后端可以依据 rest 风格进行通信，前后台只在接口上有交集，代码上完全独立。

1.1.2 基本构成

利用 Vue 提供的这套声明式的、组件化的编程模型，前端项目主要由以下几部分构成：①Components 和 Ui，负责基础组件的基本逻辑以及各页面的渲染功能。②Common 部分。包括基础的 2d 绘图函数以及基础的特征类如坐标类和 agent 类。③Service 和 Foundation 部分。作为单独的前端对外通信是必不可少的。单就此项目来说涉及的交互数据可能有 agent 状态和地图数据等。并且，对于不同的数据类型又可能采用不同的通信协议进行前后端传输。④其他部分，包括路由、主 App.vue 等。

1.1.3 页面渲染及 Ui 交互

地图使用基础栅格地图，以灰实线分隔，其中白色方格代表可行、黑色方格代表有障碍不可通行、红色方格代表当前车辆位置。如下图所示：

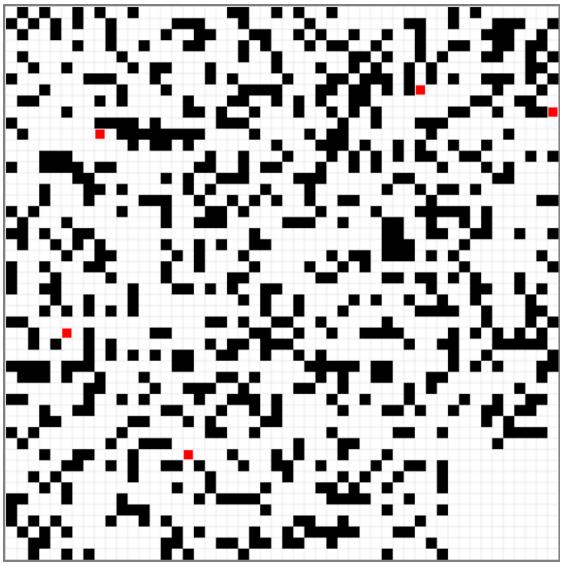

图 1.1.3.1 栅格地图

在实现基础的地图显示功能之外，我们还实现了简单的对指定的车辆指定目的地的交互功能。可以通过在地图单机指定位置，或直接在输入框中输入设定位置，按下按钮即可将目标坐标传输至后端。除此之外，还包括初始化地图，开始路径规划等功能。可参见下图：

图 1.1.3.2 交互部分

1.1.4 提供服务与远程通信

代码层面主要包括 4 个 services，抽象出了 4 个类。分别是包罗所有特征的 appservice.在他下一层是 world，其代表了当前仿真的世界。即 agents 和 map，这二者一个包括了每一个 agent 的状态，一个保存了“世界”地图中可行区与障碍区的坐标，是最基础的仿真数据，也是需要前后端进行传输的关键数据。

从功能来看，appservice 控制着整个世界的运行速度和开始、结束。Worldservice 负责与渲染部分的连接，使用渲染部分的方法，将经由后端处理传输而来的数据实时显示到用户的可视页面上。同时，agentservice 和 mapservice 在前端保存则一份世界的基础数据，并利用后端传来数据进行实时的删改与更新。由于前后端之间需要实现多种数据类型的传输，在此我设计了两套传输方案。其一是使用 axios 库向后端发送 http 请求，如 GET、POST 等。另一套是采用了基于 TCP 的 Websocket，它可以实现浏览器与服务器间的全双工通讯。

在首次使用时，需要对前端与服务器之间的数据进行统一，也就是从服务器得到最新的 map 与 agent 数据，并绘制与界面。由于此任务不需要多次重复执行且可能出现较大的数据量，在此直接采用 axios 库发送 http 请求到服务器进行最初的设置。于其相似的在设置服务端数据，如设定任务坐标时，同样采用此方案。但当仿真世界开始运行时，训练连续重复得到服务器计算的最新数据，更新仿真世界中各小车的位置。此时若仍使用 http 是很不合适的，当我们使用 http 协议探知服务器上是否有内容更新，就必须频繁的从客户端到服务器端进行确认。而 http 一下的这些标准会成为一个瓶颈：一条连接上只可以发送一个请求、请求只能从客户端开始。客户端不可以接收除了响应以外的指令、请求 / 响应首部未经过压缩就直接进行传输。首部的信息越多，那么延迟就越大、发送冗长的首部。每次互相发送相同的首部造成的浪费越多、可以任意选择数据压缩格式。非强制压缩发送。虽然 Websocket 借用了 http 的协议来完成一部分握手，但在连接创建后，在服务器和客户端之间交换数据时，用于协议控制的数据包头部相对较小。可以让 Vue 与服务器保持长连接，由此也就更强的实时性。由于协议是全双工的，所以服务器可以随时主动给客户端下方数据，在此表现为更新 agents 的实时状态。

1.2 服务器实现

1.2.1 SpringBoot

本项目的服务端采用了 SpringBoot 框架进行搭建。开发平台则采用 VSCode

与 Maven 包管理工具。对于类似于本工程的一些小项目，相较于直接使用 IDE，上述方案完全可以实现必要的开发。在此方案下，maven 通过 pom.xml 来描述项目，并提供了一系列插件来构建项目，源代码位于 scr/main/java 中，测试代码位于 scr/main/test 中，项目需要的配置文件则放在 scr/main/resources 目录下。通过在 pom 文件中添加 dependency，在 build 中添加 plugin 等方式可以轻松获取需要的开发各种资源。

1.2.2 Entity 与 Service

作为服务端自然是需要存储、记录工程中设计到的所有数据。由此演化出多个实体(Entity)，如 agent、map、world 等。这些实体与前端中的响应部分一一对应，在初次通信时变是将对应二者之间的数据更新统一。

在实体以外还要其对应的服务(Service)。一般来说，由实体提供对应数据存储的数据类型，而服务则会提供各种方法以供外部调用。以增删改查四类为主，当客户端发送请求或服务端需要处理数据上传客户端时即可调用。

1.2.3 Api 与 Websocket

在 Springboot 中我们使用 Controller 来对应处理不同的 URL 请求。比如来自 agents 和 map 的请求就分属两个 URL 也即两个 Controller。同时，对于同一个请求地址我还利用其内部传参来细分对于任务返回不同需求对于结果。由于本项目采用了前后端分离的设计理念，在通信时需要额外处理跨域请求问题，否则会被浏览器拦截。

上文提到，对于需要由服务器主动多次发送的数据，我采用了 Websocket 协议进行通信。在 SpringBoot 中搭建一个 Websocket 服务器也是较为任意的，再加上 Maven 强大的包管理机制可以轻易实现基础的 Websocket 功能。由于要发送的数据内容较为复杂，我选择将其编码为 json 字符串并插入特定的分隔符。在前端即可轻易划分并重新解码，实现数据更新与页面渲染。

1.2.4 Api 与 Websocket

本项目共设计有五台仿真车，其中三台为纯数字车，剩下的两台分别为 Matlab 和手机 App 的模拟车。为了简化服务端程序，减少不必要的代码，我们决定将二者与服务端通信的协议均设为 UDP。并且两台模拟车上传数据均采用相同的格式，便于服务端的接收与解包。

每一条数据由 48 个 byte 组成。开头一个 byte 系设备号，在服务端可直接使

用 uint8 解码，后接 7 个 byte 的 0。随后为 5 个 double 的数据部分。以 Matlab 上传数据为例子，分别为位置 x 坐标、位置 y 坐标、转向角度、行驶速度以及加速度。每当接收到 Matlab 的数据后服务端就可以更新当前车辆的状态，并及时回传最新的数据给 Matlab 端进行解析、处理、控制。

对于手机 App 的模拟车与此类似。由于位姿估计算法在手机端完成，服务器端任务相对简单。只需要接收的同时更新对应数字车的状态，并与其他车辆数据一齐传递给前端。前端各数据实时接收情况如下:

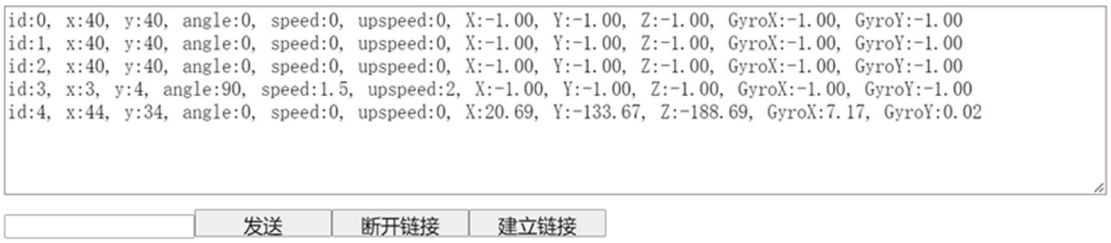

图 1.2.4.1 A*算法流程图

1.2.5 路径规划——A*算法

A*算法是一种重要的启发式算法，主要是用于在两点之间选择一个最优路径，而 A*的实现也是通过一个估值函数 F=G+H，其中 G 表示该点到起始点位所需要的代价 H 表示该点到终点的距离(如曼哈顿距离)。F 就是 G 和 H 的总和，而最优路径也就是选择最小的 F 值，进行下一步移动。

由于本项目是在栅格地图的基础上做路径规划，算法编程较为简单。同时考虑到数字车辆的目标点可能实时变化，我们选择每次都重新计算完整的路径，确保规划正确无误。而规划的频率则由前端设定的运行速度决定。A*算法流程图如下所示，其中“Open 表”存储可被访问的节点，“Close 表”存储已被访问过的节点（最优路径上的节点）。