哈 尔 滨 工 业 大 学

硕 士 学 位 论 文 中 期 报 告

智能重载 AGV 系统的研究

院　　（系）　机电工程与自动化学院

学　　科　　机械工程

导　　师　　王昕

研　究　生　　吴钟

学　　号　　15S153546

中期报告日期　　2017 年 3 月 10

研究生院制

二〇一七年三月

目 录

1. 主要研究内容及工作进度

1.1 课题的主要研究内容

本课题的主要研究内容是通过对四轮独立驱动中 AGV 进行运动学分析，实现车体前行、后退、原地旋转以及任意模式转弯的功能，通过对梯形加速度的分析和轨迹跟踪算法的研究，可以使 AGV 能够满足工作环境设定的要求；通过对 AGV 和载车板进行静力学分析，可以使得车体的结构更优化和更轻量化，从而使得 AGV 能够更合理地工作；通过对图优化 SLAM 的研究，对闭环检测的研究可以使得建立的地图更加的完整和精确，为后面基于 ROS 的导航的定位提供精确的地图。在研究的过程中，首先通过对四轮独立驱动 AGV 的运动学进行研究选择出适合出车体的运动学，从而能够使得 AGV 的四个独立驱动轮分配到正确的速度进而可以完成我们想要的运动；然后用 matlab 对我们的运动学和轨迹跟踪算法进行仿真加以验证，并在车体上验证算法的正确性和可行性；接着，对三种目前比较常见的建图的方案进行了比较，然后选择了图优化 SLAM 的进行了建图；最后在上述工作的基础上，提出将 DWA 算法和 backstepping 算法相融合替换 DWA 算法后，将整个系统移植到 ROS 上，验证重载 AGV 在 ROS 下导航的可行性。

1.2 课题的研究进度安排及完成情况

截止到目前为止，本课题主要完成的内容有：四轮独立驱动 AGV 的运动学分析；用 matlab 仿真了轨迹跟踪算法的可行性，并在 ROS 中用虚拟的环境验证了算法的正确性；研究了梯形加速度的方法，并已经在 AGV 上得到了检验；比较了 gmapping、hectormapping、kartoslam 之间的差异，并提出了用二维码对 kartoslam 的闭环检测加以增强的思想；并且已经在差速 AGV 上验证了基于 ROS 导航的可行，但是精度还没有得到提高；最后针对本课题，准备利用 Qt 和 ROS 相结合，制作出一款人机界面，以能够实时检测到 AGV 的运行状态以及传感器的运行状态。课题的研究进度安排以及完成情况如表 2-1 所示：

表 2-1 课题的研究进度安排及完成情况

时间	课题进展与预期目标	完成情况
2016.09.01——2016.09.22	收集资料，阅读文献，确定研究内容与研究方案，完成开题报告，准备开题答辩	完成
2016.09.23——2016.10.31	建立四轮独立驱动 AGV 运动模型，并对四轮独立驱动 AGV 进行运动分析，以及相应的电机选型进行研究	完成
2016.11.01——2016.12.31	用 matlab 仿真轨迹跟踪算法以及对 AGV 和载车板进行静力学仿真，进行轻量化设计	部分完成
2017.01.01——2017.02.28	研究图优化 SLAM，比较几种建图方案的优缺点，提出用二维码辅助闭环检测的思想，用 kartoslam 建立地图；用差速 AGV 将整个系统基于 ROS 系统下的导航调通	完成
2017.03.01——2017.04.01	利用 Qt 和 ROS 相结合，设计一款人机交互界面	未完成
2017.04.02——2017.05.31	将实验期间取得的相关成果进行整理，并发表一篇学术小论文	未完成
2016.06.01——2016.07.31	整理研究成果，撰写、修改、完善硕士学位论文	未完成
2016.08.01——2016.12.20	准备硕士学位论文答辩	未完成

2.目前已完成的主要研究工作及结果

2.1 四轮独立驱动 AGV 运动学和静力学分析

四轮独立驱动 AGV 的每个轮子拥有独立驱动和转向的功能，通过对四轮独立驱动 AGV 进行运动学分析可以使 AGV 能够实现前行、后退、横移、原地旋转、任意方向的移动以及任意方向的转弯。

2.1.1 重载 AGV 运动学分析

2.1.1.1 机器人位置表示

在 AGV 的运动分析中，AGV 被建模成运动在水平面轮子上的一个刚体并忽略 AGV 与轮子之间内在的关联和约束。在平面上，AGV 的维数为 3,2 个沿平面方向的移动和绕垂直轴的转动，为了描述 AGV 在平面中位置和姿态，我们建立如图 2.1 所示的世界坐标系和机器人坐标系。

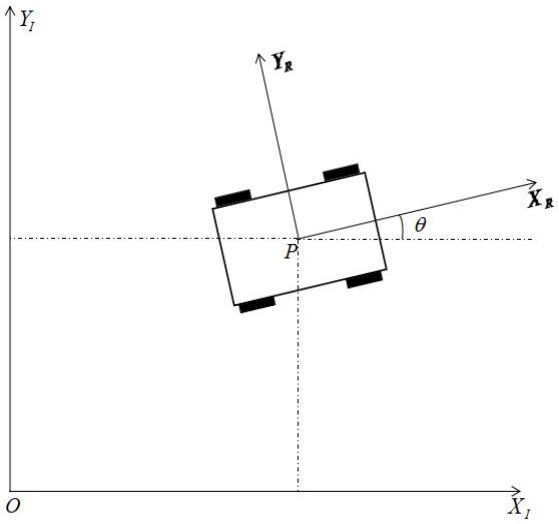


图 2-1 世界坐标系与机器人坐标系

$\{X_I, Y_I\}$是世界坐标系的框架，O可以是惯性平面内的任意一点，选取机器人的质心P作为机器人在世界坐标系中的参考点，以P为原点沿着 AGV 前进方向和侧移方向建立局部坐标系$\{X_R, Y_R\}$。在世界坐标系中，P的位置由其坐标值(x, y)确定，世界坐标系和机器人坐标系之间的角度差由θ确定，因此我们可以用一个向量来描述机器人在世界坐标系中的位姿：

$$\varepsilon_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{2.1}$$

其中下标I表示机器人姿态参考的基是世界坐标系。

为了能够在机器人坐标系中描述机器人的运动，需要将沿着世界坐标的运动映射到机器人坐标系中：

$$x_R = x \times \cos\theta + y \times \sin\theta \tag{2.2}$$

$$y_R = -x \times \sin\theta + y \times \cos\theta \tag{2.3}$$

$$\theta = \theta \tag{2.4}$$

用ε_I表示机器人在世界坐标系中的姿态，ε_R表示在机器人坐标系中的姿态，联合上面的三个公式，得到如下的公式，其中$R(\theta)$是映射的正交旋转矩阵：

$$\varepsilon_R = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \varepsilon_I = R(\theta)\varepsilon_I \tag{2.5}$$

同样我们可以得到机器人运动在机器人坐标系中的表示：

$$\varepsilon_R' = R(\theta)\varepsilon_I' \tag{2.6}$$

2.1.1.2 AGV 的转向模式

转向系统是地面行走装置的重要组成部分，用来操纵行走装置的行走方向。常见的轮式地面转向方式分为以下五类：四轮完全独立转向、阿克曼转向、腰部关节转向、滑移转向和合成轴转向[2]。这五类结构形式如图 2.2：

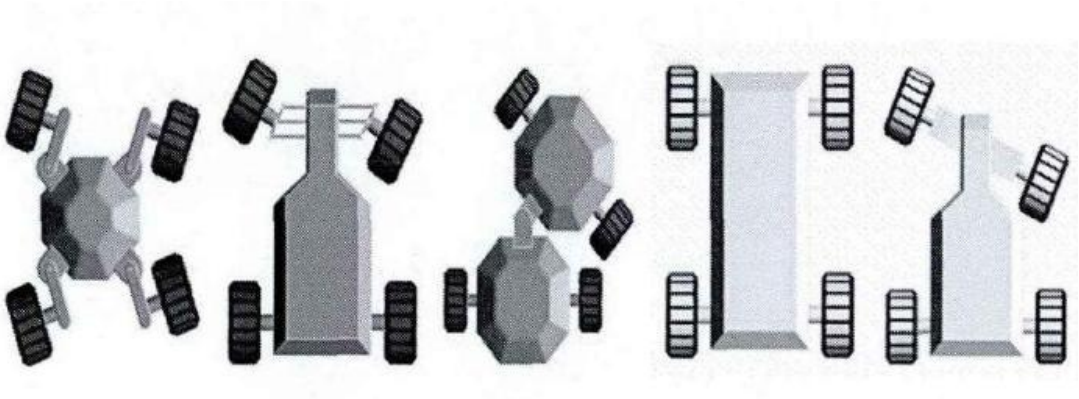


图 2-2 五类转向类型

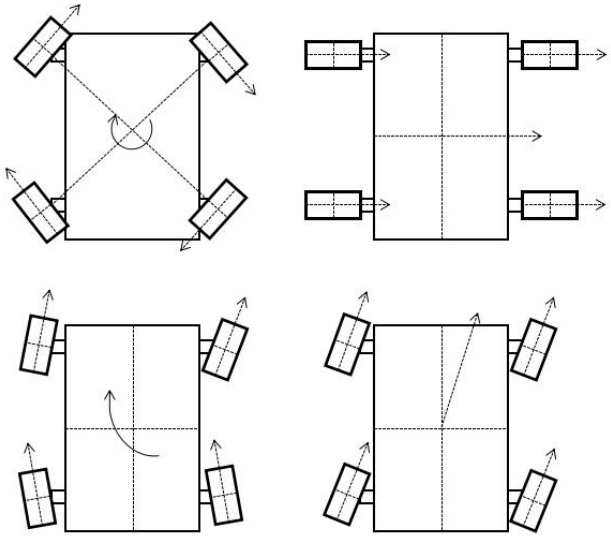
五种不同类型的转向模式具有不同的特点，表 2.1 对这几种模式进行了简单的比较。

表 2-2 五种基本转向方式对比

	四轮独立	阿克曼	腰部关节	滑移	合成轴
结构复杂度	高	高	低	低	低
控制复杂度	低	低	中	低	中
转向驱动功率	中	低	中	高	低
转向关节数	4	2	1	0	1
操作灵活性	中	中	中	高	中

每种转向方式都有自己优点，同时也有一定的缺点。对于阿克曼转向结构而言，由于需要转向桥装备将会使得 AGV 的底盘尺寸变大不适用重载托运 AGV 的要求。而关节转向和合成轴转向的转向半径过大，转向的灵活性较差。而且关节转向需要同车轮连在一起，将限制底盘其他部分的分布，与重载托运 AGV 的性能要求不符合。滑移转向由于转向时消耗功率高，与 AGV 的续航能力矛盾；其次滑移转向可以通过原地转向和直线行驶实现任意行驶轨迹，但是却无法实现 90° 的极限转弯；第三，滑移转向会对转向机构产生极大的力，可能对转向机构产生极大的破坏；第四，由于转向时产生的极大的力，对地面和车轮都会产生极大的磨损[2]，这些缺点正是 AGV 性能极力避免的。虽然四轮完全转动对需要 8 个电机分别控制轮子的转向和轮子的驱动，但是四轮完全驱动最为灵活，可以实现原地转向、横移转向以不同的相位转向以及相同位转向，转向过程

中四个轮子做纯滚动，消耗也最小，而且转向的过程中不需要差速器和转向器，这样就可以使得 AGV 尺寸特别是高度方向上的尺寸变小。



（a）原地转向；　（b）横移；　（c）不同相位转向；　（d）相同相位转向

图 2-3 四轮完全转向模式

综合考虑，我们的重载拖拽 AGV 的转向模式选择四轮完全转向方式，其中图 2-4 和图 2-5 分别是四轮在 AGV 三维模型上的分布位置和行走部分的布置：

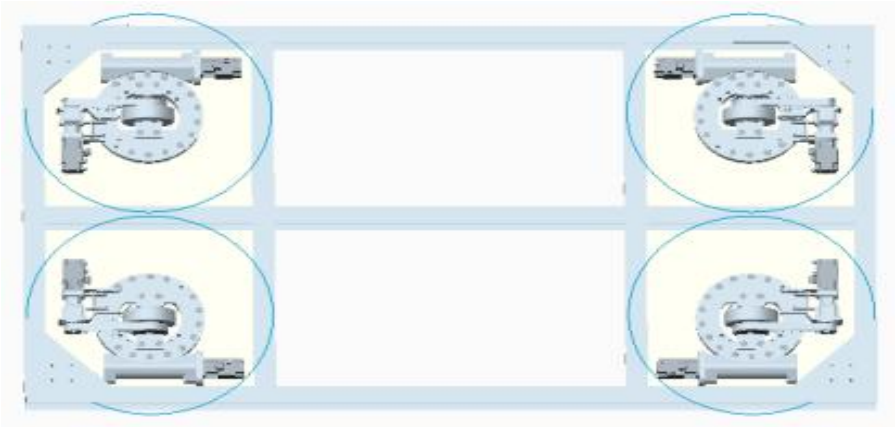
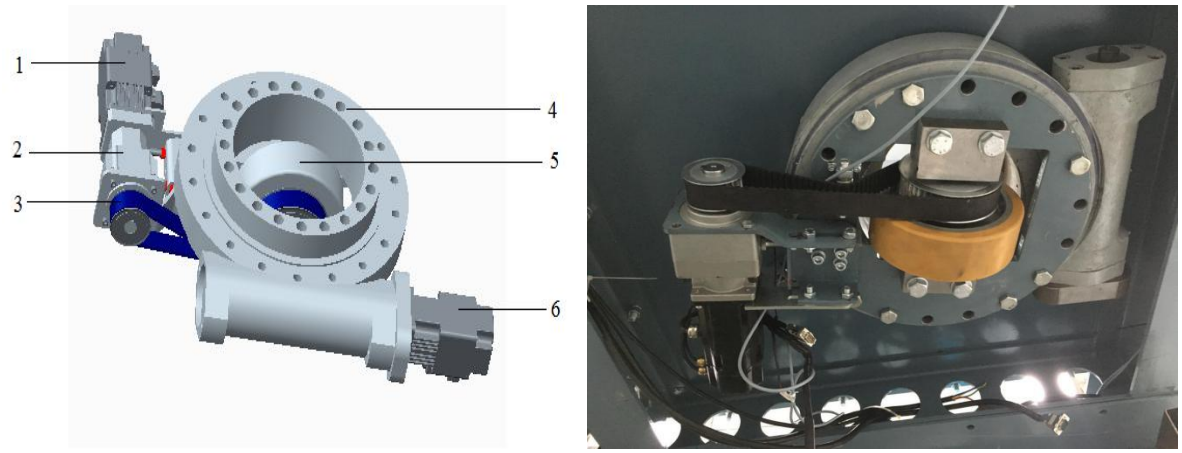


图 2-4　AGV 上四轮的安装位置



1 伺服电机　2 减速机　3 同步带　4 蜗轮蜗杆　5 车轮　6 蜗轮蜗杆电机

图 2-5 四轮完全转向行走部分布置

采用这种行走装置在一定程度上减小了车体的高度方向的尺寸，使得机器人能够在平面运行的更加稳定。采用四轮完全转向使得轮式行走装置的路径规划、轨迹跟踪问题变得简单，在狭小环境里的工作能力大大提高。

2.1.1.3 固定标准轮和受操纵标准轮的运动学约束

固定标准轮没有可操纵的垂直转动轴，因此它相对于底盘的角度是固定，因此限制了沿轮子平面前后的运动和围绕与地面接触点的转动[1]。图 2-7 刻画了固定标准轮，并说明了它相对于机器人局部坐标系$\{X_R, Y_R\}$的位姿。在机器人局部坐标系中，A的位置用距离l和角度α表示。其中β是轮子平面现对于底盘的角度，因为轮子的类型是固定标准轮，因此β是一个固定值。

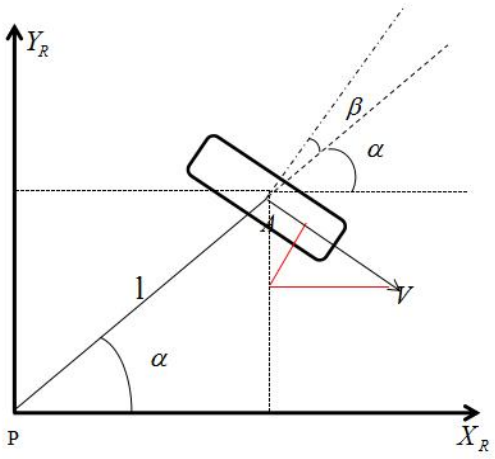


图 2-7 固定标准轮和它的参数

此时将速度向量投影到车轮对应的方向上，也就是向量$(\cos(\alpha+\beta-\pi/2), \sin(\alpha+\beta-\pi/2)$上，此时需要满足车轮的纯滚动的条件，可以得到下

面的公式：

$$\left[-\sin(\alpha+\beta)\ \cos(\alpha+\beta)\ l\cos\beta\right]R(\theta)\ \varepsilon_I'+r\varphi'=0 \qquad (2.7)$$

同理将速度向量投影到轴向量（$\cos(\alpha+\beta)$，$\sin(\alpha+\beta)$ 上，此时需要满足车轮无侧移的条件，得到如下的公式：

$$\left[\cos(\alpha+\beta)\ \sin(\alpha+\beta)\ l\sin\beta\right]R(\theta)\varepsilon_I'=0 \qquad (2.8)$$

受操纵标准轮与固定标准轮的差别只在于受操纵轮添加了一个附加的自由度：轮子通过垂直轴中心与地面接触，围绕垂直轴转动。受操纵轮位置表示如图 2-8 所示，其中参数的意义和固定标准轮的意义相同，(r,φ) 分别是轮半径和轮子转动角度。

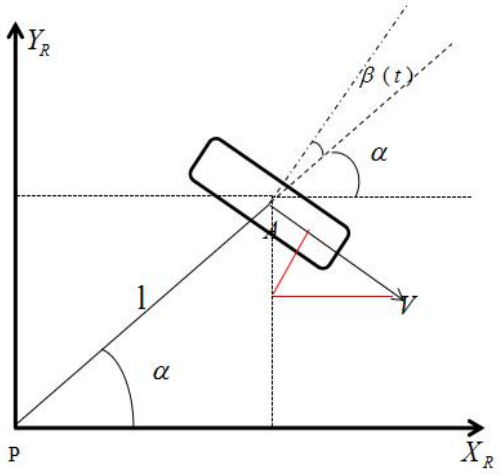


图 2-8 受操纵标准轮和它的参数

根据假设，车轮与地面是纯滚动和无滑动的特点，我们得到如下的受操纵标准轮的约束方程：

纯滚动约束：

$$\left[-\sin(\alpha+\beta(t))\quad \cos(\alpha+\beta(t))\quad l\cos\beta(t)\right]R(\theta)\varepsilon_I'+r\varphi'=0 \qquad (2.9)$$

无滑动约束：

$$\left[\cos(\alpha+\beta(t))\quad \sin(\alpha+\beta(t))\quad l\sin\beta(t)\right]R(\theta)\varepsilon_I'=0 \qquad (2.10)$$

其中 $R(\theta)$ 是正交旋转矩阵，ε_I 是机器人在世界坐标系中的位姿表示。

2.1.1.4 AGV 转向运动学分析

四轮独立驱动机器人和转向机器人是有四个伺服电机和四个轮毂分别驱动和控制转向，属于冗余驱动的移动机器人，利用冗余驱动这一特点，可以实现在实现控制移动机器人普通运动的同时同时实现例如优化各电机力矩的输出、避障等附加的运动，非常适合在一些狭小的空间、机动性能要求较高的场合。在这里主要对前轮转向后轮驱动、前

轮转向与驱动、前后轮反向偏转、前后轮同向偏转、原地旋转模型。

假设 AGV 是一个刚体并在在理想的平面上运动，AGV 车轮地面发生的是纯滚动并且没有侧移，在整个运动过程中忽略车轮的变形，车轮的平面始终保持垂直于地面。建立如图 2.9 的前轮转向后轮驱动的差速计算的原理图。

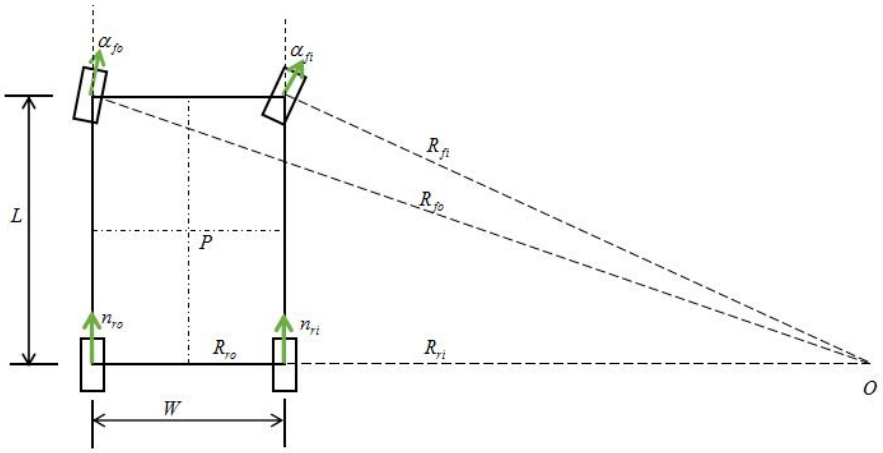


图 2-9 前轮转动后轮驱动的差速计算的原理图

图中的参数的意义如下表：

表 2-3 参数的意义

W	前后轮距	α_{fi}	前轮内侧偏转角
L	轴距	α_{fo}	前轮外侧偏转角
R_{fo}	前轮外侧转动半径	n_{ri}	后轮内侧转速
R_{fi}	前轮内侧转动半径	n_{ro}	后轮外侧转速
R_{ro}	后轮外侧转动半径	R_{ri}	后轮内侧转动半径

根据转弯的定义：以最低稳定车速转向行驶时，外侧转向的中心平面的支承平面上滚过的轨迹圆半径，故α_{fo}为移动机器人的转弯的半径。假设前外侧偏转角α_{fo}已知，根据三角函数的关系可以得到：

$$R_{fo} = L / \sin(\alpha_{fo}) \tag{2.11}$$

$$R_{ro} = R_{fo} \times \cos(\alpha_{fo}) = L \times \cot(\alpha_{fo}) \tag{2.12}$$

$$R_{ri} = R_{ro} - W = L \times \cot(\alpha_{fo}) - W \tag{2.13}$$

$$R_{fi} = \sqrt{R_{ri} \times R_{ri} + L \times L} = \sqrt{L \times L + (L \times \cot(\alpha_{fo}) - W) \times (L \times \cot(\alpha_{fo}) - W)} \tag{2.14}$$

$$\tan(\alpha_{fi}) = L/(L\times\cot(\alpha_{fo}) - W) \tag{2.15}$$

此时我们可以得到后两轮之间的转速比：

$$\frac{n_{ri}}{n_{ro}} = \frac{R_{ri}}{R_{ro}} = \frac{L\times\cot\alpha_{fo} - W}{L\times\cot\alpha_{fo}} \tag{2.16}$$

因此当我们知道转弯的偏转角是多少的时候，并且是用前轮转向后轮驱动的方式工作时，并根据当前的车身的角速度可以很快地计算出驱动轮的转速的大小从而实现转弯的功能。

建立如图 2.10 所示前轮驱动前轮转向的差速计算原理图，其中的参数的意义和前轮转动后轮驱动的意义是相同。

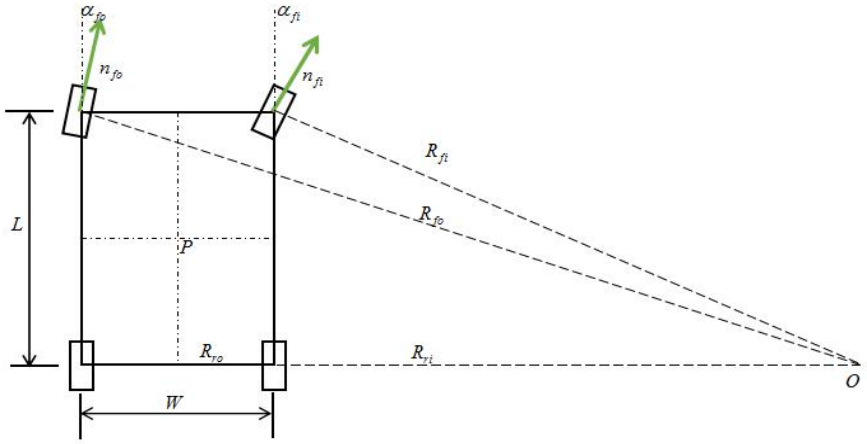


图 2.10 前轮转动前轮驱动的差速计算的原理图

假设已知转弯角度，根据图中的三角关系，我们可以得到移动机器人的转弯半径：

$$R_{fo} = L/\sin(\alpha_{fo}) \tag{2.17}$$

得到后轮内侧的半径为：

$$R_{ri} = L\times\cot(\alpha_{fo}) - W \tag{2.18}$$

进而求得前轮内侧的偏转角度为：

$$\tan(\alpha_{fi}) = L/(L\times\cot(\alpha_{fo}) - W) \tag{2.19}$$

前轮驱动轮的转速比为：

$$\frac{n_{fi}}{n_{fo}} = \frac{R_{fi}}{R_{fo}} = \frac{L/\sin\alpha_{fo}}{L/\sin\alpha_{fi}} = \frac{\sin\alpha_{fi}}{\sin\alpha_{fo}} \tag{2.20}$$

建立如图 2. 11 所示的前后轮反向偏转差速计算原理图：

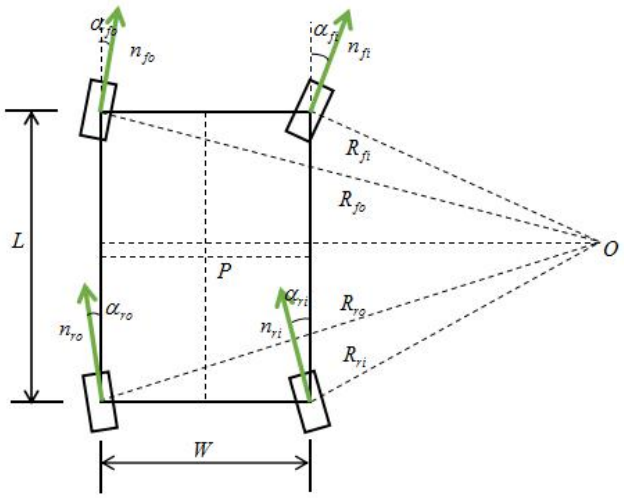


图 2.11 前后轮反向偏转差速计算原理图

其中W是轴距，L是前后轮轮距，α_{fo}是前轮外侧偏转角，α_{fi}是前轮内侧偏转角，α_{ro}是后轮外侧偏转角，α_{ri}是后轮内侧偏转角，n_{fo}是前外轮转速，n_{fi}是前内轮转速，n_{ro}是后外轮转速，n_{ri}是后内轮转速，R_{fo} R_{fi} R_{ro} R_{ri}分别是前外轮旋转半径、前内轮旋转半径、后外轮旋转半径、后内轮旋转半径，根据原理图中的几何关系，引进两个临时变量m r，然后根据三角关系可以得到如下的一系列公式：

移动机器人的转弯半径为：

$$R_{fo} = m / \sin \alpha_{fo} \tag{2.21}$$

移动机器人前内侧转弯半径为：

$$R_{fi} = m / \sin \alpha_{fi} \tag{2.22}$$

因此前外侧和前内侧车轮的转速关系为：

$$n_{fi} = \frac{R_{fi}}{R_{fo}} n_{fo} = \frac{\sin \alpha_{fo}}{\sin \alpha_{fi}} n_{fo} \tag{2.23}$$

同理得到后外侧与前外侧的车轮的转速关系为：

$$n_{ro} = \frac{R_{ro}}{R_{fo}} n_{fo} = \frac{(W + r) / \cos \alpha_{ro}}{(W + r) / \cos \alpha_{fo}} n_{fo} = \frac{\cos \alpha_{fo}}{\cos \alpha_{ro}} n_{fo} \tag{2.24}$$

同理得到后内侧与前外侧的车轮的转速的关系为：

$$n_{ri} = \frac{R_{ri}}{R_{fi}} n_{fi} = \frac{r / \cos \alpha_{ri}}{r / \cos \alpha_{fi}} n_{fi} = \frac{\cos \alpha_{fi}}{\cos \alpha_{ri}} n_{fi} = \frac{\cos \alpha_{fi}}{\cos \alpha_{ri}} \times \frac{\sin \alpha_{fo}}{\sin \alpha_{fi}} n_{fo} \tag{2.25}$$

我们发现当轮外侧的偏转角相等，轮内侧的偏转角相等的时候，前后两个内侧的轨迹圆重合，此时机器人具有最小的旋转半径。

建立如图 2.12 所示的前后轮同向偏转差速计算原理图：

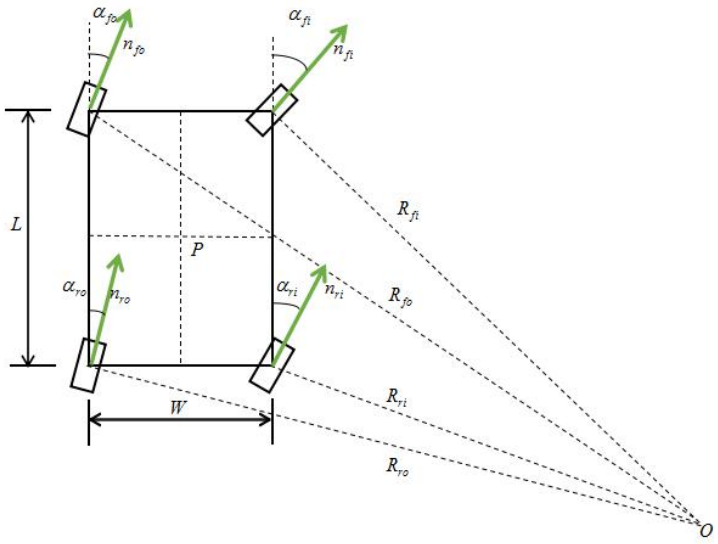


图 2.12 前后轮反向偏转差速计算原理图

前外侧与前内侧的车轮的转速比为：

$$n_{fi}=\frac{R_{fi}}{R_{fo}}n_{fo}=\frac{\sin\alpha_{fo}}{\sin\alpha_{fi}}n_{fo} \tag{2.25}$$

后外侧与前外侧的车轮的转速比为：

$$n_{ro}=\frac{R_{ro}}{R_{fo}}n_{fo}=\frac{(W+r)/\cos\alpha_{ro}}{(W+r)/\cos\alpha_{fo}}n_{fo}=\frac{\cos\alpha_{fo}}{\cos\alpha_{ro}}n_{fo} \tag{2.26}$$

后内侧与前外侧的车轮的转速之比为：

$$n_{ri}=\frac{R_{ri}}{R_{fi}}n_{fi}=\frac{r/\cos\alpha_{ri}}{r/\cos\alpha_{fi}}n_{fi}=\frac{\cos\alpha_{fi}}{\cos\alpha_{ri}}n_{fi}=\frac{\cos\alpha_{fi}}{\cos\alpha_{ri}}\times\frac{\sin\alpha_{fo}}{\sin\alpha_{fi}}n_{fo} \tag{2.27}$$

在这种转向控制的情况下，需要注意当轮外侧偏转角相等，轮内侧的偏转角分别相等的时候，轨迹的瞬时圆心在无穷远处。

当 AGV 出于狭小的空间，需要 AGV 具有原地回转的能力，这个时候需要保持轮子的速度一致并保持一定的偏转角就可以实现，实现原地回转的原理图如 2.13 所示。当 AGV 需要斜行运动的时候，只需要保持所有的车轮的转速一致，偏转角一致就可以实现。

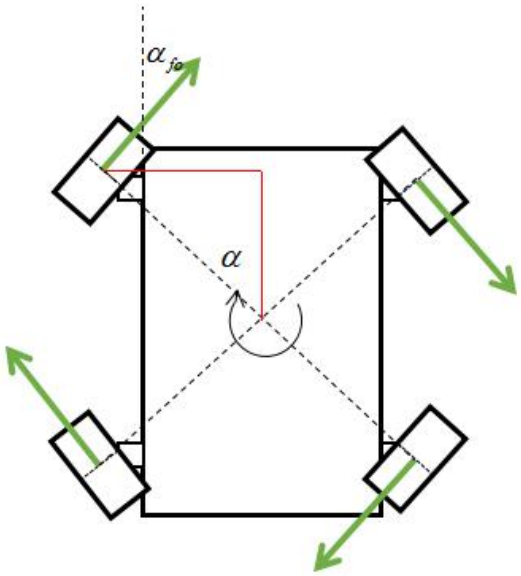


图 2.13 原地回转原理图

此时 AGV 的车轮的偏转角为：

$$\tan\alpha_{fo} = L/W \tag{2.28}$$

从图中我们可以看到，当 AGV 原地转向的时候，需要保证对角轮的偏转角保持一致，并且每个轮子的角速度一样才能实现原地的转向功能。

2.1.2 载车板的静力学分析

本文研究的重载 AGV 的载车板和车架是占据整个车体重量比例的最大的部件，所以首先需要对车架进行轻量化设计，在这个研究中，用 Creo 对车体进行建模，然后倒入到 Creo 中自带的 ANSYS Workbench 中进行有限元分析。

2.1.3 梯形加速度运动规划

梯形加速度为控制电机系统的启动和停止提供了平滑的动作，对 AGV 的平滑运动与控制有着重要的意义，在我们设计的 AGV 中给定的加速度和最大速度的参数分别是：1m/s^2 和 1m/s，我们设定的一个梯形加速度有三个区域，分别是加速区域、恒速区域和减速区域，对应的梯形速度曲线图如图 2-14 所示。

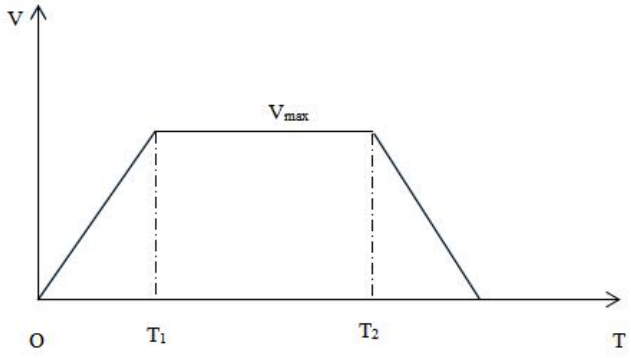


图 2-14 梯形加速度曲线图

在进行加减速的时候，其流程图 2-15 所示，在本次的梯形加速度设计之中我们假定加减速的加速度要求是一样的，起始点的坐标为(x1,y1)和目标点的坐标为(x2,y2)，加速度的大小是 1m/s^2，速度的大小是 1m/s。

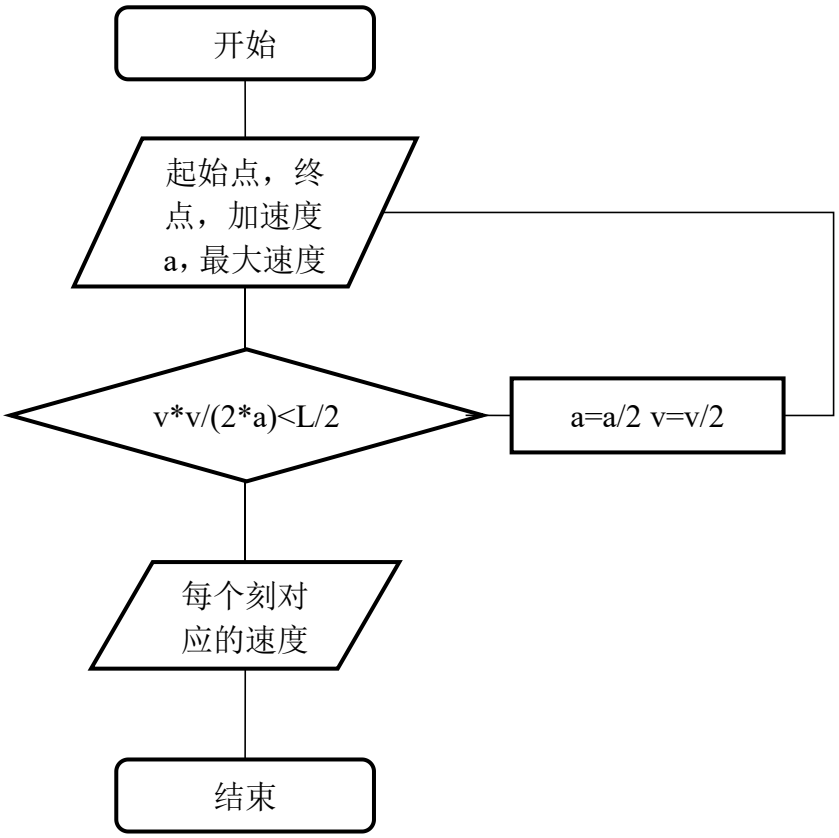


图 2-15 梯形加速度规划流程图

2.1.4 轨迹跟踪算法及实验结果

以四轮独立驱动 AGV 作为研究对象，其中的跟踪控制的核心思想是：有路径规划算法给出 AGV 在参考点的参考位姿 p_r 和参考速度 q_r，参考位姿 p_r 与反馈得到的 AGV 的实际位姿进行比较，得到 AGV 的全局位姿误差矢量，经过变换矩阵 T 转换进而得到局部位置误差矢量 e。在得到的位姿误差 x_e 和 y_e 之后加上参考速度 q_r 一起作为路径轨迹跟踪的输入，进而控制 AGV 能够得到正确的速度而尽快的沿着轨迹进行运动。

假设 AGV 的当前的实际位姿时 $p_c=[x\ y\ \theta]^T$，任意给定的期望位姿是 $p_r=[xr\ yr\ \theta r]^T$，其中 x，y 是机器人的位置表示，θ 是机器人的姿态角表示。位姿误差 $e=[xe\ ye\ \theta e]^T$，当车体中心的实际驱动速度为 $q_c=[v\ \omega]^T$，期望的 AGV 中心的速度为 $q_r=[vr\ \omega r]^T$，其中 vr 和 ωr 是参考的机器人中心的线速度和角速度。AGV 的参考图 2-16 所示

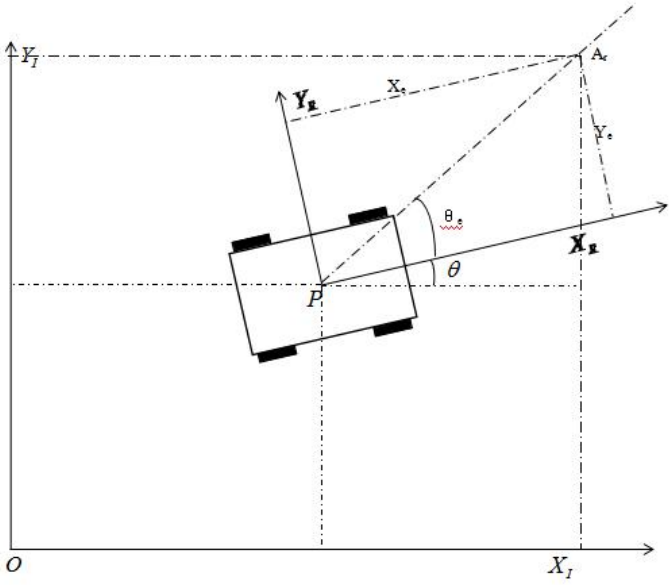


图 2-16 轨迹跟踪示意图

其中 AGV 的运动学方程是：

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}=\begin{bmatrix}\cos\theta & 0\\ \sin\theta & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}=S\begin{bmatrix}v\\ \omega\end{bmatrix} \tag{2.29}$$

式中 S 是一个 Jacobian 矩阵，AGV 在坐标内的位姿误差描述为：

$$e=T(p_r-p_c) \tag{2.30}$$

其中式 T 表示的是一个转换矩阵，将世界坐标系中的位姿转转为 AGV 坐标系的位姿。

根据期望的位姿 $p_r=[xr\ yr\ \ \theta\ r]^T$ 和当前的机器人的实际的位姿 $p_c=[x\ y\ \ \theta\ ]^T$ 可以计算出位姿误差为：

$$e=\begin{bmatrix}x_e\\ y_e\\ \theta_e\end{bmatrix}=\begin{bmatrix}\cos\theta & \sin\theta & 0\\ -\sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_r-x\\ y_r-y\\ \theta_r-\theta\end{bmatrix} \tag{2.31}$$

AGV 位姿误差的微分方程为：

$$\dot{e}=\begin{bmatrix}\dot{x}_e\\ \dot{y}_e\\ \dot{\theta}_e\end{bmatrix}=\begin{bmatrix}y_e & -1\\ -x_e & 0\\ -1 & 0\end{bmatrix}\begin{bmatrix}\omega\\ v\end{bmatrix}+\begin{bmatrix}v_r\cos\theta_e\\ v_r\sin\theta_e\\ \omega_r\end{bmatrix} \tag{2.32}$$

对 AGV 位姿误差微分方程构造 Lyapunov 函数

$$V=\frac{1}{2}(x_e^2+y_e^2)+2(1-\cos\frac{\overline{\theta}_e}{2}) \tag{2.33}$$

其中 $\bar{\theta}_e = \theta_e + \arctan(v_r, y_e)$

对上面的式子进行求导，我们可以得到：

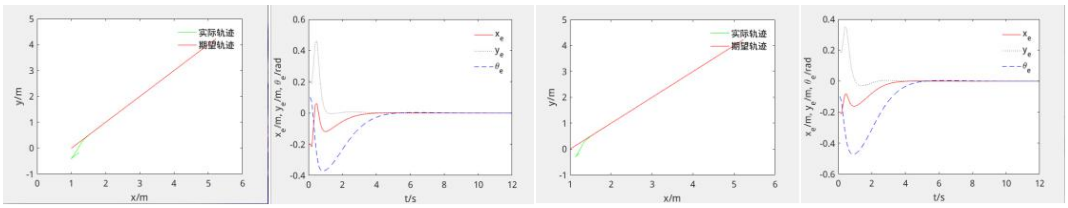
$$V' = x_e(y_e\omega - v + v_r\cos\theta_e) + y_e(-x_e\omega + v_r\sin\theta_e) + \sin\frac{\bar{\theta}_e}{2}[\omega_r - \omega - \frac{\partial\psi}{\partial v_r}v_r - \frac{\partial\psi}{\partial y_e}(-x_e\omega + v_r\sin\theta_e)]$$

$$= x_e(-v + v_r\cos\theta_e + \sin\frac{\bar{\theta}_e}{2}\frac{\partial\psi}{\partial y_e}\omega) + y_e v_r\sin\theta_e + \sin\frac{\bar{\theta}_e}{2}[\omega_r - \omega - \frac{\partial\psi}{\partial v_r}v_r - \frac{\partial\psi}{\partial y_e}v_r\sin\theta_e + 2y_e v_r\cos(\psi + \frac{\bar{\theta}_e}{2})] \quad (2.34)$$

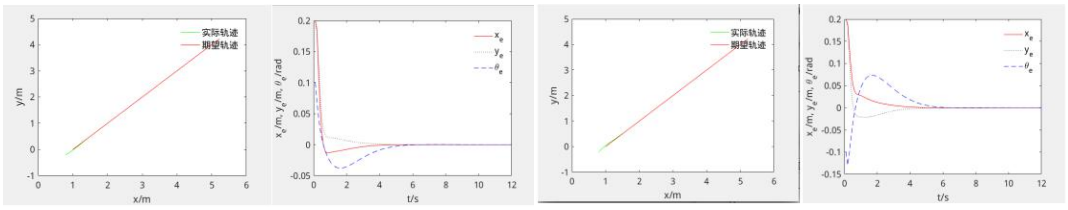
然后我们可以得到 AGV 实际应该运行的速度为：

$$q = \begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} v_r\cos\theta_e + \dfrac{\partial\psi}{\partial y_e}\sin\bar{\theta}_e + c_1 x_e \\ \omega_r - \dfrac{\partial\psi}{\partial v_r}v_r - \dfrac{\partial\psi}{\partial y_e}v_r\sin\theta_e + 2y_e v_r\cos(\psi + \dfrac{\bar{\theta}_e}{2}) + c_2\sin\dfrac{\bar{\theta}_e}{2} \end{bmatrix} \quad (2.35)$$

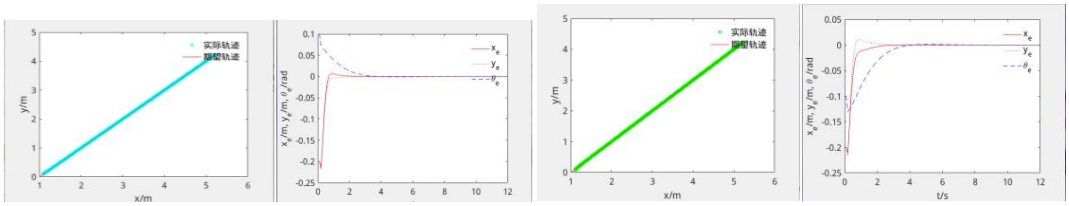
在得到了上诉的轨迹跟踪的方案之后，我们在 matlab 里面进行了仿真，已验证算法的正确性和可行性，在 matlab 的仿真实验环境中，一共进行了 8 组实验，在这 8 组实验中，我们设定最后的机器人应该的达到的速度为 0.2m/s，角速度为 0rad/s，分别使得机器人分别位于需要寻找轨迹的上前、上后、下前和下后，每个里面又分为机器人实际的位姿与轨迹位姿的夹角是正值还是负值两种情况，下图 2-17 就是仿真得到的实验结果图。



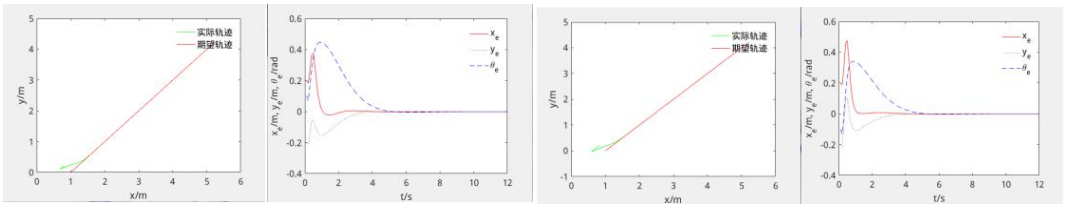
图（a）



图（b）



图（c）



图（d）

图 2-17 寻迹算法的仿真图

由上面的仿真图我们可以看到，机器人在不同位置的时候，都能够实现最后达到最终直线轨迹的功能，唯一的缺陷是，中间会有很大的速度和角速度的跳跃，这一点是需要根据实际情况改进的。

2.2 重载 AGV 系统的搭建

AGV 运动控制和 AGV 车体设计有着密切关系，而控制器是用于控制车辆的速度和方向，保持车辆运行重载期望的路径上面，控制器算法要在 AGV 车辆负载运行的情况下保持鲁棒性，实验验证则是验证控制算法的性能，因此合理的车体设计是一个至关重要的因素。

2.2.1 重载 AGV 硬件设计

2.2.1.1 机械机构及相关的机械参数

本课题所采用的是四轮独立驱动的全向移动 AGV，该 AGV 的三维模型图以及相应的实体装配图如图 2-18 和图 2-19 所示：

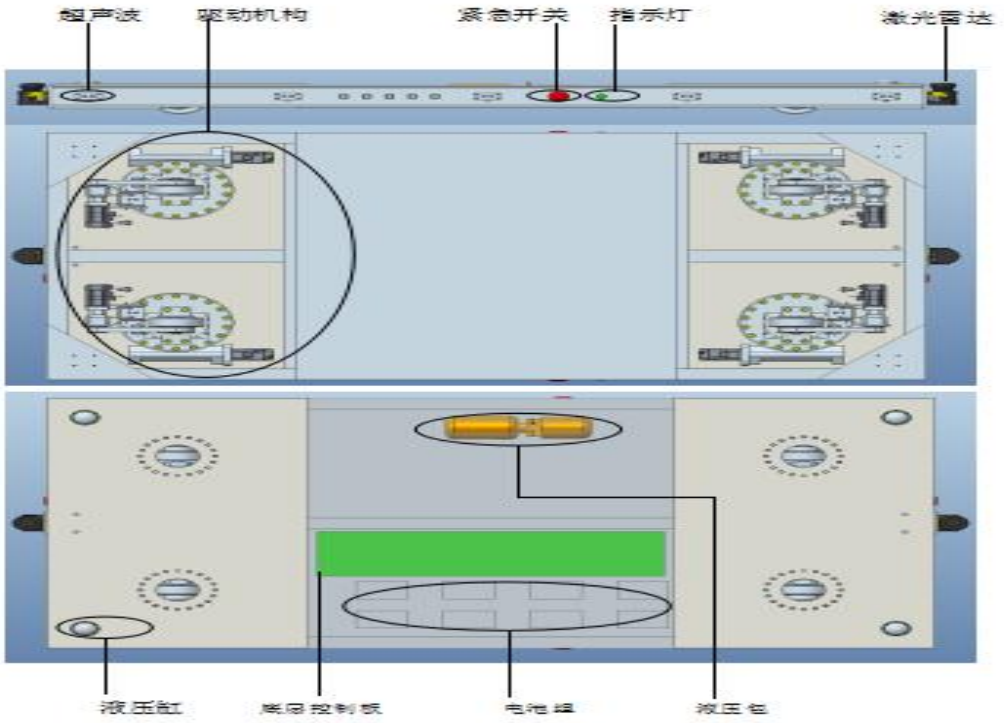


图 2-18 AGV 的三维装配图

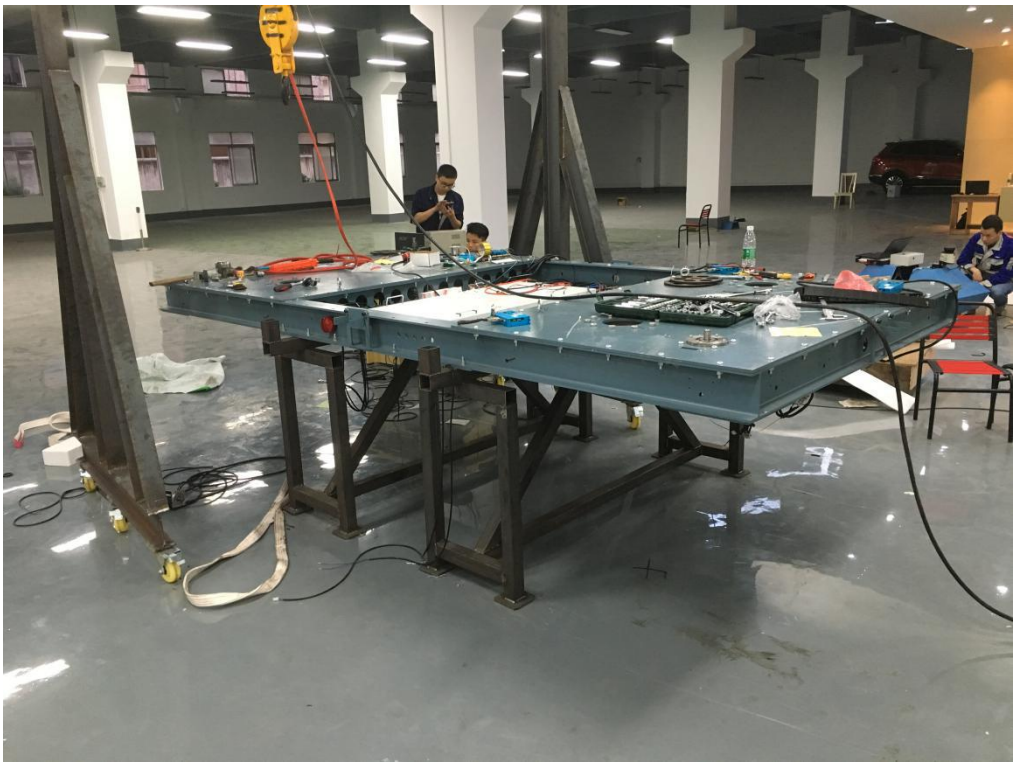


图 2-19　AGV 的三维装配图

AGV 的机械结构由车架、驱动模块、升降结构三部分组成。其中驱动模块分布在车体的四周，四个驱动模块的相互配合可以实现 AGV 的全方位运动；升降结构采用液压缸顶起需要拖拽的车体。由于车体才加工出来，很过的传感器还没有安装，后面还有许多的工作需要做。

AGV 拖车时需要配合拖拽板，通过液压缸顶起拖拽板从而实现将车拖拽起来的功能，其中 AGV 的拖拽方案和三维示意图如下图所示，AGV 的相关参数如下表所示：

表 2-4 AGV 运行的相关参数

最大速度	最大加速度	最大转速	宽	长	高	承载量
0.6m/s	0.5m/s2	10s/转	2.5m	3.5m	0.4m	2.5t

2.2.1.2 传感器及电机选型

控制系统是控制 AGV 车体的运行，是 AGV 系统的控制核心。本文拟采用三层的控制结构，分别是上位机、上位机和控制后台。其中控制后天是控制平台或者监控后台，可以实时地监控机器人运行状态；上位机是操作系统，用来发布控制命令；底层采用控制器控制直流伺服电机或者直流无刷电机，这样的控制结构提高了 AGV 车体的运动的控制精度，控制 AGV 电机的启动与停止、路径规划与导航、实时处理电机速度控制、安全防撞信息、实时通讯等；然后中位机与下位机通过串口进行通信，控制系统硬件结构组成框图如图 2-20 所示：

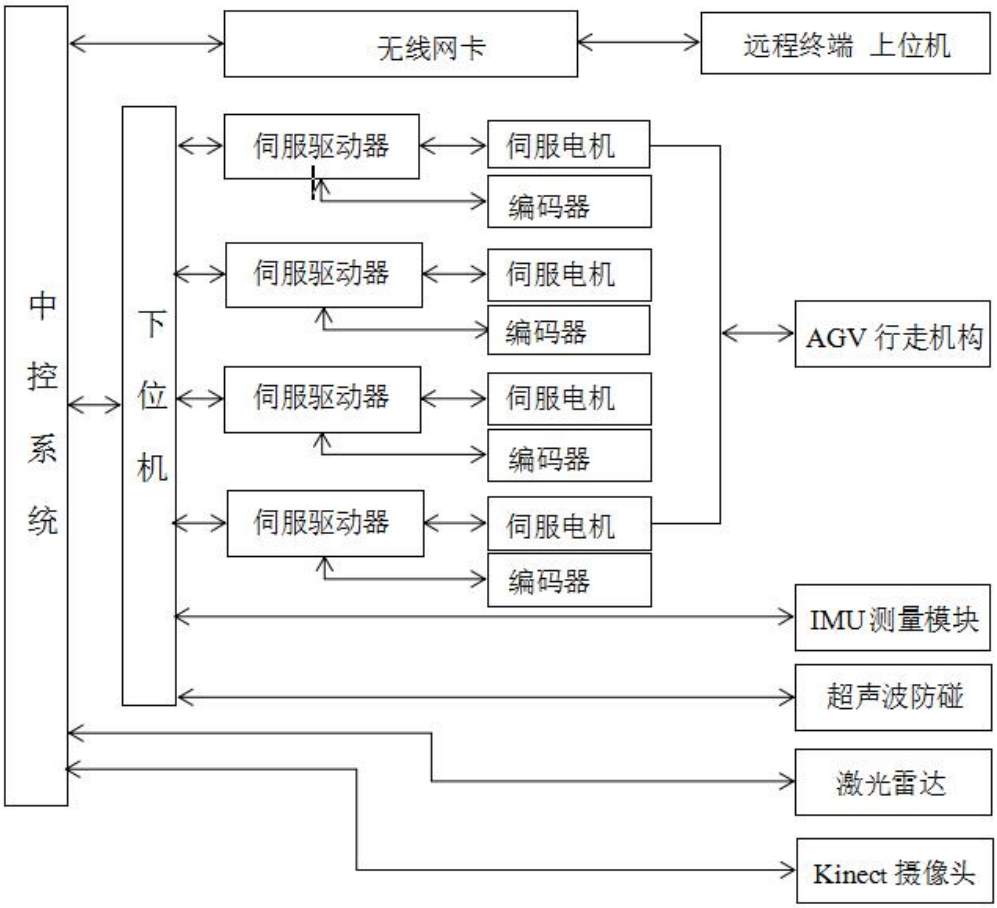


图 2-20 控制系统的框架

智能重载 AGV 在全程的运动过程中不涉及到人为参与，需要根据自身携带的传感器感知环境，并根据环境信息实现路径规划、导航和避障的功能，而且要求机器人能够很好地与其他机器人进行合作，这对控制系统的要求极高。不仅需要良好的底层控制，而且需要上层的决策设计。由于本 AGV 拟采用的里程计、IMU 和激光雷达混合的定位方式，所以需要对各种传感器数据进行融合；在避障的过程中，采用激光雷达来检测障碍物；在建图过程中基于图优化 SLAM 的方法，采用激光雷达和视觉相结合的方式，实现地图的闭环构建。由于 ROS 是一款很强大的并且发展很快的系统，通过较为简单的消息、节点处理机制，可以使得很多程序能够模块化并且可以调用，所以整个 AGV 的控制系统会在 ROS 的基础上搭建起来，基于 ROS 的控制系统的构建图如图 2-21 所示：

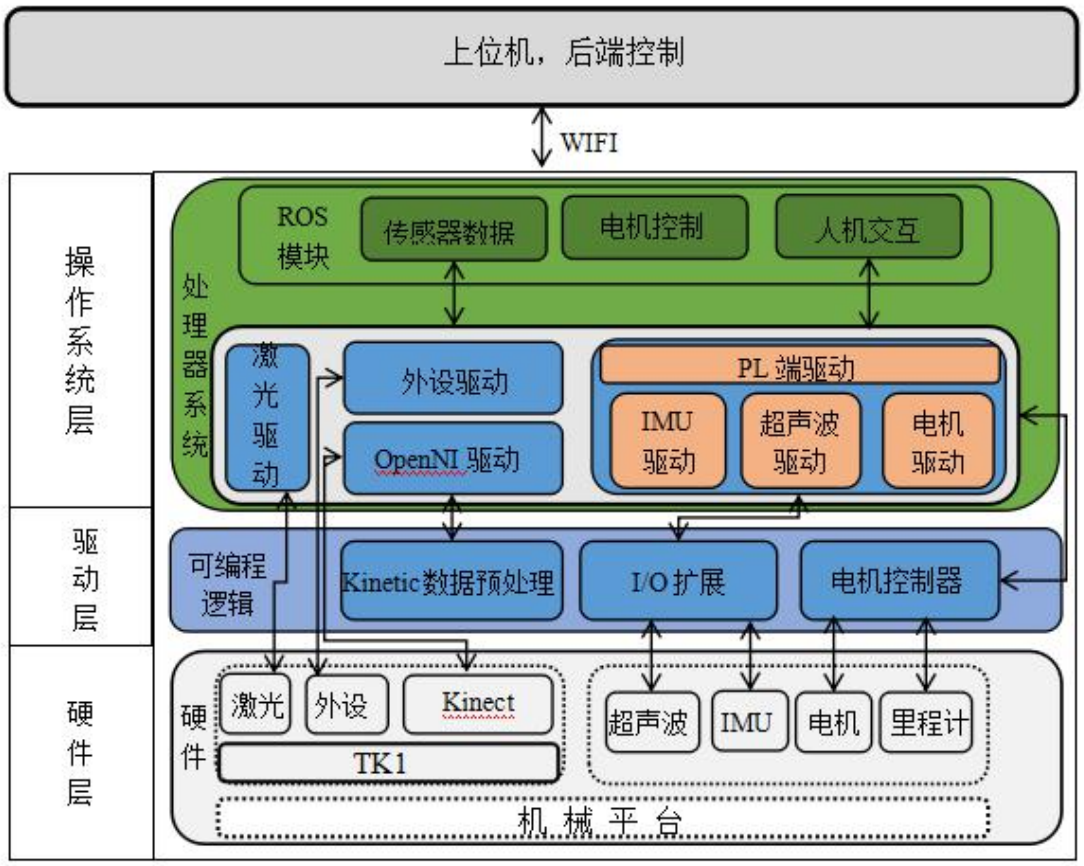


图 2-21 基于 ROS 的控制系统

智能重载 AGV 需要在未知的复杂的工作环境下面工作，需要实时的感知周围的环境；另外一方面，AGV 在运行的时候也需要实时了解本身的运行情况，方面后台的控制，所以本 AGV 采用了如下的一系列的传感器。

表 2-5 AGV 的传感器和设备

传感器或设备	型号	性能参数
激光雷达 SICK		
IMU 传感器		
超声波传感器		
Kinect 视觉传感器		
里程计		
TX1		
单片机		

2.2.2 重载 AGV 软件设计

2.2.2.1 重载 AGV 的运动校准

当 AGV 设计出来之后，必须对 AGV 的里程计进行校准，在对机器人进行校准的时候，我们需要对 AGV 的直线运动和旋转运动进行一个良好的校准，只有当机器人对线

速度和角速度进行一个好的响应并且误差不大的时候，对于后面的导航才是有意义的。在对机器人进行校准的时候，我们在实体机器人和 ROS 中的模拟机器人中都进行了实验，我们让机器人先相应线速度并且直行 20m 后，比较实际偏移了多少米；然后让 AGV 相应角速度进行两圈的旋转，并且比较与实际情况进行偏转了多少度。多次重复实验，观察数据，并且将数据记录到文件里面，然后将文件里面的数据绘制出来进行比较。

首先我们根据 ROS 中的启动文件的格式，现在 RVIZ 里面验证直线校准的正确性：

```
<?xml version="1.0"?>
<launch>
    <include file="$(find rbx1_bringup)/launch/fake_turtlebot.launch"/>
    <include file="$(find rbx1_nav)/launch/fake_move_base_blank_map.launch"/>
    <node pkg="rviz" name="rviz" type="rviz"/>
    <node pkg="agvcontrol" name="agvdebugCmd" type="odom20m.py"/>
</launch>
```

在上面的文件中，我们首先启动由 ROS 开源库里面的一个虚拟的 turtleBot 机器人，然后打开一张空白底图之后再打开 RVIZ 人机交互界面，在这个界面里面，我们让 AGV 进行 20m 的直线移动，观察机器人的移动情况；在最后的一个文件里面，我们让机器人以 0.5m/s 的速度进行 20m 的移动，实验结果如图 2-22 所示。

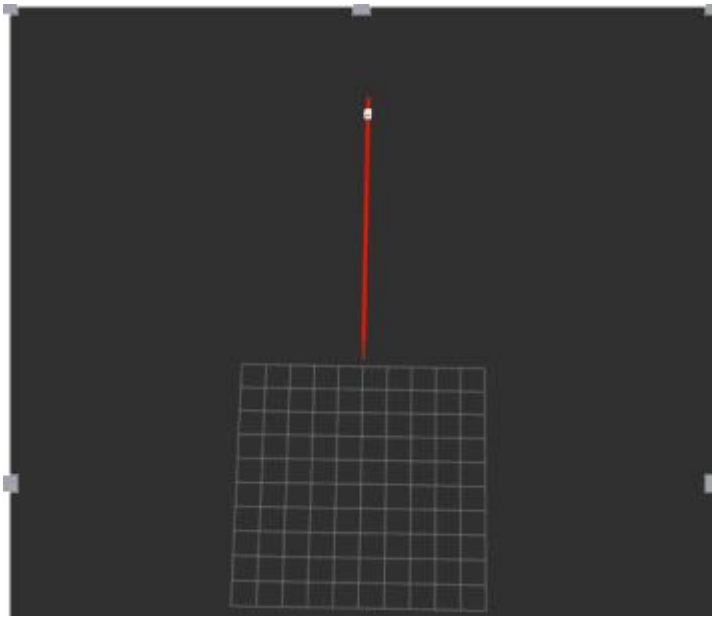


图 2-22 虚拟机器人直线校准

通过数据观察和 RVIZ 里面的观察，在虚拟机器人里面进行校准的时候基本上是没有误差的，下面通过 launch 文件，进行实体机器人上面的校准。

```
<?xml version="1.0"?>
<launch>
    <node pkg="agvcontrol" name="agvdebug" type="debugagv"/>
```

```
    <node pkg="agvcontrol" name="agvdebugPub" type="agvdebugPub"/>
    <node pkg="agvcontrol" name="agvdegbuCmd" type="odom20m.py"/>
    <node pkg="rviz" name="rviz" type="rviz"/>
</launch>
```

在这个文件中，我们首先通过串口通信，将我们上位机发布出来的速度命令根据运动学方程分解出来的各个轮的速度传递给单片机，然后单片机讲这些命令传递电机上面去；然后在第二个文件里面，我们编写了一个发布里程计的节点，并且将理论上计算的轮子的速度和实时从编码器中得出来的轮子的速度都写到一个文件里面去，方便后面的比较；在第三个文件里面，我们用 Python 语言写了一个让机器人以 0.5m/s 的线速度移动 20m 的节点；在最后的一个文件里面，在 RVIZ 里面，将根据实际编码器返回来的值绘画出一条轨迹在显示界面里面。通过观察 RVIZ 里面的机器人的走过的路径，可以看出走 20m 的时候直线偏移是在容忍范围之内的。

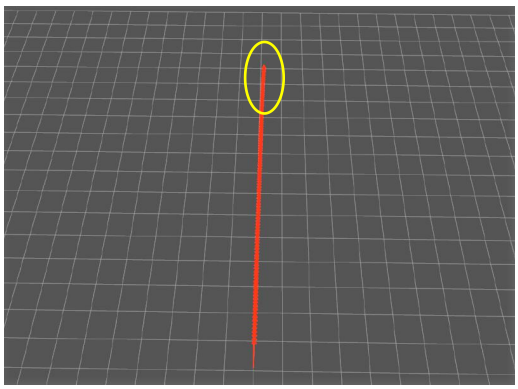


图 2-23 实际直线校准在 RVIZ 的显示

在图 2-24 中是我们实际的校准的环境以及我们实验的结果的一副场景，在图中的起始点到终点之间的距离为 20m，由于四轮独立驱动 AGV 还没有完全组装好加上在 ROS 的运动驱动还没有完全写完，这里的部分内容先用了差速驱动 AGV 进行了实验。

图 2-24 实际 AGV 的校准场景

在实际的校准的情况，我们发现偏差很大，我们将写入到文件里面的数据进行了分

析。

机器人的的校准的原理都是一样的，只要在差速 AGV 上面验证了校准的可行性，到时候就可以经过修改运动学方程就可以将这些程序移植到四轮独立驱动的重载 AGV 上面。在实验中我们的 AGV 的轮子的大小是 250mm，减速器的传动比为 35，轮距为 480mm，驱动机器人的线速度为 0.2m/s，角速度为 0rad/s，以 20m 为目标距离进行四次实验，实验中左右轮子的实际转速与理论转速如下图 2-25 所示。

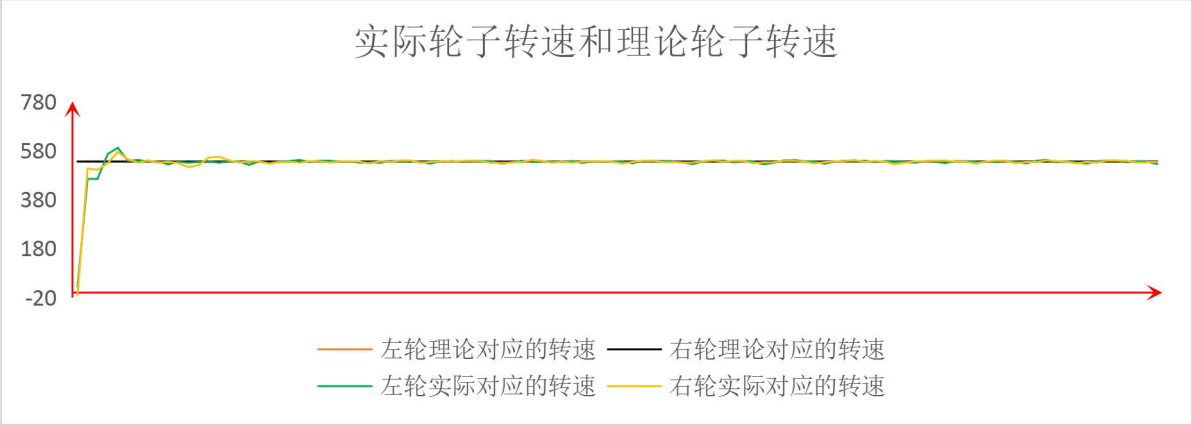
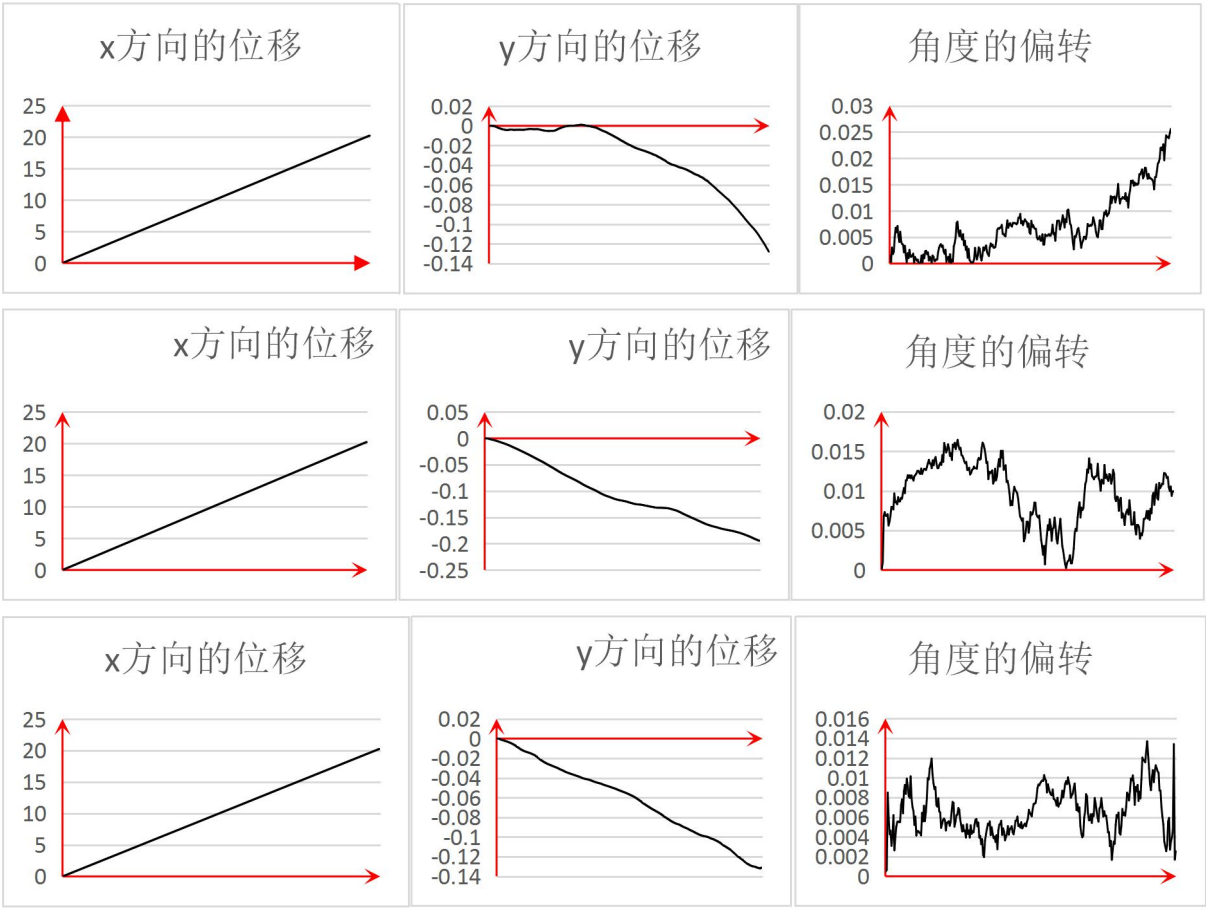


图 2-25 实验过程中左右轮的转速

在校准的过程中，我们分别对机器人 x 方向的位移，y 方向上的位移以及角度的偏移，然后将这些汇总到如下 2-26 所示的图表中



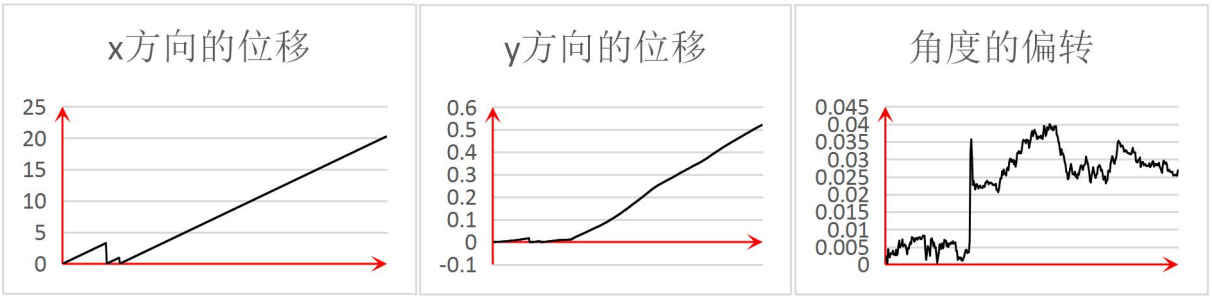


图 2-26 实验过程中的数据

通过上面的图表显示当驱动机器人行走 20m 的距离的时候，机器人在 x 轴方向基本上能够达到目标点，y 轴方向上最差的情况达到了 0.5m，但是平均值大部分在 0.2m 左右，因为现在校准的机器人里程计的准确性，这样的准确定是可以接受的。另外通过这四次实验，角度的偏差最大值 0.04rad，相当于 2° 左右，也是可以接受的。但是在实中我们测量到在 y 轴方向上偏差的平均值是 1.9m 左右，角度的偏差值达到了 18° 左右，通过分析我们发现造成这么大误差的原因是由于机械结构的原因使得我们机器人习惯性地想一侧偏转，进而造成了这么大的误差。因而有一个好的机械结构的车体也是能够实现准确导航的关键性因素之一。

由于上面的机械结构原因造成了误差很大，后面我们又加入 SLAM 构建的地图，看地图和激光能不能将车体纠正一些，实验过程如下。启动 AGV 的文件，使得 AGV 上电开始进入工作状态，然后启动激光雷达的文件使得激光处于工作的状态，最后启动 move_base 的文件，其中 move_base 的 launch 文件如下：

```
<?xml version=”1.0”?>
<launch>
    <param name=”use_sim_time” value=”false”/>
    <node pkg=”map_server” type=”map_server” name=”map_server” args=”$(find agvcontrol)/industry_map/hectortest.yaml”>
    <node pkg=”move_base” type=”move_base” respawn=”false” name=”move_base” output=”screen” clear_params=”true”/>
        <rosparam file=”$(find agvcontrol)/config/costmap_common_params.yaml” command=”load” ns=”global_costmap”/>
        <rosparam file=”$(find agvcontrol)/config/costmap_common_params.yaml” command=”load” ns=”local_costmap”/>
        <rosparam file=”$(find agvcontrol)/config/local_costmap.yaml” command=”load” />
        <rosparam file=”$(find agvcontrol)/config/globl_costmap.yaml” command=”load”/>
        <rosparam file=”$(find agvcontrol)/config/base_local_planner_params.yaml” command=”load”/>
```

<node/>

<include file="$(find agvcontrol)/launch/amcl.launch"/>

<node pkg="rviz" name="rviz" type="rviz" args="-d $(find agvcontrol)/rviz/wz_testblank_map.rviz"/>

</launch>

实验中同样给出目标点和起始点距离为 20m 的地方，然后开始让 move_base 自己规划速度让机器人移动，将机器人的每个时刻的位姿记录到文本中，最后绘出如 2-27 所示的图表：

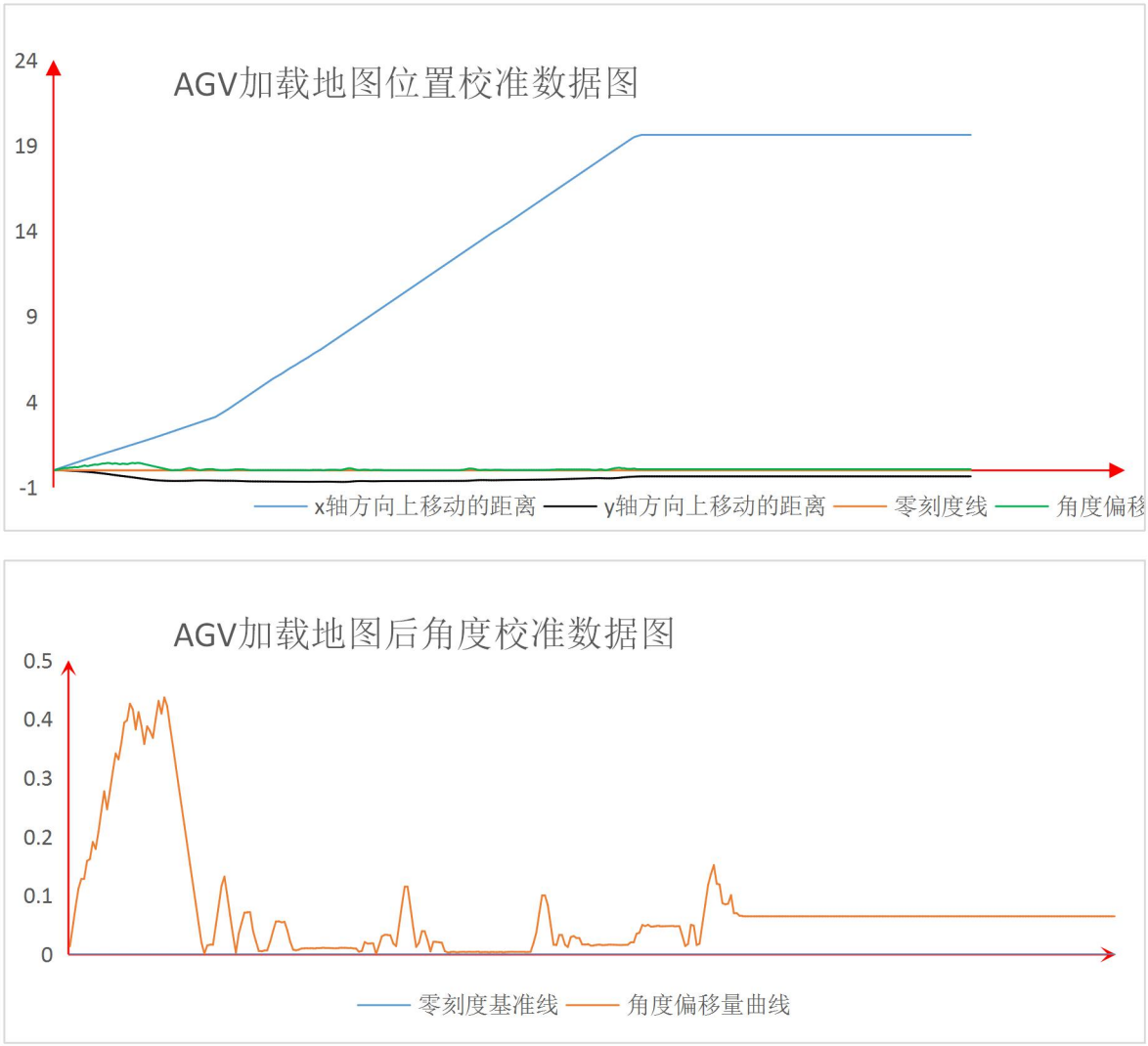


图 2-27 加载地图对 odom 数据校准的数据统计

根据上面的图表显示可以知道，机器人实际获取的数据在 x 轴方向最终能够达到 20m 左右，y 轴方向和角度误差都在一定的范围之内，并且我们可以很明显看到，加入地图后对 AGV 的位姿有较好的校准。在机器人达到目标点之后，通过测量数据可以知道实际 x 轴方向为 20.1m，y 轴方向的偏差为 1m 左右，角度为 10° 左右，与上面没有加载地图的数据相比已经有了很大的改进，这个从另外一个方面也证明了在运动过程中

move_base 中的 amcl、地图、激光确实对 AGV 的位姿进行了调整。

2.2.2.2 重载 AGV 的人机交互界面

目前人机交互界面还没有完成，另外一方面在 Ubuntu 下面还没有完成对 AGV 驱动的程序的编写，所以现在的 AGV 的人机交互界面只完成了一部分，后面的工作还有待与继续完成和、改进和测试。

目前用 Qt 编写了一份关于 AGV 运行的软件，在整个软件中我们分为了四个模块：启动项模块、机器人运行状态反馈模块、SLAM 建图模块以及导航模块。下面分别对四个模块进行简单的介绍。

（1）启动项模块

在启动项模块里面我们根据自己的需要来启动需要启动的项目，首先启动 AGV，就是打开上位机与下位机之间的通信，给每个电机上电，使得 AGV 出于即将工作的状态；在激光定位、ROS 导航、SLAM 建图的时候，需要启动激光雷达，利用激光返回的数据，进行特征提取、数据匹配等方法确定机器人自己的定位；在需要视觉和二维码相结合来定位的时候，这个时候需要启动我们的摄像头进入到工作状态，启动模块如图 2-28 所示：

图 2-28 AGV 启动模块

（2）机器人运行状态反馈模块

在状态反馈模块中，我们需要实时的监控 AGV 的运行状态，也就是需要知道机器人的定位、机器人周围的环境信息、机器人自身的状态、AGV 上面的设备的运行状态等等，图 2-29 的交互界面就可以将 AGV 在运行模式下的状态反馈回来。

图 2-29 AGV 运行状态反馈模块

（3）SLAM 建图模块

本课题的研究内容主要是将重载 AGV 应用到用 SLAM 建图的进行导航，所以在导航的时候，我们需要进行 SLAM 建图。目前三种建图方案，在这个人机交互界面我们可以根据需要来启动哪一种建图方案，并且我们可以比较任意一种建图方案实现的定位于我们航延推算算法的定位进行比较，看看哪个定位更加准确；另外一方方面，我们也可以在建图的时候选择是否显示 AGV，是否显示 AGV 运行的路径；在建图完成以后，我们可以将生成的地图进行保存，界面如图 2-30 所示。

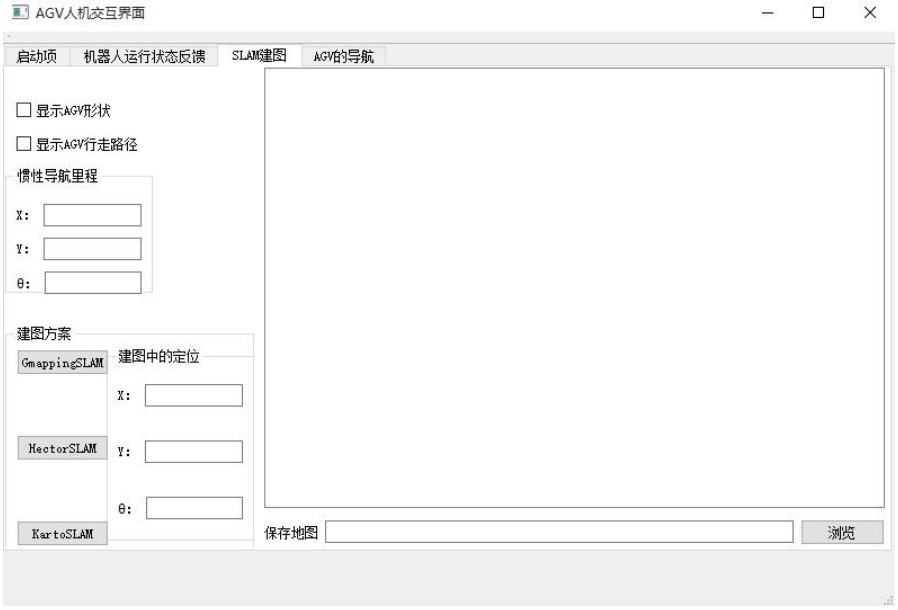


图 2-30　SLAM 建图模块

（4）AGV 的导航模块

在导航模块里面，我们需要知道我们机器人的实时运动的位置，以及机器人的起始点、终点等信息，倒入地图等等功能；另外，在 AGV 的软件设计上面，我们也加入了一

些关于视觉方面的处理，可以为后面的视觉定位提供一些帮助，其模块如图 2-31 所示。

图 2-31　AGV 导航模块

2.3 重载 AGV 的图优化 SLAM 和 ROS 导航

2.3.1 图优化 SLAM

基于图优化的 SLAM 方法，利用图模型对 SLAM 问题进行建模，模型中的节点对于与机器人及环境组成的系统在不同时刻的状态，边则描述了系统状态（节点）之间的约束关系。Gutmann 提出了图构建方法，形成了基于图优化的增量式 SLAM 算法框架，其主要包括图的构建以及图优化，并将 SLAM 问题划分为前端和后端两部分，图优化 SLAM 的核心思想如图 2-32 所示。

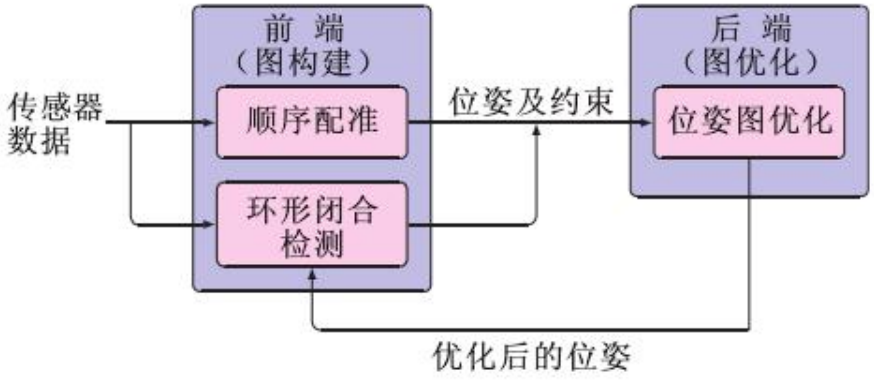


图 2-32 图优化 SLAM

图是由节点和边构成，在图优化 SLAM 中，机器人的位姿是一个节点（node）或顶点(vertex)，位姿之间的关系构成边(edge)。具体而言比如 t+1 时刻和 t 时刻之间的 odometry 关系构成边，或者由视觉计算出来的位姿转换矩阵也可以构成边。一旦图构建完成了，就要调整机器人的位姿去尽量满足这些边构成的约束。图优化 SLAM 的过程可

以描述为下面

（1）构建图，机器人位姿作为顶点，位姿之间的关系当做边，这一步就是我们常说的前端构图；

（2）优化图，调整机器人位姿顶点尽量满足边的约束，这一步成为后端优化。

图优化的过程就是，先堆积数据，将机器人位姿作为构建图的顶点；边就是位姿之间的关系，可以通过编码器来计算位姿也可以通过 ICP 匹配计算出位姿，还可以通过闭环检测来检测位姿之间的关系。

由于图图优化建图就是求最小的一个结果，所以根据下面的公式和算法，我们在 matlab 里面进行了仿真，实验结果如图 2-33 所示：

$$F(x)=\sum_{<i,j>\in C} e(x_i,y_j,z_{ij})^T \Omega e(x_i,y_j,z_{ij}) \tag{2-35}$$

$$x^*=\arg\min F(x) \tag{2-36}$$

$$e(x_i,y_j,z_{ij}) \overset{def}{=} e(x_i,x_j) \overset{def}{=} e_{ij}(x) \tag{2-37}$$

$$e_{ij}(x_i+\Delta x_i, x_j+\Delta x_j)=e_{ij}(\hat{x}+\Delta x)=e_{ij}+J_{ij}\Delta x \tag{2-38}$$

$$\begin{aligned} Fij &= e_{ij}^T(\hat{x}+\Delta x)\Omega_{ij} ei_j(\hat{x}+\Delta x) \\ &= (e_{ij}+J_{ij}\Delta x)^T \Omega_{ij}(e_{ij}+J_{ij}\Delta x) \\ &= e_{ij}\Omega_{ij}e_{ij}^T + 2e_{ij}^T\Omega_{ij}J_{ij}\Delta x + \Delta x^T J_{ij}^T\Omega_{ij}J_{ij}\Delta x \\ &= c_{ij}+2b_{ij}^T\Delta x+\Delta x^T H_{ij}\Delta x \end{aligned} \tag{2-39}$$

$$F(\hat{x}+\Delta x)=\sum F_{ij}(\hat{x}+\Delta x)=c+2b^T\Delta x+\Delta x^T H\Delta x \tag{2-40}$$

$$2b^T+2\Delta x^T H=0 \Rightarrow H\Delta x=-b \tag{2-41}$$

$$x^*=\hat{x}+\Delta x^* \qquad\qquad x_i^*=\hat{x}_i\oplus\Delta x_i^* \tag{2-42}$$

$$(H+\lambda I)\Delta x^*=-b \tag{2-43}$$

$$J_{ij}=\left[\cdots\frac{\partial e(x_i)}{x_i}\cdots\frac{\partial e(x_j)}{x_j}\cdots\right]=\left[\cdots A_{ij}\cdots B_{ij}\right] \tag{2-44}$$

$$b_{ij}=e_{ij}^T\Omega_{ij}J_{ij}=\left[\cdots e_{ij}^T\Omega_{ij}A_{ij}\cdots e_{ij}^T\Omega_{ij}B_{ij}\cdots\right] \tag{2-45}$$

$$H_{ij}=J_{ij}^T\Omega J_{ij}=\begin{bmatrix} A_{ij}^T\Omega A_{ij} & A_{ij}^T\Omega B_{ij} \\ B_{ij}^T\Omega A_{ij} & B_{ij}^T\Omega B_{ij} \end{bmatrix} \tag{2-46}$$

$$e_{ij}(x_i,x_j)=Z_{ij}^{-1}(X_i^{-1}X_j)=Z_{ij}^{-1}\begin{pmatrix}R_i^T(t_j-t_i)\\ \theta_j-\theta_i\end{pmatrix} \tag{2-47}$$

$$\Delta t_{ij}=R_z^T\left[R_i^T\left(\begin{pmatrix}x_j\\ y_j\end{pmatrix}-\begin{pmatrix}x_i\\ y_i\end{pmatrix}\right)-\begin{pmatrix}x_z\\ y_z\end{pmatrix}\right] \tag{2-48}$$

$$\Delta\theta=\theta_j-\theta_i-\theta_z \tag{2-49}$$

$$A_{ij}=\begin{bmatrix}-R_z^TR_i^T & R_z^T\dfrac{\partial R_i^T}{\partial x_i}(t_j-t_i)\\ 0\quad 0 & -1\end{bmatrix} \tag{2-50}$$

$$B_{ij}=\begin{bmatrix}R_z^TR_i^T & 0\\ 0\quad 0 & 1\end{bmatrix} \tag{2-51}$$

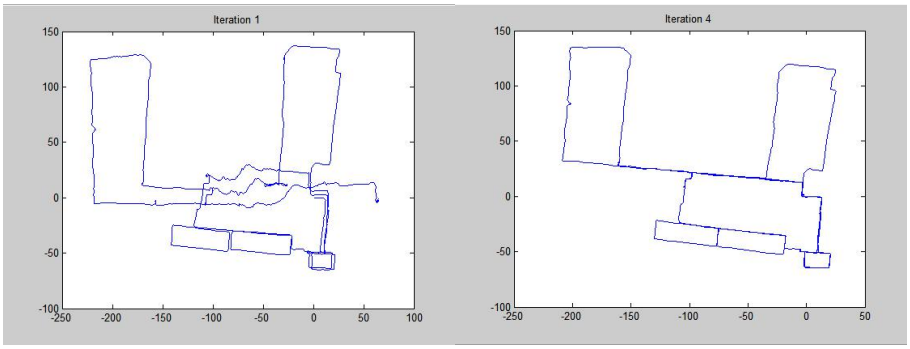


图 2-33 图优化 SLAM 在 matlab 仿真图

从上面在 matlab 里面的仿真结果我们可以看出，在数据堆积的时候也就是前端的时候我们建立的地图有很多重叠的地方，如果用这种地图进行导航很显然不能满足我们的导航的要求；在后端优化的时候，我们可以看到经过多次的优化和迭代之后，能够形成一个较好的清晰的地图。

在 matlab 里面验证了图优化算法的正确性之后，我们在 Ubuntu 里面 ROS 的方法，开始进行了建立栅格地图的实验：

实验场景分别是工厂和公司的走廊环境，环境如图 2-34 和 2-35 所示：

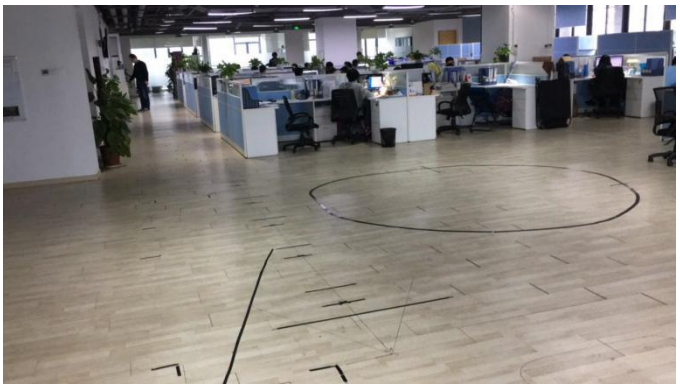


图 2-34 建立地图试验场地 1

图 2-35 建立地图试验场地 2

所建立的地图的效果如图 2-36 所示：

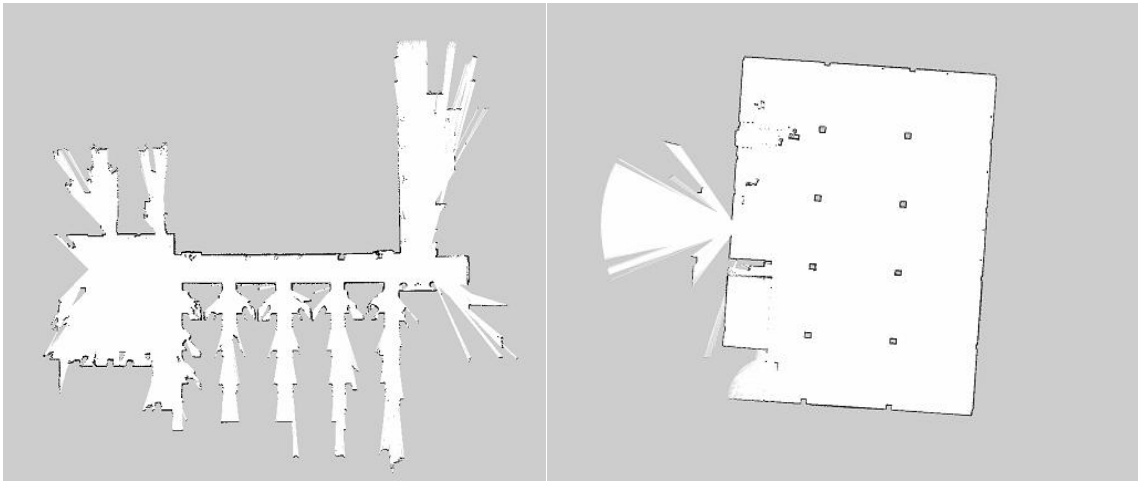


图 2-36 kartoslam 建立的地图

在建图的时候，我们选择了 kartoslam 的建图方案，选择该方案有两个原因：一方面就是 kartoslam 是基于图优化方法，中间加入了闭环检测的环节，建立的地图有着比较好的闭环效果；另外一方面就是 kartoslam 的方法的运算速度是最快的，最 CPU 的占用资源最少，这种对 CPU 占用少的特点对于我们后面边建图边导航带来的便捷。

下面是在工厂环境下对 gmapping 也就是粒子滤波建图方案和 kartoslam 建图方案对于闭环的比较，如图 2-37 和图 2-38 所示：

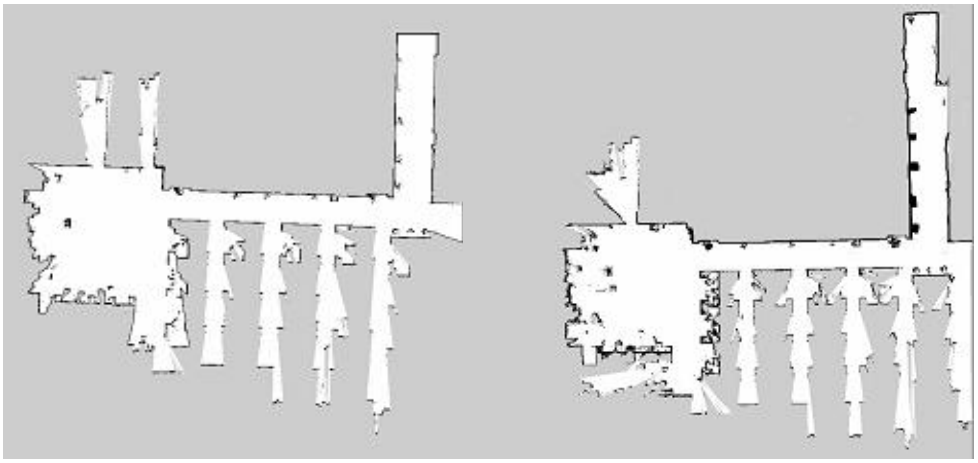


图 2-37 kartoslam 和 gmapping 在办公室环境建立的地图

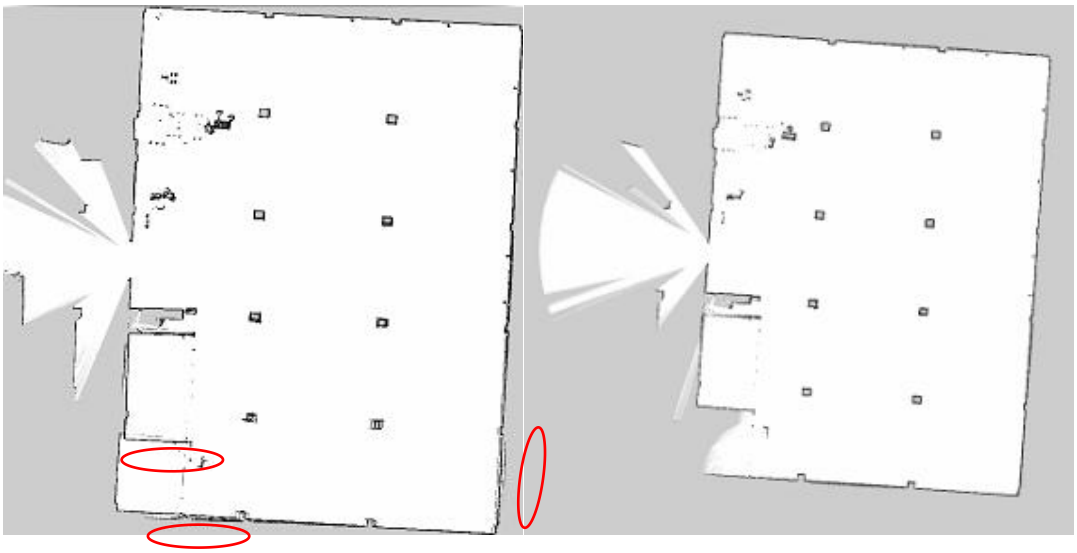
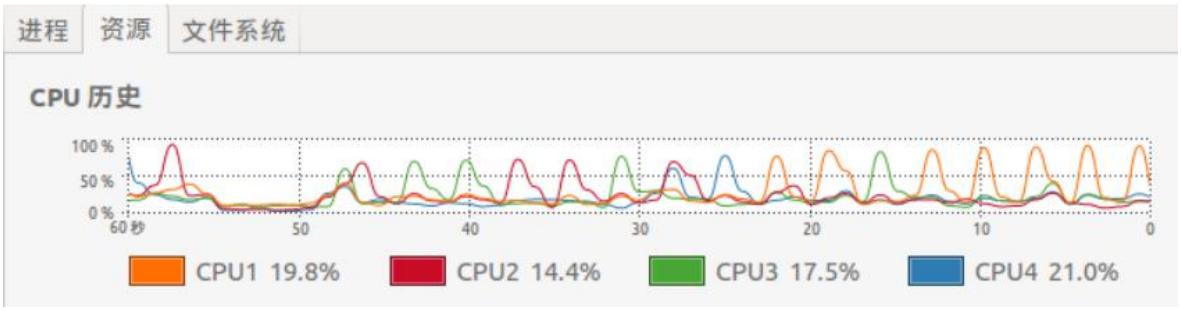


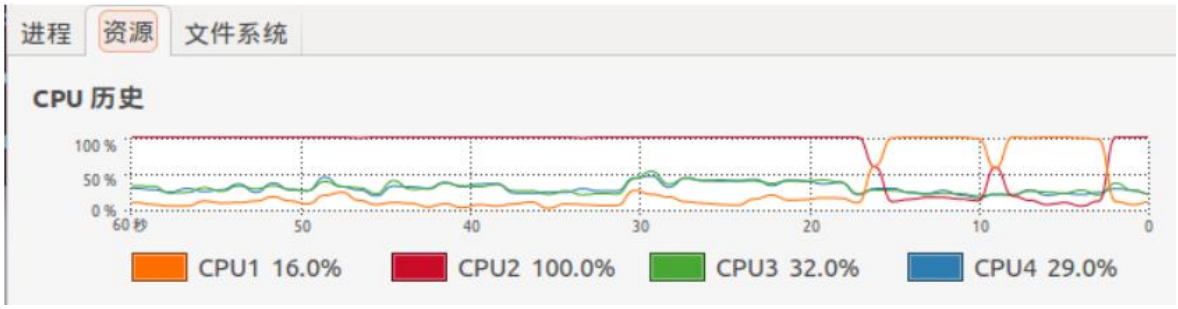
图 2-38 kartoslam 和 gmapping 在工厂大环境建立的地图

分析比较上面建图的两种方案，在大环境下面，粒子滤波的方案中间的闭环效果并不是很好，而且如果机器人不能很好地定位，粒子滤波的方法容易出现粒子枯竭和重采样的过程，这不是我们想看到；而基于图优化的方法建立起来的地图闭环效果相对来说就好一些，用在导航的效果中就会更好一些。

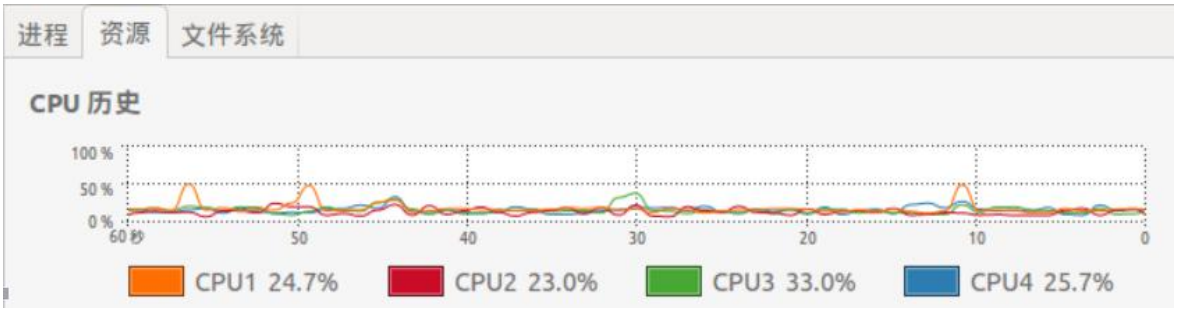
上面说到，三种建图方案中基于图优化的 kartoslam 占用 CPU 资源是最少的，所以在建图的时候，我们对这三种建图方案建图的时候对 CPU 的消耗进行了监控，实验获得对 CPU 消耗如图 2-39 所示：



图（a） hectorSLAM 对 CPU 消耗示意图



图（b） GMappingSLAM 对 CPU 消耗示意图



图（c） KartoSLAM 对 CPU 消耗示意图

图 2-39 不同建图方案对 CPU 消耗

在上面的图中分别对应的是 gmapping、hectorslam 和 kartoslam 对应的对 CPU 的消耗，通过比较我们确实发现 kartoslam 对 CPU 的消耗是最小的，这样的结果对于我们后面机器人在未知环境的探索实现导航非常有意义。

2.3.2 特征提取算法以及实验结果

激光雷达具有能同时精确测量距离和方位角的能力并且受环境影响程度小，故而被广泛地应用于各种机器人导航于定位中。但是激光雷达数据仍然存在一些问题，这些问题给特征提取方法的可靠性、精度和重复度带来了挑战。激光雷达数据主要存在以下几个问题：

a. 噪声 虽然激光数据精度非常高，但是数据不可避免地会受到噪声的影响，这些噪声会表现在光滑平面的激光雷达观测存在锯齿状结果，进而会产生大量不可重复的虚假特征点；

b. 离散化误差 由于激光雷达将环境数据高度离散化，因此其离散化误差对特征提取具有较大的影响，特别是目标物体距离传感器距离远时，这种离散化误差会导致同样的物体返回差异非常大的观测，进而影响特征提取的重复度；

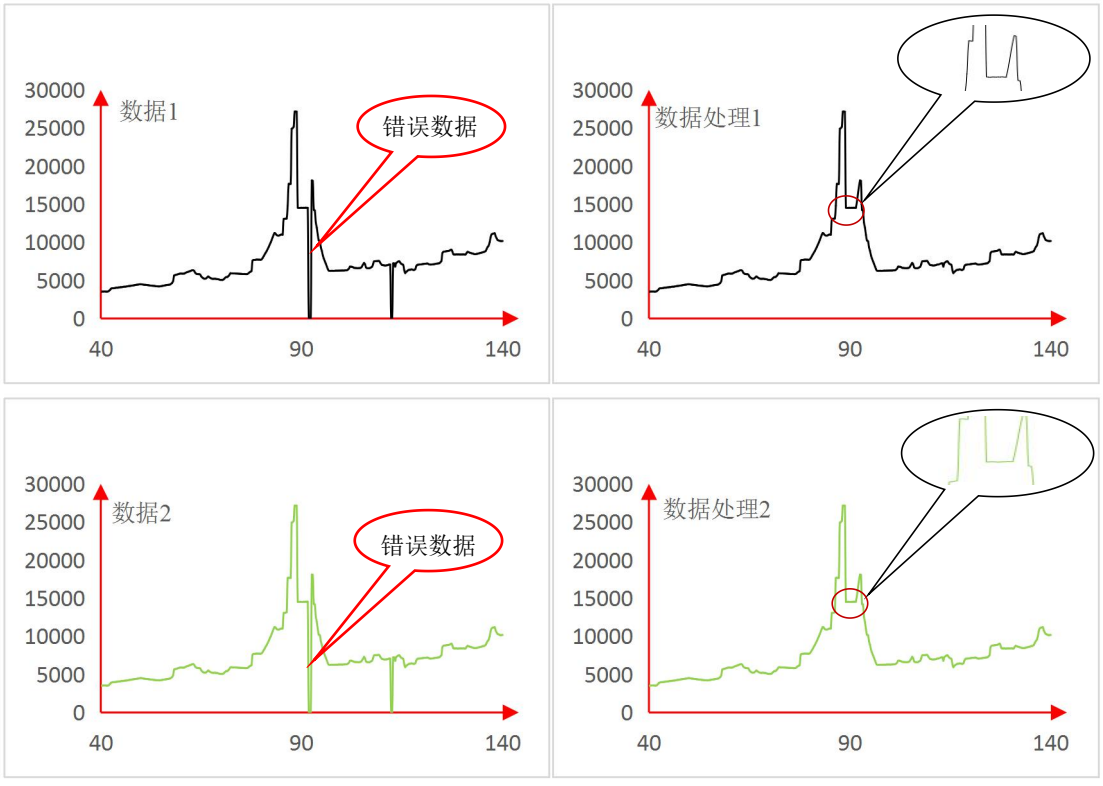
c. 数据缺失 由于存在遮挡和高反射性物体，激光雷达数据存在较强的非连续性，从而影响对数据的正确处理，导致提取特征的重复度降低。

本课题采用的德国生产的 SICK151 激光雷达，其分辨率为 0.25°，可以测得有效距离为 20m，其频率为 50Hz 或者 25Hz 图片如图 2-40 所示：

图 2-40 实验所用激光雷达

在用 ROS 下做完激光雷达的驱动之后，分别测取了距离激光雷达 15 米和 20 米的数据，每组实验做了五组并对激光的数据记录。实验环境以及实验数据得出的图表如下所示。

图 2-41 激光特征提取实验场景图



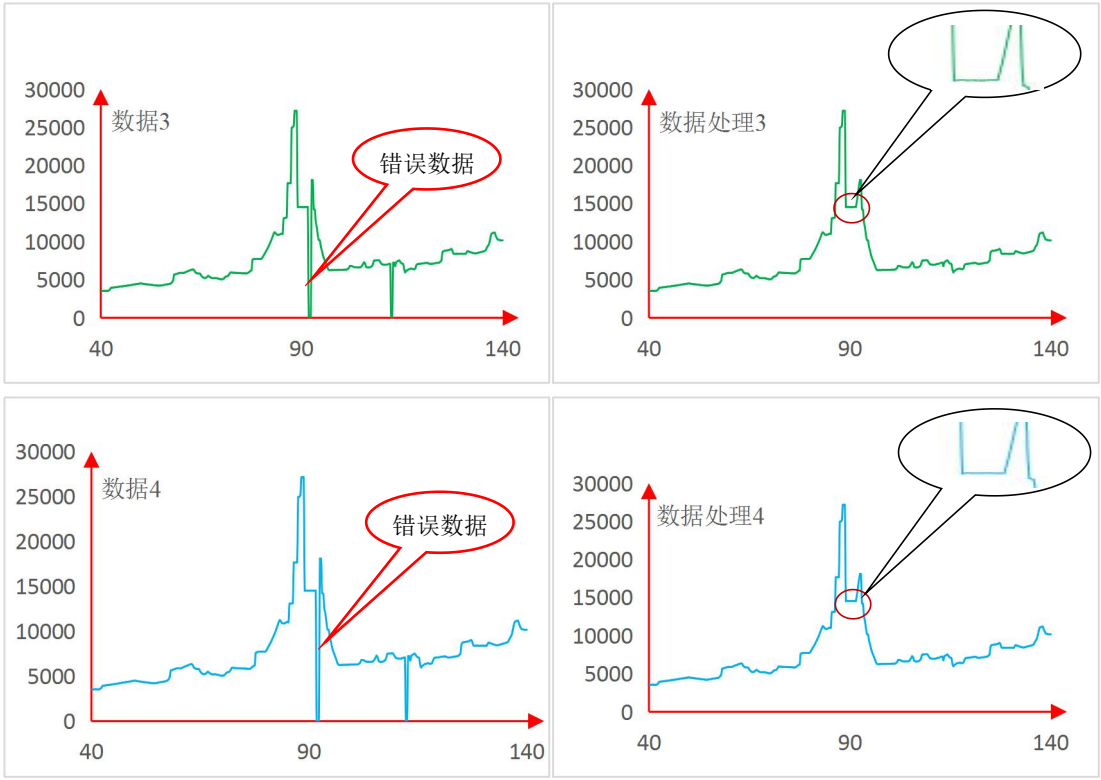


图 2-42 15m 激光特征提取数据图

上面测得是目标物体距离激光中心大约 15m 的数据，其中在图的左侧出现了很多跳跃的数据，那是由于没有数据反射回来的原因，右侧的图中在对错误数据进行过滤之后得到的曲线，其中放大图就是我们需要提取的特征，并且通过观察曲线图，我们需要的特征大约就落在了 15m 左右，这是与实际情况相符合的。

下图给出另外 20m 的实验数据处理后的结果图如图 2-43 所示。

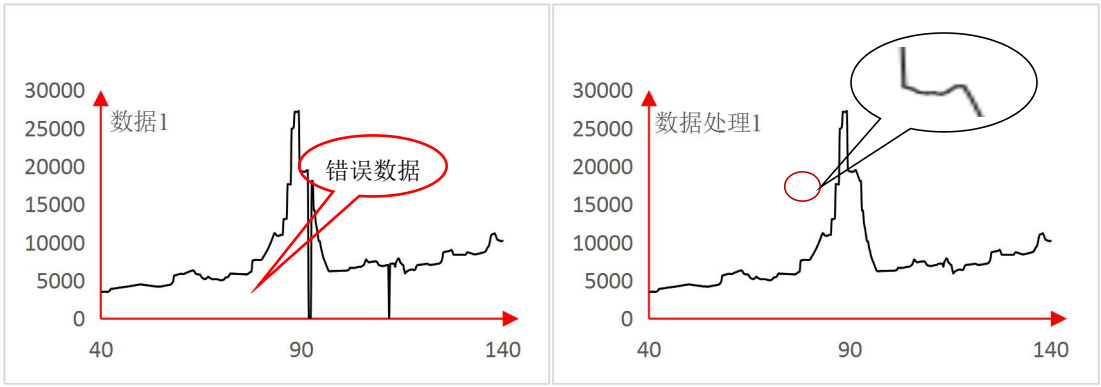


图 2-43 20m 激光特征提取数据图

在实验环境中有很多的直线、角点、圆弧等，这些都可以作为待提取的特征。三维环境中的墙壁落在平面上后反映出来的就是一条直线，而激光雷达在扫描的时候会有很多的激光点打在墙壁上，只要找到这些数据点就可以找到墙壁。

在用激光进行特征提取的时候，我们需要做下面几件事情：

1.预处理 因为激光返回的数据中会有一些错误的数据，这些数据需要进行预处理剔除掉；另一方面就是需要对激光雷达数据进行聚类，聚类的根据就是：从相同的物体返

回的激光雷达数据往往具有相同的特征。

2.提取特征 利用相关的算法，提取环境中的直线、角点和圆弧这些特征。

一条在 2D 平面坐标系中仅用 ρ 和 θ 两个参数描述即可，其中 ρ 是指机器人到直线 L 的垂直距离， θ 是垂线与正 x 轴的夹角，其描述如图图 2-44 所示：

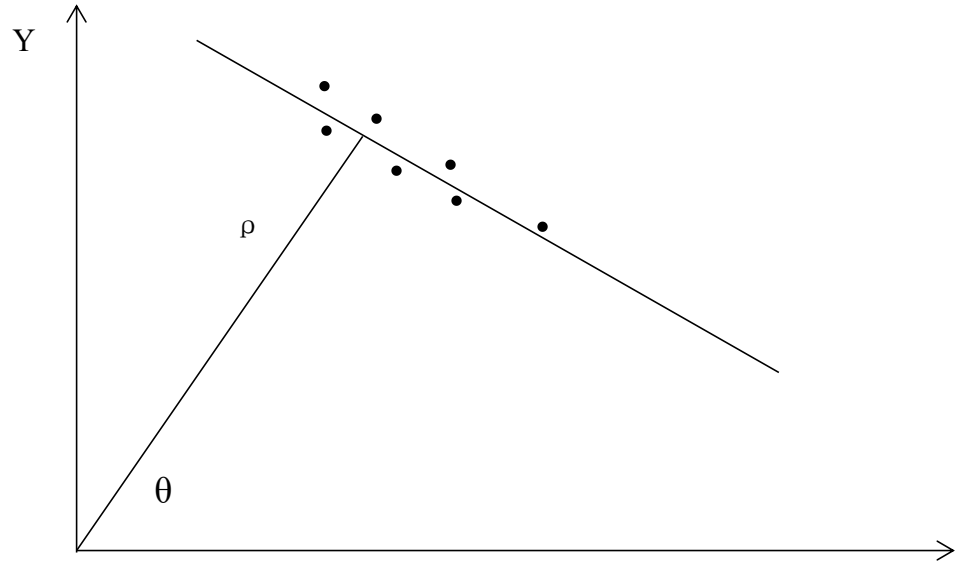


图 2-44 2D 平面直线的表示方法

通过公式 2-2 我们可以利用最小二乘法进行直线你和成为下面的形式：

$$y = k * x + b \tag{2-52}$$

$$\begin{cases} k = \dfrac{n*\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n*\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i * \sum_{i=1}^{n} x_i} \\ b = (\sum_{i=1}^{n} y_i - k * \sum_{i=1}^{n} x_i)/n \end{cases} \tag{2-53}$$

在算法中选择进行一次直线拟合的天数为 7，即为点集[x_i-3,x_i+3]。对激光雷达传递回来的数据进行拟合可以拟合出 1080 个数据点可以拟合出 1079 条直线，每一条直线 L_i 都可以用 ρ 和 θ 两个参数进行表示，然后采用角度直方图的方法对每条直线进行投票。同一个墙面上的点拟合而成的直线必然具有相同或相近的 ρ 和 θ 值，在角度直方图中获得票数最多的直线就可以找到对应的墙壁所在，并可以推算出墙壁的长度、与机器人的距离和墙壁的倾斜角等几何信息。在实际的情况中，可能遇到墙面凹凸不平的现象，必须设置一个阈值作为过滤器，如果 ρ 和 θ 小于这个阈值就可以认为是同一条直线。

由于环境中直线已经提取出来，而两条直线的交点就是我们所需要的角点。但是在实际情况中，根据上面的算法提取出来的直线并不一定具有相同的顶点，而且两条直线

也无法保证完全垂直。假设(x1,y1,xc1,yc1)和(x2,y2,xc2,yc2)是形成相互垂直的两条直线，（xc1,yc1）和（xc2,yc2）分别是处于拐点区域的坐标点。

在实际应用中拐点的判断准则如下：

$$\begin{cases} \left| \left\| \arctan\left[\frac{Y_1 - X_1}{X_1 - X_{c1}} \right] - \arctan\left[\frac{Y_2 - X_2}{X_2 - X_{c2}} \right] \right\| - 90 \right| < e_s \\ \sqrt{(Xc1 - Xc2)^2 + (Yc1 - Yc2)^2} < e_d \end{cases} \tag{2-54}$$

其中 e_d 和 e_s 是根据经验值来获取的，在实际中，两个值不是很大。所以我们可以近似认为角点的坐标为：

$$\begin{cases} X = \frac{X_{c1} + X_{c2}}{2} \\ Y = \frac{Y_{c1} + Y_{c2}}{2} \end{cases} \tag{2-55}$$

对于凸起障碍物的提取，比如我们认为放在环境中的路标。由于打在路标和非路标上的激光数据会发生很大的跳跃，所以在处理数据时候我们会找到这种数据跳跃很大的区域，只要满足了这个条件的数据，将这个区域的数据存储起来；然后将存储区域里面的数据进行直线拟合，如果能够拟合出一条直线，我们就认为这是一条直线；在得到直线以及直线的两个端点后，我们就可以知道障碍物的长度，然后用这个求得的长度与实际测量的路标的长度进行比较，如果两者之间的误差满足我们设置的阈值条件，我们就认为我们找到了我们放在环境的路标。

2.3.3 基于 SLAM 地图的导航

导航系统主要由地图、自定位模块和多层递阶规划模块三部分组成导航的系统，其中地图由图优化 SLAM 构建闭环的地图，自定位模块利用已知地图结合里程计信息和激光传感器的混合信息进行定位，其方案均在上面有所描述。后面着重介绍多层规划模块。

在前面的已经建立的准确地地图信息上，对地图进行膨胀处理，然后在膨胀的全局静态地图上用基于改进的 A*算法规划处一条全局的路径，然后再在局部的代价地图上规划处局部路径用来躲避障碍物。当在规划的全局路径上出现障碍物的时候，机器人根据局部路径规划出来的路径绕开障碍物，当局部没有障碍物的时候，让机器人紧贴着全局规划出来的路径行走。多层次递阶规划的方案如下图所示：

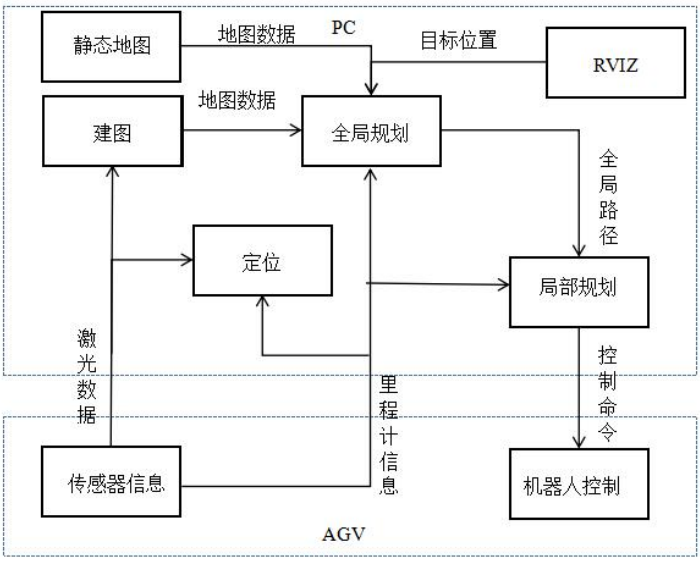


图 2-45 多层次规划框图

AGV 负责数据采集和运动控制。

传感器信息：采集所有的传感器数据，包括激光数据、IMU 数据、里程计的速度和距离信息以及 Kinect 信息，并将这些数据打包发布出来；

控制命令：ROS 中的导航核心 move_base 根据局部规划出来的路径并将路径上面的打分最高的速度采样值（角速度和线速度）发布出来，控制机器人快速、准确地根据指令移动；

PC 端主要负责大量运算量的地图构建、定位和导航。

建图：根据激光数据进行 SLAM 地图构建，并且发布地图数据；

定位：接受 AGV 返回的 IMU 数据、里程计数据，并将其与激光数据融合，负责机器人定位并发布机器人位置信息；

全局规划：根据地图数据、机器人位置信息、目标位置以及里程计的数据计算出最优的全局路径，并将路径的话题发布出来；

局部规划：根据机器人里程计的速度和角度信息，同时计算每个控制周期的最优速度和角度。

当机器人执行导航任务时，由于在线规划路径的实时性特征，局部规划层对规划算法的执行效率要求很高。鉴于局部规划所覆盖的地图范围较小，考虑用栅格地图表示局部环境。由于环境信息的不确定性，为保证导航任务的顺利执行，在从地图库中获取局部地图作为当前局部规划的参照基础上，导航系统通过激光测距仪不断采样环境对局部地图进行更新，确保局部地图的准确性。其中递阶规划算法流程图如下：

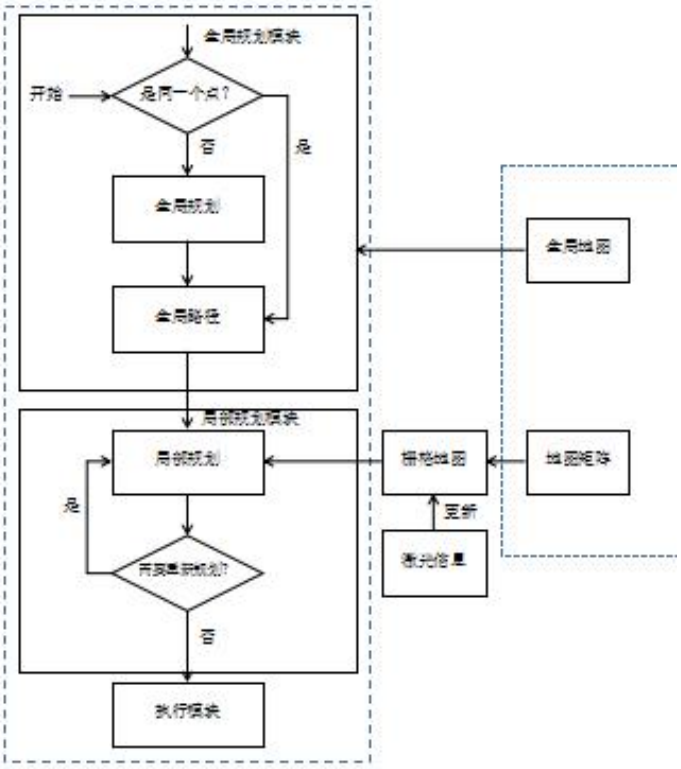


图 2-46　AGV 基于 ROS 的多层次递阶规划流程

基于 ROS 导航的核心 move_base 框架如下所示：

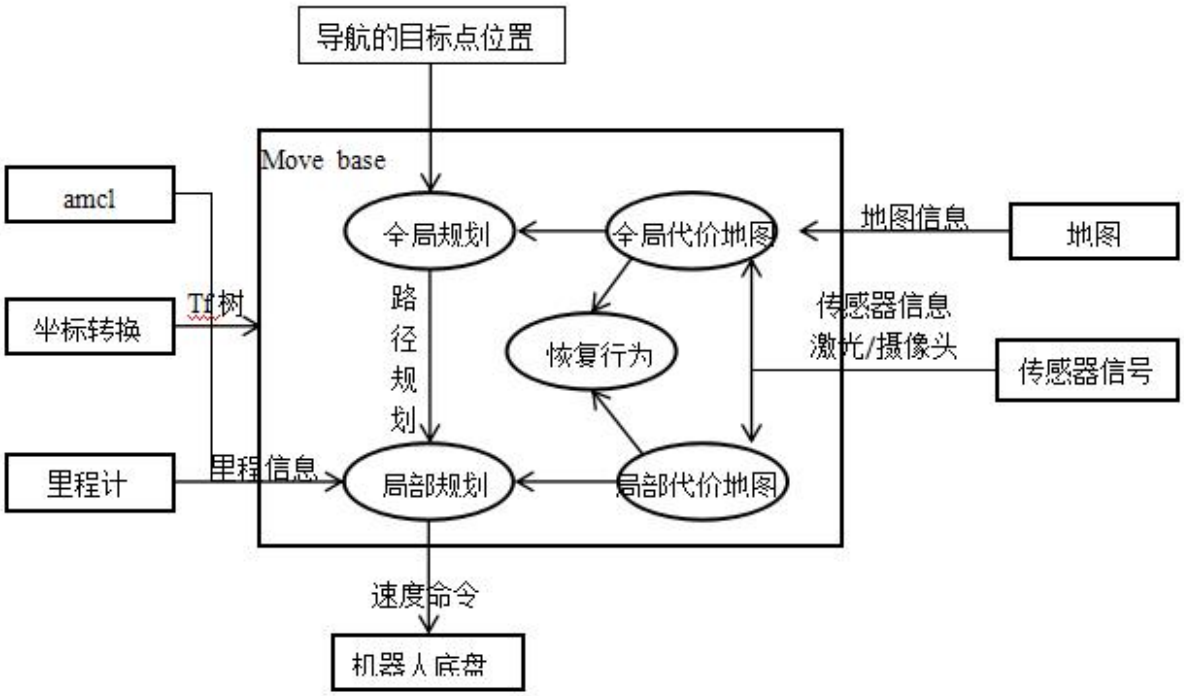


图 2-47　AGV 基于 ROS 的流程图

3.后期拟完成的研究工作及进度安排

时间	课题进展与预期目标
2017.03.01——2017.04.01	利用 Qt 和 ROS 相结合，设计一款人机交互界面
2017.04.02——2017.05.31	将实验期间取得的相关成果进行整理，并发表一篇学术小论文
2016.06.01——2016.07.31	整理研究成果，撰写、修改、完善硕士学位论文
2016.08.01——2016.12.20	准备硕士学位论文答辩

4. 研究过程中遇到的困难和技术问题

由于 AGV 的算法方面涉及到的数学知识比较多，需要在数学方面进行部分的不足；另外一方面，对于用 Qt 开发人机交互界面没有什么经验，需要大量的时间来进行学习；还有一方面，目前 AGV 在用传统的激光三角定位算法进行导航的时候，精度可以达到 5mm，但是用 SLAM 的精度目前还打不到要求，还需要对定位算法进行改进，如果需要加入其它的例如像二维码来进行辅助定位的话，又必须考虑到光照条件的影响，所以需要拿出一套比较好的在 ROS 下的定位方案；在用 ROS 的时候，由于 ROS 的框架是一个分布式通信的框架，因而造成了一些不同步的现象，这个问题还没有得到解决。