第十四届“恩智浦”杯全国大学生
智能汽车竞赛
室外光电创意组

技 术 报 告

参赛学校：中南大学

队伍名称：比亚迪丑牛

参赛队员：孔维航

范金泽

孙飘宇

带队教师：王击

关于技术报告和研究论文使用授权的说明

本人完全了解第十四届“恩智浦”杯全国大学生智能汽车竞赛关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会和恩智浦公司可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：

带队教师签名：

日　　期：

前言

本项目以全国大学生恩智浦杯智能车大赛为背景,将以 ARM Cortex-M4 内核的 K66 单片机为主控模块，选择合适的元器件作为主控芯片的外围电路，完成硬件系统的制作。我们利用激光雷达、IMU、底盘、i3 处理器构建智能车系统，在特定道路的基础上，加强对非特定环境（如临时障碍）的识别和自主处理能力（如自主调速、躲避），加深对控制理论的研究。目前，我们已经实现基本的建图、规划、导航、避障，速度 3m/s。

Leonard 和 Durrant-Whyte 提出智能机器人自主移动需要解决的三个问题：“我在哪儿？”、“我周围的环境如何？”以及“我该怎样到达目的地？”指出了机器人需要解决自我定位，周围环境描述和路径规划问题才能做到自主移动。对于定位问题，1988 年 Smith、Self 和 Cheeseman 最早将 SLAM 问题定义为滤波问题，通过卡尔曼滤波方法对 SLAM 进行优化并得到广泛应用。2000 年 Doucet、Freitas 和 Murphy 提出了 Rao-Blackwellized 粒子滤波方法，使用机器人的位姿估计进行环境地图的构建。针对视觉传感器计算量大的问题，Kummerle、Grisetti 和 Strasdat 等研究人员提出了图优化的方法。2016 年，谷歌开源了基于图优化的 cartogrpher 算法，使得基于激光 SLAM 的机器人更易产品化。而路径规划回答了“我该怎样到达目的地？”的问题。最早在 1968 年研究人员提出了基于栅格法的路径规划方法，通过将地图划分为多个栅格，简化地图中的环境特征用于规划移动路径。1959 年荷兰计算机学家狄克斯特拉提出了 Dijkstra（狄克斯特拉）算法，成为最基础的最短路径搜索算法。随后，Floyed 算法、SPFA 算法(Bellman_Ford 改进算法)也应运而生。

本篇技术报告将从硬件、软件两方面阐述移动机器人的自主避障和导航。硬件方面，我们阐述了元件的选型及原理图。软件方面，我们将简单描述建图原理，对比 gmapping 和 cartographer 的实际应用，选择合理建图方案，再详细阐述使用 EKF 实现激光里程计和 IMU 的数据融合，最后，我们将简要介绍基于 Dijkstra 和 TEB 的路径规划 pure-persuit 的转向控制器。

目录

第一章　绪论

1.1 选题背景及意义

本项目以全国大学生恩智浦杯智能车大赛为背景,将以 ARM Cortex-M4 内核的 K66 单片机为主控模块，选择合适的元器件作为主控芯片的外围电路，完成硬件系统的制作。本项目将利用激光雷达、IMU、底盘、i3 处理器构建智能车系统，在特定道路的基础上，加强对非特定环境（如临时障碍）的识别和自主处理能力（如自主调速、躲避），加深对控制理论的研究。

自第一台工业机器人诞生以来，机器人的民展已经遍及机械、电子、冶金、交通、宇航、国防等领域。近年来机器人的智能水平不断提高，并且迅速地改变着人们的生活方式。人们在不断探讨、改造、认识自然的过程中，制造能替代人工作的机器一直是人类的梦想。 其中智能车可以作为机器人的典型代表。其需要实现自动导引功能和避障功能就必须要感知导引线和障碍物，实现自动识别路线，选择正确的行进路线，使用传感器感知路线并作出判断和相应的执行动作。智能小车设计与开发涉及控制、模式识别、传感技术、汽车电子、电气、计算机、机械等多个学科。现智能小车发展很快，从智能玩具到各行业都有实质成果，其基本可实现循迹、避障、检测贴片、寻光入库、避崖等基本功能，有向声控系统发展的趋势。比较出名的“恩智浦杯”智能车大赛更是走在前列，于今年首次引入利用激光雷达自主避障导航的创意组。我们此次的设计主要实现自主避障导航这一个功能， 开展基于激光 SLAM 的自主避障导航这一功能的研究工作，本次研究为机电一体化专业学科的学生学习和掌握机电一体化技术有很大的帮助，对机电一体化专业学科的学生进一步巩固已学知识加深已学知识起到促进作用，引导和激励学习。

1.2 国内外研究背景

SLAM 是同时定位与地图构建，是指搭载特定传感器的主体，在没有环境先验信息的情况下，于运动过程中建立环境的模型，同时估计自己的运动。

1988 年 Smith、Self 和 Cheeseman 最早将 SLAM 问题定义为滤波问题，通过卡尔曼滤波方法对 SLAM 进行优化并得到广泛应用。2000 年 Doucet、Freitas 和 Murphy 提出了 Rao-Blackwellized 粒子滤波方法，使用机器人的位姿估计进行环境地图的构建。针对视觉传感器计算量大的问题，Kummerle、Grisetti 和 Strasdat 等研究人员提出了图优化的方法，通过特征提取得到每一帧的图像并利用帧间完成 SLAM 的定位。南京航空航天大学的满增光博士结合 EKF（卡尔曼滤波）算法对编码器进行校正确认机器人位姿，再利用下一时刻的位姿与此时刻的位姿进行卡尔曼滤波预测，校正地图与机器人位姿。中国科学技术大学的陈赢峰博士基于四叉树的 SLAM 方法，降低了 50%到 70%的内存消耗，并通过里程计标定改进了环境自感知定位算法。2016 年，谷歌开源了基于图优化的 cartogrpher 算法，使得基于激光 SLAM 的机器人更易产品化。

Leonard 和 Durrant-Whyte 提出智能机器人自主移动需要解决的三个问题：“我在哪儿？”、“我周围的环境如何？”以及“我该怎样到达目的地？”指出了机器人需要解决自我定位，周围环境描述和路径规划问题才能做到自主移动。路径规划出的路径应该是轨迹最短并最为安全的路线。常用路径规划算法主要有人工势场法、Dijkstra 算法、Floyed 算法、SPFA 算法(Bellman_Ford 改进算法)、A*算法、D*算法、DWA 算法、图论最短算法、遗传算法、元胞自动机算法、免疫算法、禁忌搜索算法、模拟退火算法、人工神经网络算法、蚁群算法、粒子群算法等。其中，基于地图的机器人路径规划算法最常用的是 Dijkstra 和 A*。最早在 1968 年研究人员提出了基于栅格法的路径规划方法，通过将地图划分为多个栅格，简化地图中的环境特征用于规划移动路径。随着研究的深入，路径规划被分为全局路径规划与局部路径规划两个部分。1959 年荷兰计算机学家狄克斯特拉提出了 Dijkstra（狄克斯特拉）算法，得到广泛应用，成为最基础

的最短路径搜索算法。相比 Dijkstra 算法，A*算法增加了估价函数，在最短时间内找到最优路径，代价最小，虽然牺牲了一部分精度但节省了遍历全地图的时间与计算资源。

1.3 报告的主要内容及章节安排

本文利用低成本二维激光测距传感器实现了一种 SLAM 算法，在基于机器人操作系统的嵌入式平台上实现了即时定位与地图构建，搭建了基于激光 SLAM 的自主导航和避障的移动机器人。

第一章为绪论，简要说明了研究课题的技术背景及研究意义。

第二章介绍了机器人的硬件系统设计。

第三章构建了基于激光 SLAM 的自主导航和避障的四轮机器人的系统设计，介绍了系统的整体框架、软件平台和功能模块。

第四章简要介绍了 gmapping 的原理，并分析了选择 gmapping 建图的原因，以及它与谷歌开源的 cartographer 的建图的对比。

第五章介绍了 amcl 定位的原理和应用。

第六章简要阐述了路径规划的代价计算及相关算法。

第七章介绍了 pure-pursuit 及其优化。

第八章介绍了机械上的改进。

第九章介绍了单片机与 pc 端通信。

第二章　硬件系统设计

2.1 电路整体说明

电路的稳定是智能车运行的前提，所以在设计的过程中应该追求电路板的稳定，下面所示的图片是针对室外光电组别设计的电路板，总共有三块电路板，分别是主控板，OLED 显示板，电调电量测量转接板。

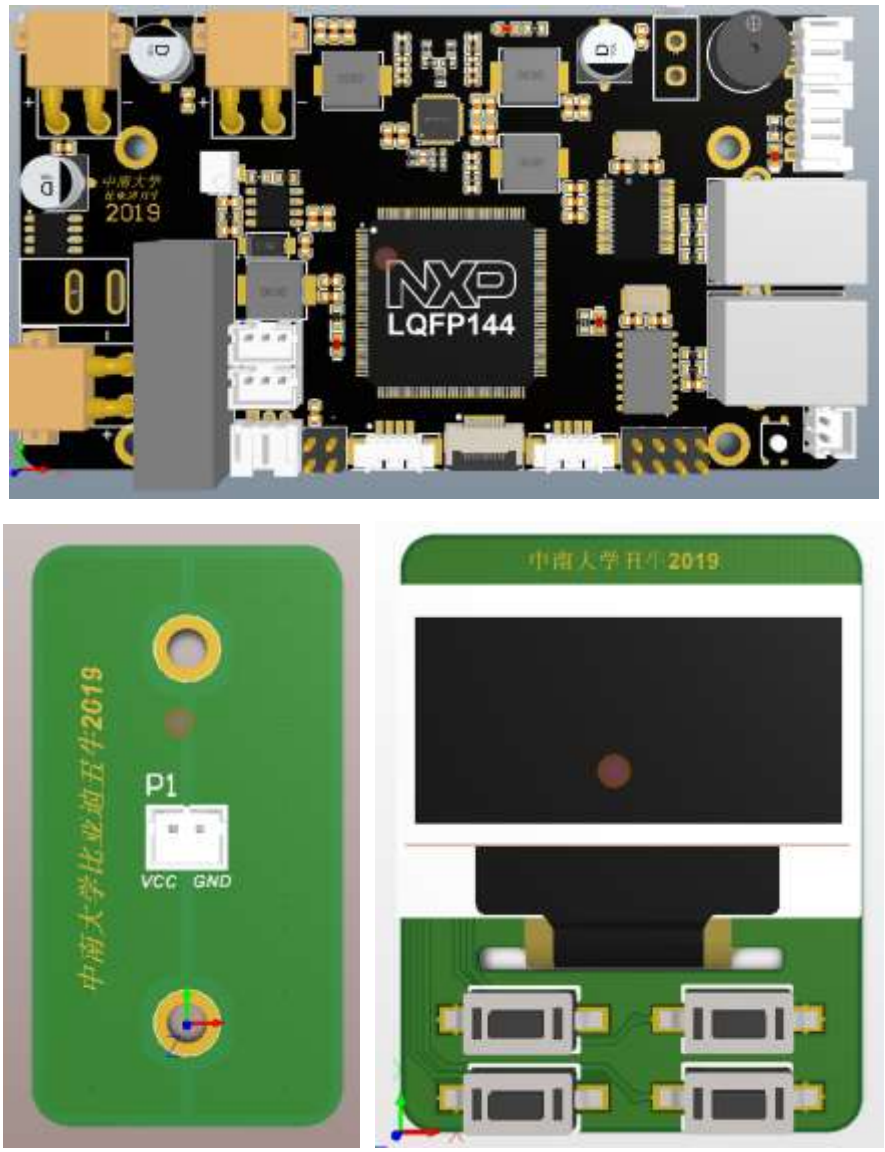


图 2.1 电路板正面图片

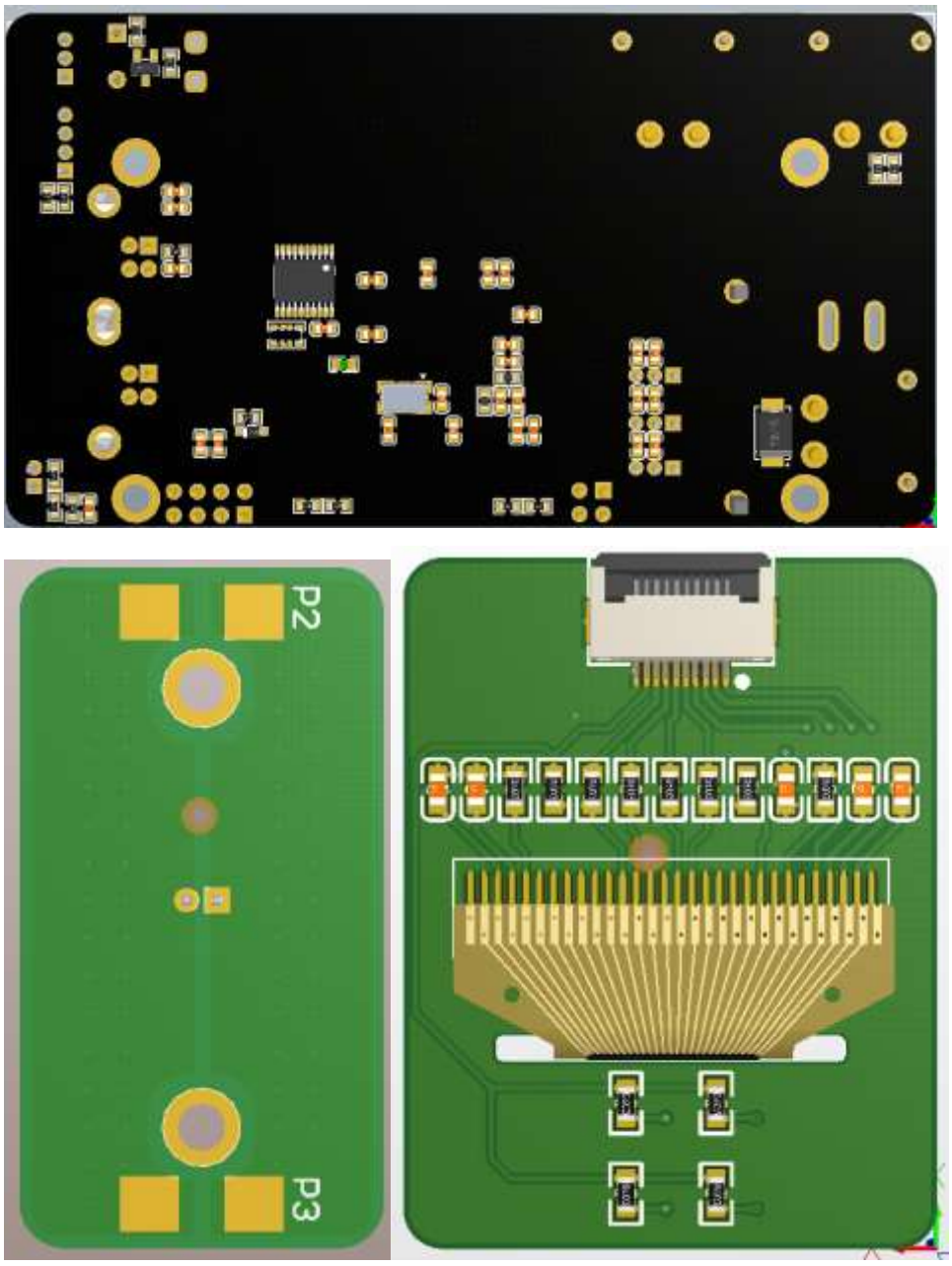


图 2.2 电路板背面图片

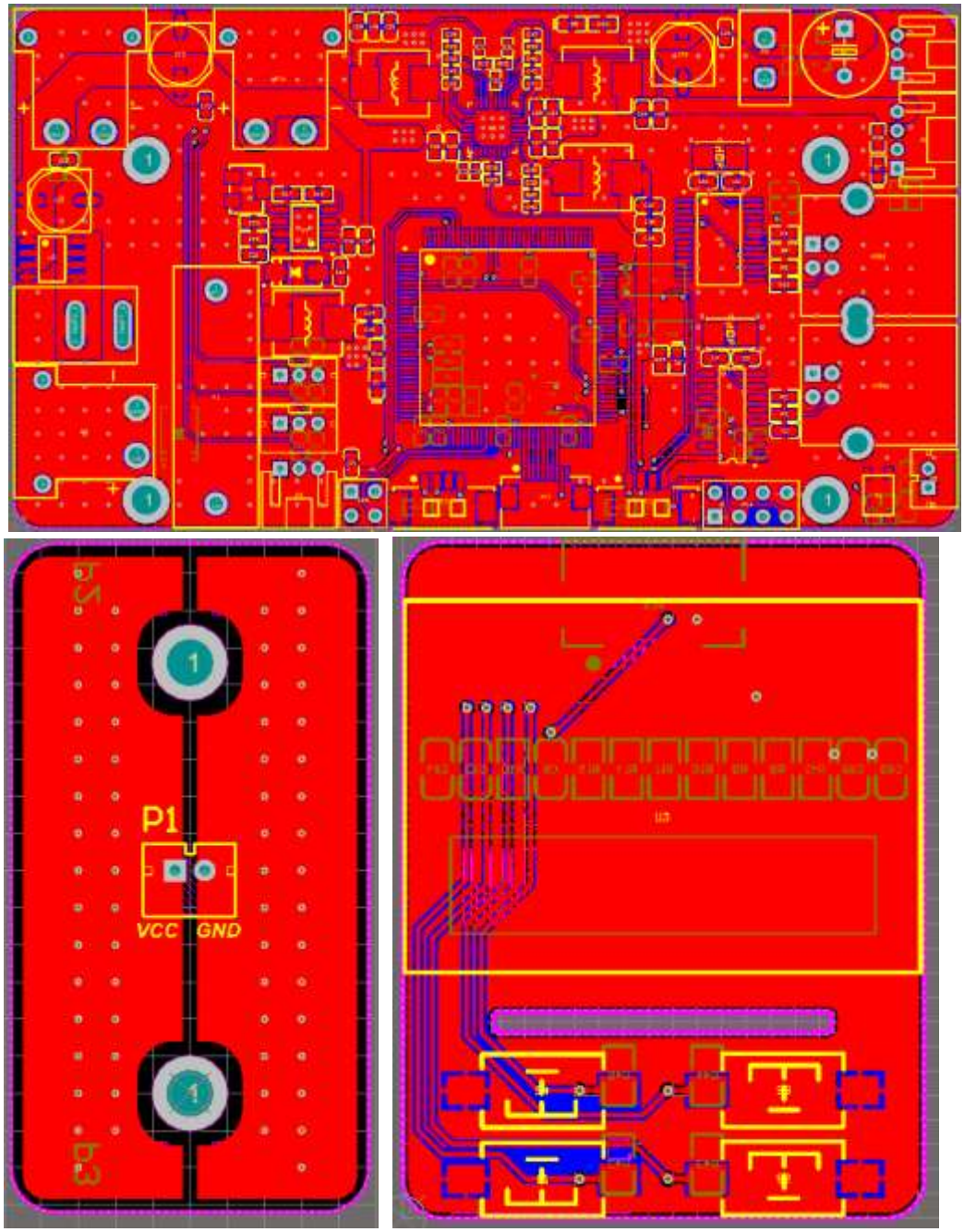


图 2.3 PCB top layer

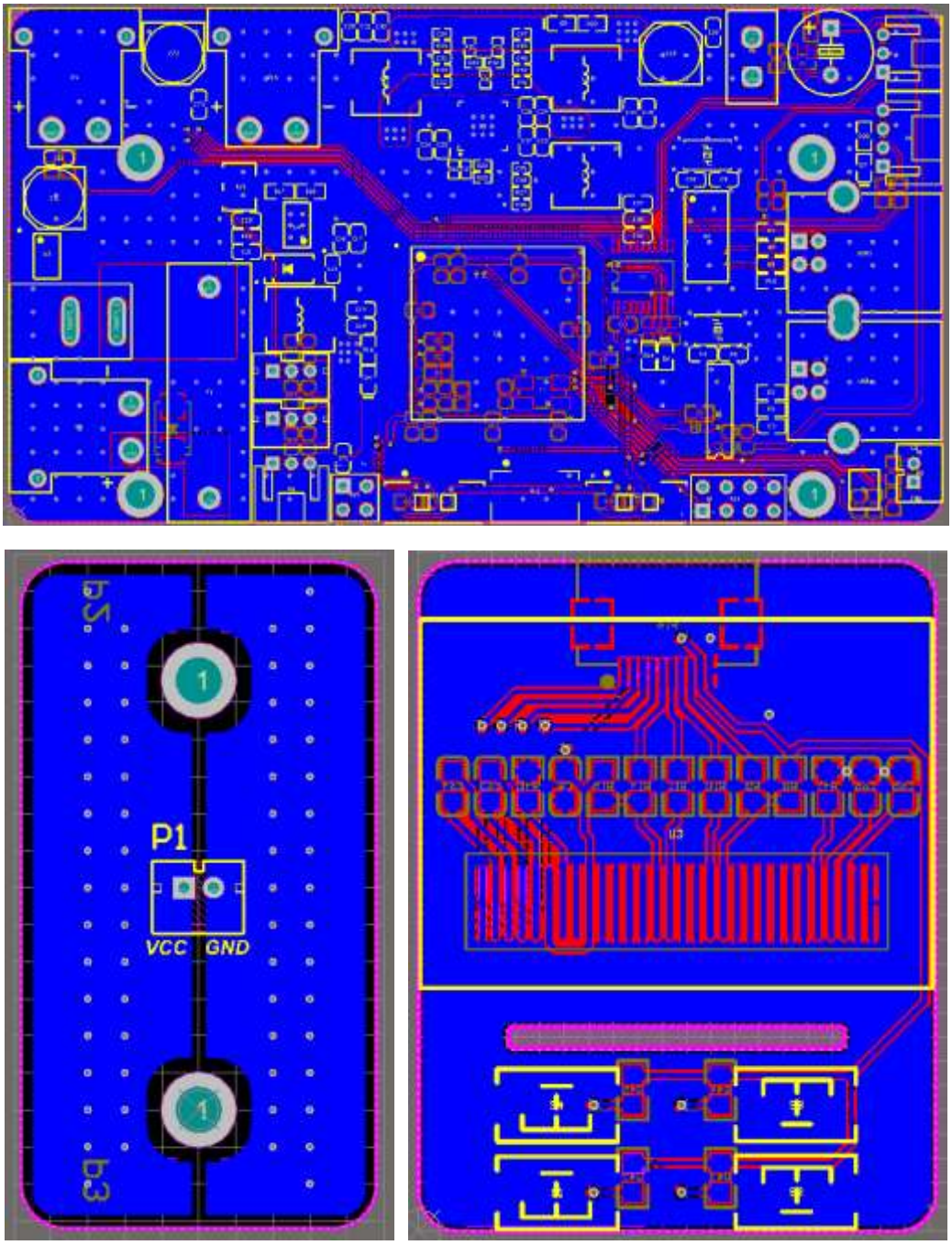


图 2.4 PCB bottom layer

2.2 单片机最小系统

主控芯片采用的是飞思卡尔公司的基于 Cortex-M4 内核的 Kinetis 系列的单片机 MK66FN 系列。

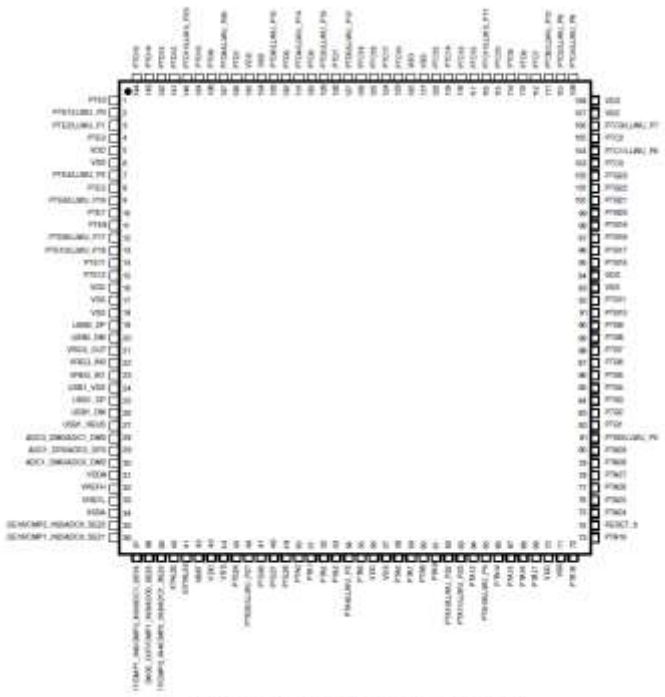


Figure 39. K66 144 LQFP Pinout Diagram

图 2.5

2.3 电源电路设计

2.3.1 电脑接口及 12V 路由器接口

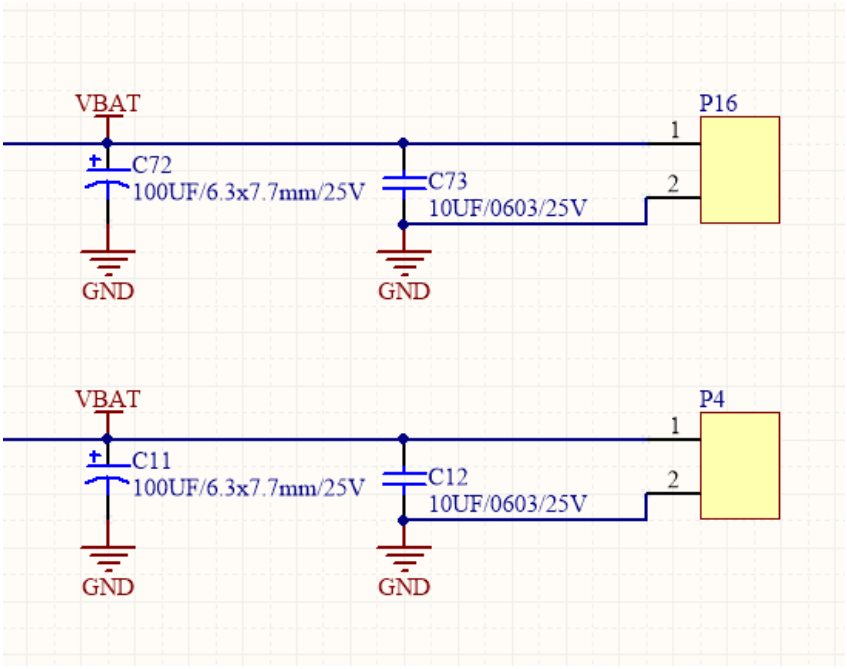


图 2.6

2.3.2 使用 TPS54560 的 5V 发生电路

TPS54560 是具有 Eco-mode™ 的 4.5V 至 60V 输入、5-A、降压直流/直流转换器，用 TPS54560 产生的 5V 用于给舵机供电。

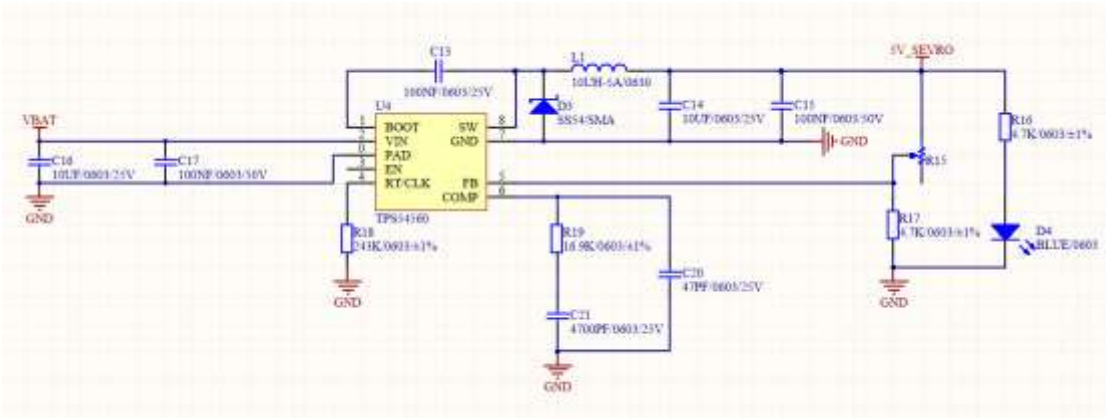


图 2.7

2.3.3 使用 TPS65265 的 5V/3.3V 发生电路

TPS65265 整合了三路同步降压转换器，支持 4.5V 至 17V 宽范围输入电压。 这款转换器具有恒定频率峰值电流模式，专用于简化应用，同时方便设计人员根据目标应用来优化系统。可通过外部电阻或外部时钟在 250kHz 至 2.3MHz 范围内调节转换器的开关频率。Buck1、Buck2 和 Buck3 之间的 120° 异相运行可最大限度降低对输入滤波器的要求，可以同时产生三路电压，输出电流最大分别为 5A、3A、2A。

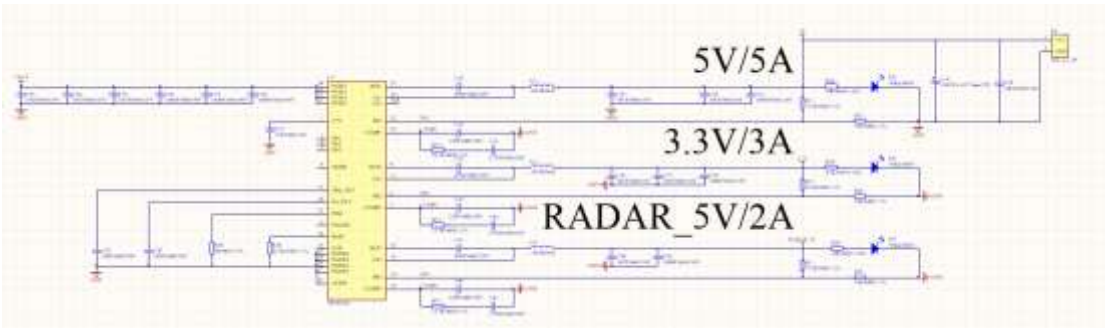


图 2.8

2.4 通讯电路设计

单片机与电脑，电脑与雷达之间需要通讯，需要将电脑的 USB 信号转换为 UART，在设计中使用两种 USB 转 UART 的芯片，CH340G 和 FT232RL，CH340G 比较便宜，FT232RL 更为稳定。

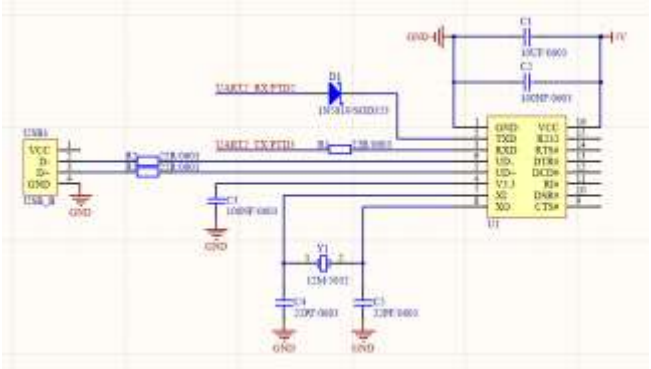


图 2.9

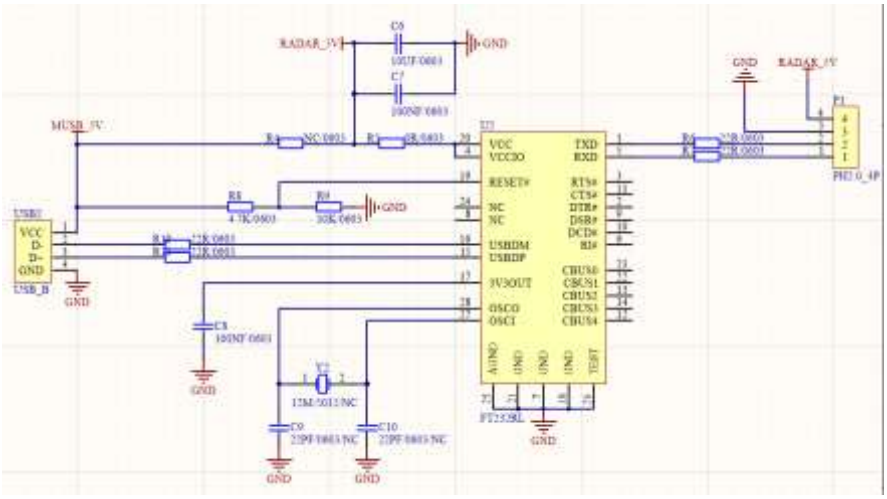


图 2.10

为了方便调试，使用了 WIFI 模块 ESP8266。

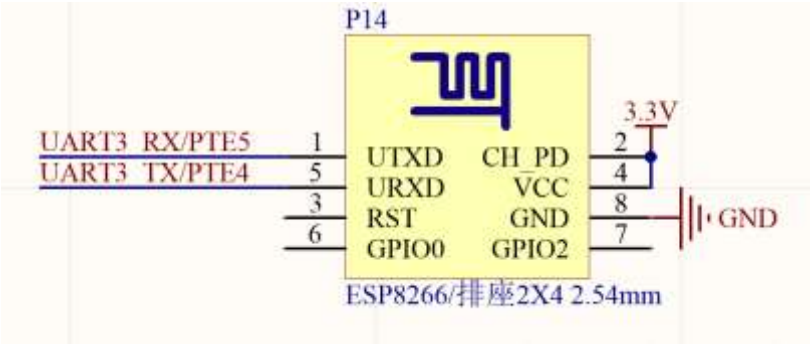


图 2.11

2.5 其他电路设计

2.5.1 隔离芯片

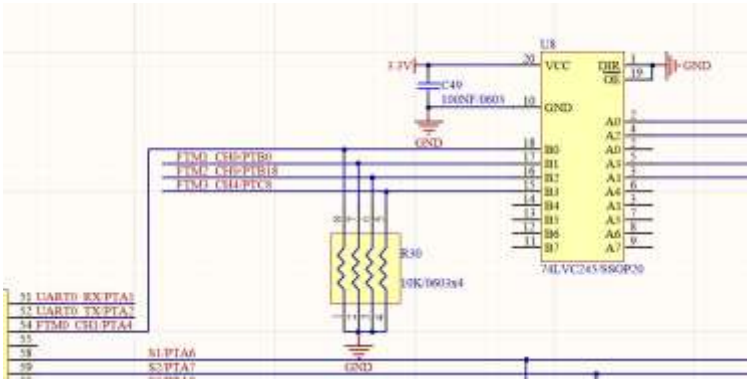


图 2.12

舵机及电调的输入信号与单片机之间加入了隔离芯片 74LVC245，减少了主控芯片损坏的可能性，并且单片机的输出经过隔离芯片后再作为舵机和电调的输入，当单片机没有输出时，处于下拉状态，不会造成舵机和电机的损坏。

2.5.2 按键及 OLED

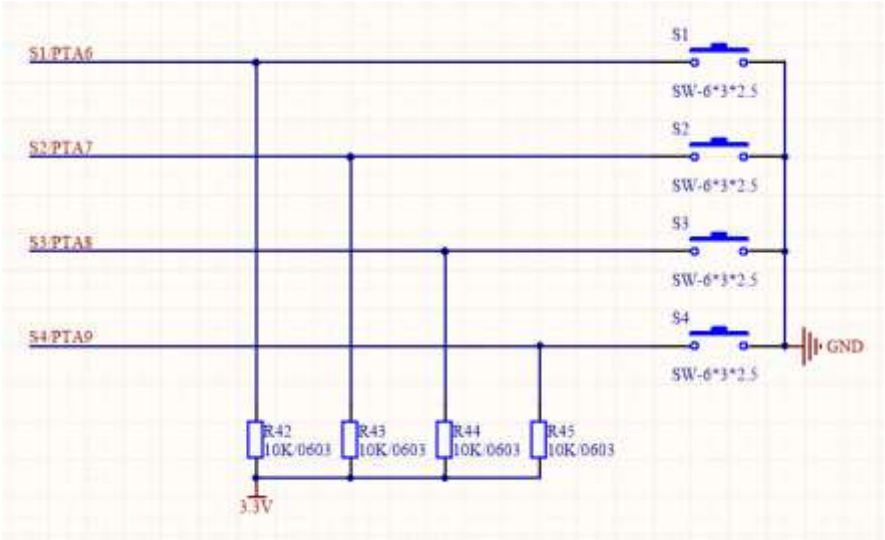


图 2.13

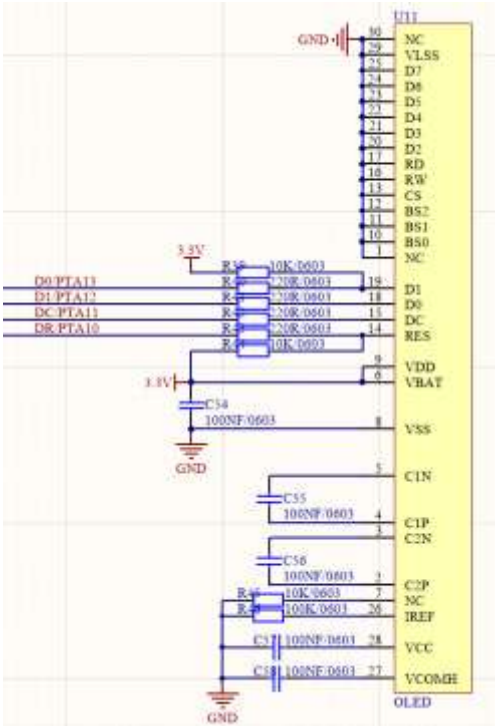


图 2.14

加入了按键和 OLED 使得调试参数更方便。

2.5.3 蜂鸣器

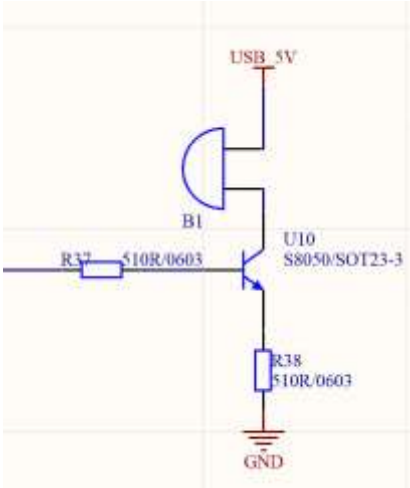


图 2.15

蜂鸣器的加入使得我们可以更方便的了解车在行驶过程中产生一定现象时的状态，然后进行修改。

第三章　软件整体框架

3.1 整体框架

软件主要由以 ROS 机器人操作系统为框架主体，在本项目中，i3 处理器将 IMU 数据与激光里程计融合，解算出小车在世界坐标系的位置。使用 ssh 远程访问服务器（小车主机），向底盘的嵌入式微控制器发送运动指令，实现无刷直流电机和舵机控制电机的移动。通过移动，可以采集到整体的环境信息。并将这些采集到的数据进行处理，通过 SLAM 的算法完成对整个环境的地图的构建。当我们获取到未知环境的地图后，通过路径规划算法计算出最佳路径。由路径计算出控制信息，输出到小车底盘，实现自主导航和避障。

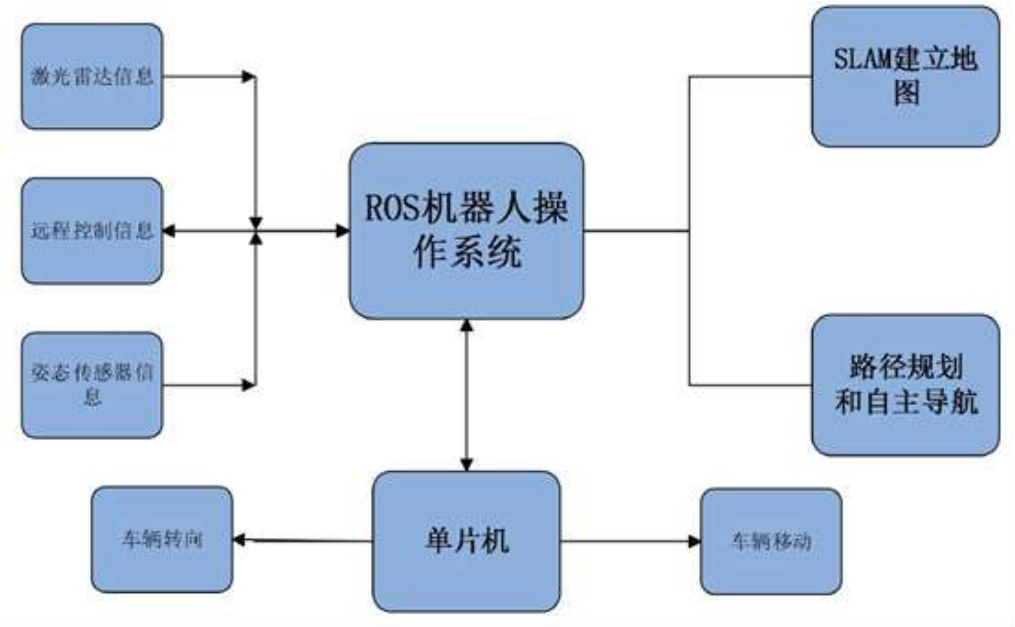


图 3.1

3.2 软件平台

ROS 是一种分布式处理框架，这使可执行文件能被单独设计，并且在运行时松散耦合。这些过程可以封装到数据包（Packages）和堆栈（Stacks）中，以便于共享和分发。ROS 还支持代码库的联合系统，这种从文件系统级别到社区一级的设计让独立地决定发展和实施工作成为可能。同时，该操作系统支持多种程序语言的程序编写，包括 C++、python 等，方便开发者选择自己偏好的方式进行研究和开发。

ROS 可以建立节点管理器，并自带可视化工具 RVIZ，如下图 2-4 所示。通过节点（Node）执行控制系统中的功能时，通常使用多个节点对应多种功能，比如一个节点运行激光雷达、读取数据，另一个节点控制 PWM 输出到电调，一个节点执行导航功能，另一个节点执行全部过程的可视化。

机器人操作系统通过主题（Topic）的发布和订阅进行节点之间的信息（Message）交换，主题的订阅与发布采用多对多的方式，主题之间一对一进行信息交换时，则是采用服务（Service）这种通信方式，两种方式并行，让机器人的多个功能在软件平台中联系在一起，形成一个紧密的整体。

RVIZ 界面是机器人操作系统自带的一个程序功能包，通过终端运行命令行指令可以安装 RVIZ 程序，通过在终端中输入 RVIZ 可以直接启动图形显示界面。通过左侧菜单栏可以添加插件，进行系统发布中主题数据的图形化显示。比如本文中激光雷达点云数据的查看可以通过添加 Laserscan 进行查看，右侧界面会显示出激光雷达在此时刻扫描周围环境得到的点云图。Rviz 也可以用来显示 Path 和 Odometry 等，极大程度帮助了我们分析程序的可靠性。

3.3 功能模块

- 底盘驱动

底盘驱动节点是整个软件系统中最基础的部分，它实现软件与硬件信息互

换的功能。在本机器人中，控制车体运动的指令有两部分内容，分别为车体的行进速度与行进方向，对应着直流电机的转速和舵机的摆角。在本软件系统中，控制车体运动的指令以名为 Cmd 的话题进行发布,而底盘驱动节点 (base 节点) 通常是这一话题的唯一订阅者。Base 节点运行在机载计算机上，它接收到指令以后，根据指令的内容向 RS232 串口发送数据。ARM 控制器再对这些数据进行响应，最终改变机器人的运动状态。

• 远程控制

远程控制是移动机器人的一项基本功能，它涉及指令的产生、传送、执行三个部分。ROS 平台具有实现分布式计算的能力，可以非常容易地实现远程操控功能。在本机器人中，远程控制节点（Teleop 节点）是运行在监控计算机上的控制程序。操作人员通过操作监控计算机上的键盘和手柄产生控制指令，Teleop 节点利用 ROS 提供的驱动程序读取到这些设备的输入数据，然后通过发布话题的方式将数据发布到软件节点网络上。机载计算机上运行的 Base 节点通过订阅这一话题来得到指令数据，然后以此驱动底盘运动。

•雷达信息采集

二维激光雷达和三维激光雷达在无人驾驶车辆上得到了广泛应用。与三维激光测距雷达相比，二维激光雷达只在一个平面上扫描，结构简单、测距速度快、系统稳定可靠，但是将二维激光雷达用于地形复杂、路面高低不平的环境时，无法完成地形的重建工作，且容易出现数据失真和虚报等现象；而三维激光雷达则可以获得车辆周围的环境深度信息，准确发现车辆周围的障碍物并构建地面环境即可行驶区域。

• 可通行区域识别

可行驶区域检测流程图如上所示。在接收到激光雷达输出的原始点云数据之后，通过坐标转换形成点云的栅格化表述，并从中区分地面点的集合以及地面以上障碍点的集合，完成地面和障碍物分离，形成地面估计与分割。

- 车辆导航

机器人在未知环境中需要使用激光传感器（或者将深度传感器转换为激光数据），先进行地图建模，然后根据构建的地图进行导航与定位。在 ROS 中可以利用以下三个功能来实现自主导航。

gmapping:根据激光数据（深度数据模拟激光数据）构建地图。

move_base:将全局导航和局部导航链接在一起以完成其导航任务，全局导航用于建立到地图上最终目标或一个远距离目标的路径，局部导航用于建立到近距离目标和为了临时躲避障碍物的路径。

amcl:英文全称是 adaptive Monte Carlo localization，是优化的蒙特卡洛法，实现机器人的定位。

第四章　gmapping 建图

4.1 原理介绍

gmapping 是目前应用最广的 2D slam 方法，利用粒子滤波算法。一般包括 4 个部分

（1）给定初始位姿，初始化粒子群，采用高斯分布进行随机采样；

（2）根据运动模型模拟粒子运动；

（3）计算粒子评分：采用 bresenham 直线段扫面算法，计算出粒子当前位置与障碍物之间的栅格占据集合，再计算出粒子当前位置与障碍物之间的栅格占据集合，从而对每个粒子进行评分，选择得分高的粒子作为该时间点的机器人位姿。

（4）粒子群重采样：舍弃评分低的粒子，保留高分粒子，并复制高分粒子以保持粒子数量不变

粒子滤波的方法一般需要大量的粒子来获取好的结果,但这必会引入计算的复杂度； 粒子是一个依据过程的观测逐渐更新权重与收敛的过程,这种重采样的过程必然会代入粒子耗散问题(depletion problem)，大权重粒子显著,小权重粒子会消失(有可能正确的粒子模拟可能在中间的阶段表现权重小而消失)。自适应重采样技术引入减少了粒子耗散问题，计算粒子分布的时候不单单仅依靠机器人的运动(里程计)，同时将当前观测考虑进去，减少了机器人位置在粒子滤波步骤中的不确定性。

4.2 与 cartographer 建图对比

Gmapping 可以实时构建室内地图，在构建小场景地图所需的计算量较小且精度较高。相比 Hector SLAM 对激光雷达频率要求低、鲁棒性高（Hector 在机器人快速转向时很容易发生错误匹配，建出的地图发生错位，原因主要是优化

算法容易陷入局部最小值）；而相比 Cartographer 在构建小场景地图时，Gmapping不需要太多的粒子并且没有回环检测因此计算量小于Cartographer而精度并没有差太多。Gmapping有效利用了车轮里程计信息，这也是Gmapping对激光雷达频率要求低的原因：里程计可以提供机器人的位姿先验。而Hector和Cartographer的设计初衷不是为了解决平面移动机器人定位和建图，Hector主要用于救灾等地面不平坦的情况，因此无法使用里程计。而Cartographer是用于手持激光雷达完成SLAM过程，也就没有里程计可以用。

随着场景增大所需的粒子增加，因为每个粒子都携带一幅地图，因此在构建大地图时所需内存和计算量都会增加。因此不适合构建大场景地图。并且没有回环检测，因此在回环闭合时可能会造成地图错位，虽然增加粒子数目可以使地图闭合但是以增加计算量和内存为代价。所以不能像Cartographer那样构建大的地图，虽然论文生成几万平米的地图，但实际我们使用中建的地图没有几千平米时就会发生错误。Gmapping和Cartographer一个是基于滤波框架SLAM另一个是基于优化框架的 SLAM，两种算法都涉及到时间复杂度和空间复杂度的权衡。Gmapping牺牲空间复杂度保证时间复杂度，这就造成Gmapping不适合构建大场景地图，试想一下你要构建200乘200米的环境地图，栅格分辨率选择5厘米，每个栅格占用一字节内存，那么一个粒子携带的地图就需要 16M 内存，如果是100个粒子就需要1.6G内存。如果地图变成500乘500米，粒子数为200个，可能电脑就要崩溃了。翻看Cartographer算法，优化相当于地图中只用一个粒子，因此存储空间比较Gmapping会小很多倍，但计算量大，一般的笔记本很难跑出来好的地图，甚至根本就跑不动。以下是 gmapping（左）与cartographer(右)对比。

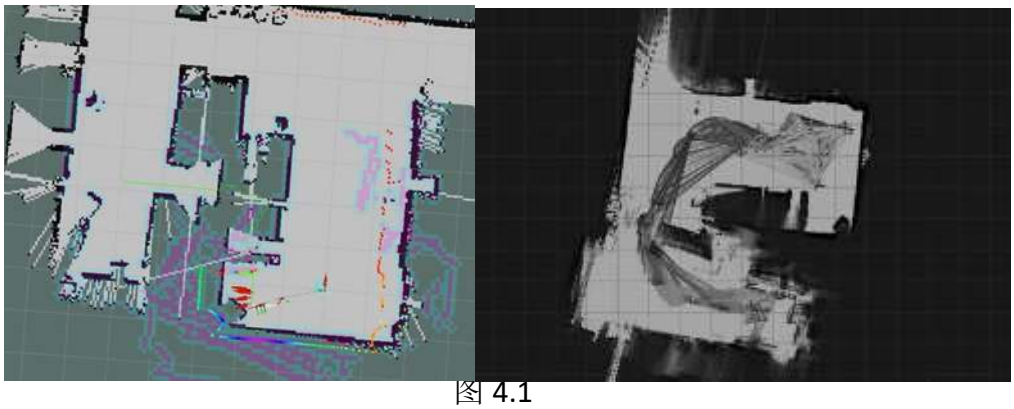


图 4.1

4.3 与 Hector SLAM 建图对比

Hector SLAM 是基于优化的算法（解最小二乘问题），采用泰勒展开近似优化激光雷达数据的匹配过程，因此两次激光雷达采集数据时位姿变化不能太大，否则余项误差过大，造成建图失败,所以对于雷达帧率要求很高，至少要 40Hz,，可以适应空中或者地面不平坦的情况。初值的选择对结果影响很大，所以要求雷达帧率较高。

由于本次比赛规定采用的雷达 ls01d 采样频率为 10hz，显然采用 hector SLAM 并不合适

第五章　AMCL 定位

5.1 相关概念

• 粒子滤波和蒙特卡洛

蒙特卡洛：是一种思想或方法。举例：一个矩形里面有个不规则形状，怎么计算不规则形状的面积？不好算。但我们可以近似。拿一堆豆子，均匀的撒在矩形上，然后统计不规则形状里的豆子的个数和剩余地方的豆子个数。矩形面积知道的呀，所以就通过估计得到了不规则形状的面积。拿机器人定位来讲，它处在地图中的任何一个位置都有可能，这种情况我们怎么表达一个位置的置信度呢？我们也使用粒子，哪里的粒子多，就代表机器人在哪里的可能性高。

粒子滤波：粒子数代表某个东西的可能性高低。通过某种评价方法（评价这个东西的可能性），改变粒子的分布情况。比如在机器人定位中，某个粒子 A，我觉得这个粒子在这个坐标（比如这个坐标就属于之前说的“这个东西”）的可能性很高，那么我给他打高分。下次重新安排所有的粒子的位置的时候，就在这个位置附近多安排一些。这样多来几轮，粒子就都集中到可能性高的位置去了。

• 重要性采样

就像转盘抽奖一样，原本分数高（我们给它打分）的粒子，它在转盘上对应的面积就大。原本有 100 个粒子，那下次我就转 100 次，转到什么就取个对应的粒子。这样多重复几次，仍然是 100 个粒子，但是分数高的粒子越来越多了，它们代表的东西（比如位姿）几乎是一样的。

• 机器人绑架

举例，机器人突然被抱走，放到了另外一个地方。类似这种情况。

• 自适应蒙特卡洛

自适应体现在：1 解决了机器人绑架问题，它会在发现粒子们的平均分数突然降低了（意味着正确的粒子在某次迭代中被抛弃了）的时候，在全局再重新的撒一些粒子。

解决了粒子数固定的问题，因为有时候当机器人定位差不多得到了的时候，比如这些粒子都集中在一块了，还要维持这么多的粒子没必要，这个时候粒子数可以少一点了。

• KLD 采样

为了控制上述粒子数冗余而设计的。比如在栅格地图中，看粒子占了多少栅格。占得多，说明粒子很分散，在每次迭代重采样的时候，允许粒子数量的上限高一些。占得少，说明粒子都已经集中了，那就将上限设低，采样到这个数就行了。

5.2 原理介绍

amcl 是移动机器人二维环境下的概率定位系统。它实现了自适应（或 kld 采样）的蒙特卡罗定位方法，其中针对已有的地图使用粒子滤波器跟踪一个机器人的姿态。

a. MCL 计算机器人所在位置的概率

1: **Algorithm MCL**($\mathcal{X}_{t-1}, u_t, z_t, m$):
2: $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
3: *for* $m = 1$ *to* M *do*
4: $x_t^{[m]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$
5: $w_t^{[m]} = \textbf{measurement\_model}(z_t, x_t^{[m]}, m)$
6: $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7: *endfor*
8: *for* $m = 1$ *to* M *do*
9: *draw* i *with probability* $\propto w_t^{[i]}$
10: *add* $x_t^{[i]}$ *to* $\mathcal{X}_t$
11: *endfor*
12: *return* $\mathcal{X}_t$

b. 机器人可能具有硬件误差，因此建立传感器模型和移动模型，再执行贝叶斯滤波器的预测和更新。

c. 矫正步骤，利用传感器模型，前面求得的概率 bel’(Xt)和归一化常数ηt,可以求得基于传感器信息提高准确度的当前位置的概率 bel(Xt)

d. 用粒子滤波器生成 N 个粒子来估计位置。

e. 抽样，这里使用前一个位置的概率 bel(Xt-1)中的机器人的移动模型来提取新的样本集合 Xt’。样本 Xt’中的第 i 个样本 Xt’距离传感器获得的距离信息 Zt 和归一化常数ηp 来计算权重值 Wt。

f.重权重采样过程中，我们使用样本 Xt’和 Wt’来创建 N 个新的样本（粒子）集合 Xt。

5.3 激光似然场模型

• beam model 测量光束模型

是纯物理模型，针对激光发出的测量光束建模，将一次测量误差分解为四个误差。其期望值的计算需要用 retraying，每一个位姿需要进行 N 次 retracing， 为一帧激光的激光束数量。

在非结构化环境中（clutter），位姿微小的改变会造成期望值的巨大变化，从而导致得分进行突变。

• likehood field model

其和测量光束模型相比，考虑了地图的因素，不再是对激光的扫描线物理建模，而是考虑测量到的物体的因素。其对图像进行高斯平滑，在任何环境中的期望值对于位姿都是平滑的。而且，该模型得分的计算不需要经过 raytracing，直接通过查表即可得到，计算量低。

1: **Algorithm likelihood_field_range_finder_model**(z_t, x_t, m):

2: $q = 1$

3: *for all* k *do*

4: *if* $z_t^k \neq z_{\max}$

5: $x_{z_t^k} = x + x_{k,\text{sens}} \cos\theta - y_{k,\text{sens}} \sin\theta + z_t^k \cos(\theta + \theta_{k,\text{sens}})$

6: $y_{z_t^k} = y + y_{k,\text{sens}} \cos\theta + x_{k,\text{sens}} \sin\theta + z_t^k \sin(\theta + \theta_{k,\text{sens}})$

7: $dist = \min_{x',y'} \left\{ \sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} \,\middle|\, \langle x', y' \rangle \text{ occupied in } m \right\}$

8: $q = q \cdot \left(z_{\text{hit}} \cdot \mathbf{prob}(dist, \sigma_{\text{hit}}) + \frac{z_{\text{random}}}{z_{\max}} \right)$

9: *return* q

第六章　路径规划

6.1 代价地图

在 ROS 中，代价地图由静态地图层、障碍地图层和膨胀层组成：

静态地图层（Static Map Layer），基本上不变的地图层，通常都是 SLAM 建立完成的静态地图。

障碍地图层（Obstacle Map Layer），用于动态的记录传感器感知到的障碍物信息。

膨胀层（Inflation Layer），在以上两层地图上进行膨胀（向外扩张），以避免机器人的撞上障碍物。

此外，还有一个全局代价地图，以及一个局部代价地图。全局代价地图通过地图上的障碍膨胀生成，是为全局路径规划准备的。局部代价地图通过对机器人传感器实时检测到的障碍物进行膨胀生成，是为局部路径规划准备的。

代价地图点每个单元中可以有 255 个不同的代价值，但它采用的底层的结构只能表示 3 个。具体的，该结构中的每个单元可以是空闲/占有/未知。每个状态都有特殊的代价值被分配到 costmap 中去。图 6.1 显示了如何计算膨胀衰减曲线。

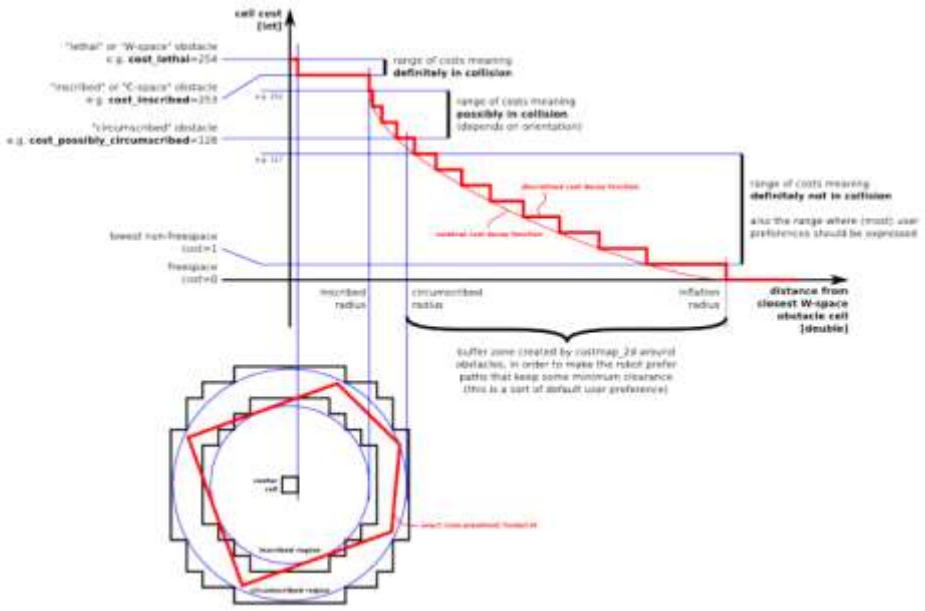


图 6.1

上图共分为五个部分：（下面的红色框图是机器人的轮廓，旁边的黑框是上图的映射位置）

（1） Lethal（致命的）:机器人的中心与该网格的中心重合，此时机器人必然与障碍物冲突。

（2） Inscribed（内切）：网格的外切圆与机器人的轮廓内切，此时机器人也必然与障碍物冲突。

（3） Possibly circumscribed（外切）：网格的外切圆与机器人的轮廓外切，此时机器人相当于靠在障碍物附近，所以不一定冲突。

（4） Freespace（自由空间）：没有障碍物的空间。

（5） Unknown（未知）：未知的空间。

代价地图在 ROS 中使用 costmap_2d 软件包实现,程序架构如图 6.2 所示。

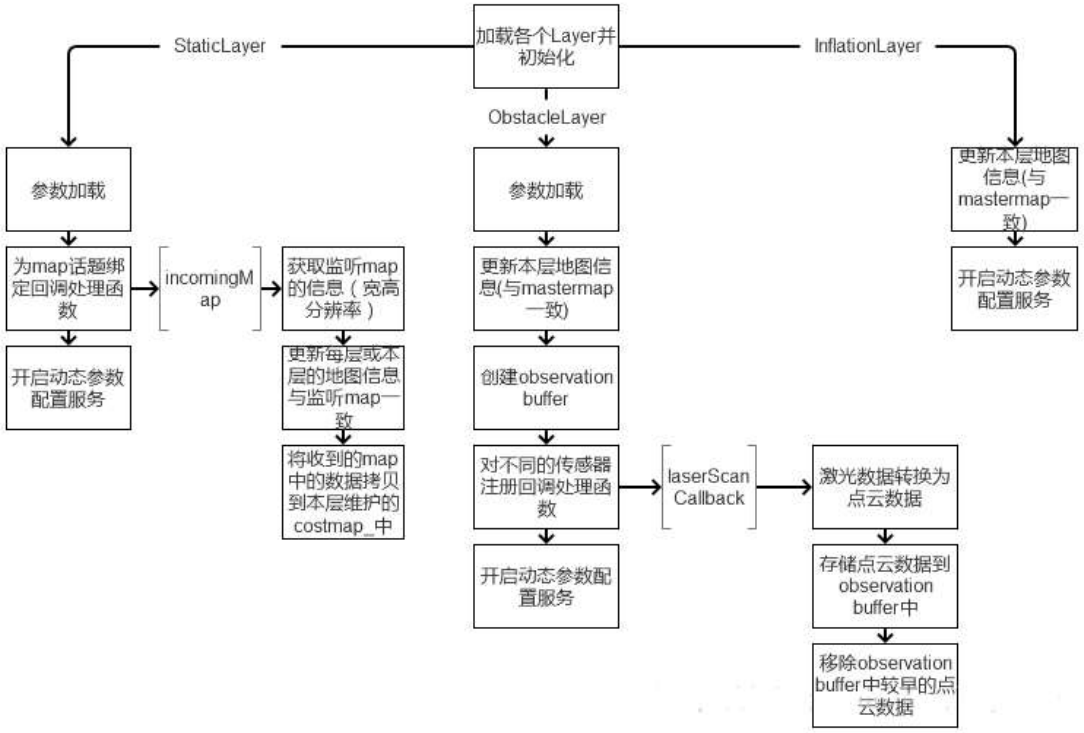


图 6.2

6.2 代价地图参数调整

• footprint

足迹，即机器人的轮廓。该占位面积将用于计算内切圆和外接圆的半径，用于以适合此机器人的方式对障碍物进行膨胀,通常为了安全起见，我们希望占用空间稍大于机器人的真实轮廓。

• global_frame

该参数用来表示全局代价地图需要在那个参考系下运行，这里我们选择了 map 参考系。

• robot_base_frame

该参数表示代价地图可以参考的机器人本体的参考系，这里我们选择了

base_footprint 参考系。

• update_frequency

该参数设置全局地图信息更新的频率，单位是 Hz。

• publish_frequency

该参数设置代价地图发布可视化信息的频率，单位是 Hz。

• static_map

该参数决定代价地图是否需要根据 map_server 提供的地图信息进行初始化，如果不需要使用已有的地图或者 map_server，则将该参数设置为 false。

• rolling_window

该参数是用来设置在机器人移动过程中是否需要滚动窗口算法，以保持机器人处于中心位置，我们将该参数设置为 false。

• plugins

图层，包括静态地图层，障碍地图层和膨胀层。

• width height resolution

width、height 和 resolution 设置设置代价地图长（米）、高（米）和分辨率（米/格）。分辨率可以设置的与静态地图不同，但是一般情况下两者是相同的。我们发现分辨率 0.1 可以满足要求。

• cost_scaling_factor （代价缩放因子）

与单元代价成反比。将其调高将使衰变曲线更加陡峭。Pronobis 提出，最优代价图衰减曲线是一条斜率相对较低的曲线，因此最优路径是尽可能远离两侧障碍物的最优路径。其优点是机器人更喜欢在障碍物中间移动，在相同的起点和目标下，当代价图曲线陡峭时，机器人往往接近障碍物。

6.3 Dijkstra 算法

迪杰斯特拉(Dijkstra)算法是典型最短路径算法，用于计算一个节点到其他节点的最短路径。它使用了广度优先搜索解决赋权有向图或者无向图的单源最短路径问题，算法最终得到一个最短路径树。该算法常用于路由算法或者作为其他图算法的一个子模块。

基本思想：

通过 Dijkstra 计算最短路径时，需要指定起点 s(即从顶点 s 开始计算)。

此外，引进两个集合 S 和 U。S 的作用是记录已求出最短路径的顶点(以及相应的最短路径长度)，而 U 则是记录还未求出最短路径的顶点(以及该顶点到起点 s 的距离)。

初始时，S 中只有起点 s；U 中是除 s 之外的顶点，并且 U 中顶点的路径是“起点 s 到该顶点的路径”。然后，从 U 中找出路径最短的顶点，并将其加入到 S 中；接着，更新 U 中的顶点和顶点对应的路径。 然后，再从 U 中找出路径最短的顶点，并将其加入到 S 中；接着，更新 U 中的顶点和顶点对应的路径。 …重复该操作，直到遍历完所有顶点。

操作步骤：

（1）初始时，S 只包含起点 s；U 包含除 s 外的其他顶点，且 U 中顶点的距离为”起点 s 到该顶点的距离”[例如，U 中顶点 v 的距离为(s,v)的长度，然后 s 和 v 不相邻，则 v 的距离为∞]

（2）从 U 中选出”距离最短的顶点 k”，并将顶点 k 加入到 S 中；同时，从 U 中移除顶点 k。

（3）更新 U 中各个顶点到起点 s 的距离。之所以更新 U 中顶点的距离，是由于上一步中确定了 k 是求出最短路径的顶点，从而可以利用 k 来更新其它顶

点的距离；例如，(s,v)的距离可能大于(s,k)+(k,v)的距离。

（4）重复步骤(2)和(3)，直到遍历完所有顶点。

A*算法与 Dijkstra 相似，但其启发式的向目标搜索，大大减小了运算量，能更快的收敛于一个最短路径。但同时，也有一些弊端。二者对比如下：

（1）Dijkstra 算法计算源点到其他所有点的最短路径长度，A*关注点到点的最短路径(包括具体路径)。

（2）Dijkstra 算法的实质是广度优先搜索，是一种发散式的搜索，所以空间复杂度和时间复杂度都比较高。对路径上的当前点，A*算法不但记录其到源点的代价，还计算当前点到目标点的期望代价，是一种启发式算法，也可以认为是一种深度优先的算法。

（3）当目标点很多时，A*算法会带入大量重复数据和复杂的估价函数，所以如果不要求获得具体路径而只比较路径长度时，Dijkstra 算法会成为更好的选择。

实际应用过程中我们发现，A*会沿障碍呈锯齿形向终点搜索，而 Dijkstra 的路径则更为流畅、圆滑。

6.4 TEB 算法

TEB 局部规划算法的思想是：整个运动路径比作为一条橡皮筋，连接起始点，目标点，并让这个路径可以变形，变形的条件就是将所有约束当作橡皮筋的外力。起始点和目标点由全局规划器指定，在中间插入 N 个控制橡皮筋形状的控制点（控制对象的姿态），并在点与点之间定义运动时间，形成一个路径点序列与时间序列对应合并的一个新的序列，再通过加权多目标优化获取最优的路径点。

该算法中的主要约束为跟随路径和避障约束、速度与加速度约束、

Non-holonomic 运动学约束以及最快路径约束（因路径点在时间上均匀分开，所以不是传统的空间上最短路径）。

值得重视的是，该算法主要的三个特点：①它可以处理阿克曼底盘车辆的运动学约束。而 ROS 导航栈默认的局部规划算法 base local planner 只能处理万向轮或差速转向的车辆模型。②它是时间最短的最优控制器。在最大速度和加速度等参数约束的条件下，TEB 局部规划算法将规划出时间最短路径和速度指令。路径上表现为明显的切弯、靠近障碍物；速度指令上将会表现为速度和方向指令的快速振荡。

该算法具有上述一些很吸引编程人员的优点，但在实际系统中使用时，也会存在一些缺点与潜在问题：①TEB 规划器对电机和舵机的动态响应性能要求很高。在给定加速度约束的情况下，TEB 规划输出，即电机控制器的输入是一个斜坡输入。使用直流电机时，需要认真调试 PID 参数才能实现比较好的效果。②TEB 规划器计算量较高。③TEB 规划器的稳定性一般。由于不断重新规划路径，且实际车辆的定位测速不可能没有误差，在车辆静止时可以观察到舵机抖动。尽管降低规划频率(此频率由 move_base 的配置确定)有助于改善这一问题，但这同样导致车辆动态避障的性能大大降低。

第七章　转向控制器

7.1 自行车模型

转向控制器是基于纯跟踪控制器的，它是一种非线性路径跟随器，广泛应用于地面机器人，以及最近在无人机中使用。它因其简单性而被采用，是一种直观的控制律，具有明确的几何意义。由于 DUC 规则将运行速度限制在每小时 30 英里以下，避免了极端机动(如高速紧转弯)，因此在控制设计中忽略了侧滑等动态效应。

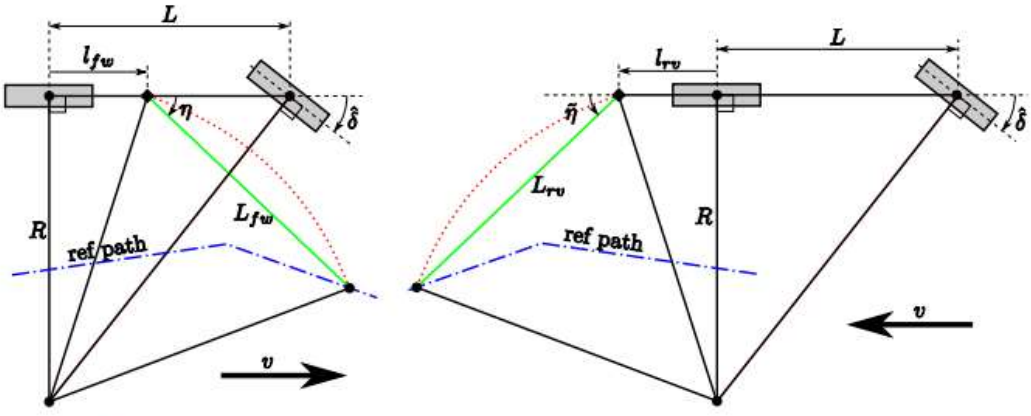
简化的自行车运动学模型如下：

$$\dot{x} = v\cos\theta, \qquad \dot{y} = v\sin\theta, \qquad \dot{\theta} = \frac{v}{L}\tan\delta$$

其中，L 为车轮轴距，x、y 为坐标。

7.2 Pure- Pursuit 模型

CL-RRT 以两种不同的方式使用控制器。一种是结合车辆模型的闭环预测，另一种是实时执行运动规划。我们采取后一种方式。



(a) Forward Drive　　(b) Reverse Drive

图中的转向角对应于方程中的负 δ 。

利用基本的平面几何，将锚点放在前瞻点的航向上，这时所需要的转向角可被计算为：

$$\delta = -\tan^{-1}\left(\frac{L\sin\eta}{\frac{L_{\text{fw}}}{2} + l_{\text{fw}}\cos\eta}\right) \quad \text{: forward drive}$$

$$\delta = -\tan^{-1}\left(\frac{L\sin\tilde{\eta}}{\frac{L_{\text{rv}}}{2} + l_{\text{rv}}\cos\tilde{\eta}}\right) \quad \text{: reverse drive}$$

其中 Lfw 为经验得出的瞄点距离，lfw、lrv 为车身转向控制点。以上，给出了修正的纯追踪控制律。注意，通过设置 LFW=0 和 LRV=0，锚点与后轴重合，恢复了传统的纯追踪控制器。

7.3 前视距离优化算法

通常，人们采用车辆的纵向速度缩放前视距离，即 Lfw = $kv_x(t)$ 。将前视距离设为速度的函数，试图获得更快速的转向。Yoshiaki Kuwata 等人提出

$$L_{\text{fw}}(v_{\text{cmd}}) = L_{\text{rv}}(v_{\text{cmd}}) = \begin{cases} 3 & \text{if } v_{\text{cmd}} < 1.34 \text{ m/s}, \\ 2.24\, v_{\text{cmd}} & \text{if } 1.34 \text{ m/s} \le v_{\text{cmd}} < 5.36 \text{ m/s}, \\ 12 & \text{otherwise.} \end{cases}$$

该分段算法是基于通过大量的模拟和现场试验的，针对的是特定车型的特定系统响应，不具有普遍性。

显然，较短的前瞻距离提供更精确的跟踪，而较长的距离则提供更及时的转向。即过小的 k 值会导致不稳定，过大的 k 值会引发“cutting corner”的问题。前视距离在稳定性和跟踪性能之间进行权衡，很难两者兼得。这在一定程度上是因为纯跟踪控制模型忽略了路径的曲率。

Lekkas 提出了一种改进的路径跟踪前视距离(LOS)制导算法。仿真结果证明了该策略的有效性，表明可变前视距离算法可以减少期望路径附近的振荡行为。.

$$\Delta(y_e) = (\Delta_{\max} - \Delta_{\min})e^{-\gamma|y_e|} + \Delta_{\min}.$$

γ 为衰减系数，ye 为航向偏差，前视距离为Δmin 到Δmax 中的一个值，其值以 γ 的速度随 Ye 衰减。

但其对全局速度小范围稳定的要求较高，一旦速度增大超过一定范围，考虑到控制的滞后和刚性物体的惯性，转向不及时将导致动态避障的不稳定性。同时，最小值、最大值以及衰减速度将极大的影响到横向控制的稳定性和快速性。

因此，我们结合了速度和曲率，给出以下的动态前瞻：

$$L = (a + V * k1) * (1 - |\theta| * k2)$$

其中，a 为前瞻的一个基础值，V 为速度，k1 为速度增益， θ 为曲率，k2 为曲率增益。

第八章　机械结构改进

8.1 车模结构调整

室外光电组由激光雷达、IMU、底盘、i3 处理器构建智能车系统。车模通过 4 根 35mm 的铜柱支撑起一块钢板，并将 IMU、激光雷达、i3 处理器等器件固定在钢板的上方。由于钢板上部的器件数量较多，重量偏大，使得整体车辆高度过高且重心偏上方。

整车重心高度作为车辆重要基本参数之一，对车辆的动力学性能有着重要影响，主要体现在动力性、制动性、稳定性等方面，例如加速、制动、转向时车身的姿态、转向特性都会受到车辆重心高度的影响。当车辆高速行驶，一旦转大弯，车辆就会发生倾斜，从而导致传感器的数据产生误差，对车辆实时避障产生不好的影响。因此我们对车模的结构进行了改进。

为了降低重心、降低车高，我们将支撑钢板的铜柱缩短为 20mm，同时计划把钢板上的计算机移至钢板与底盘中间，充分利用车模的中部空间。为防止计算机散热器对电机产生负面影响，可以将计算机翻转 180 度，使散热孔朝上。同时把 2S 电池放置到车辆中部，电调位置前移。

钢板上方的激光雷达是车模的最高点，而计算机移至下方后，就可以把雷达的支撑柱替换为 10mm。同时将电路板移至后方，给雷达、IMU 腾出空间。路由器后移避免遮挡散热孔。

除了重心、车高的影响外，车模的重量也是转向不佳的因素。所以我们在重新设计承重板（钢板）的结构与孔位期间，也将材料替换为了更轻的碳纤维。同时将大部分铜柱替换为了尼龙柱。碳纤维板设计图和实际效果图如下：

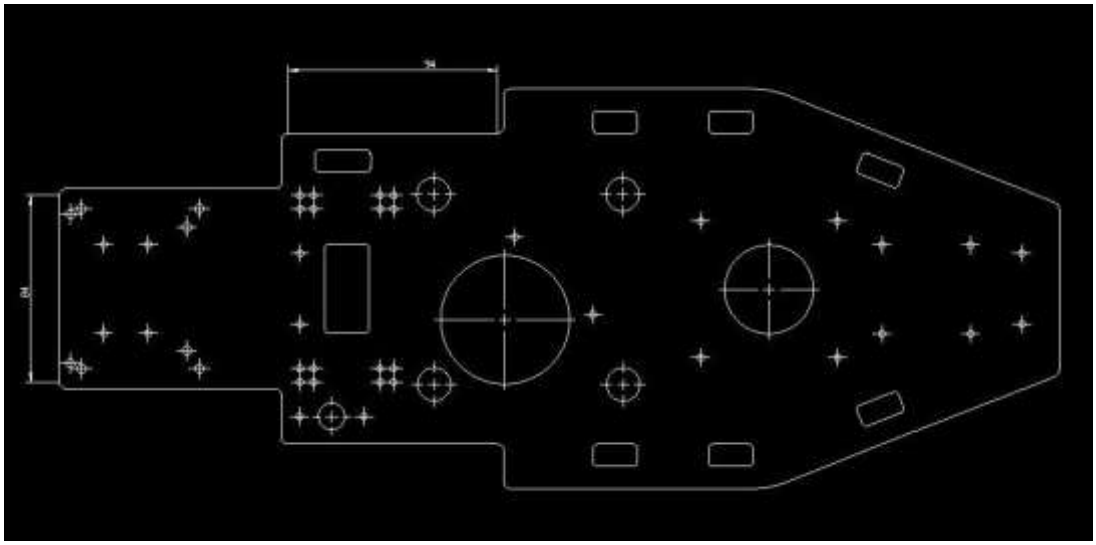


图 8.1 碳纤维板设计图

经过调整，车高、车重有了很大改善，车辆动力学性能也提升了许多。

8.2 编码器结构设计

室外光电组车辆的实际速度反馈对车辆的定位及速度控制非常关键。所以我们思考后绝对加装编码器。由于 G 车模最初设计为开环控制的遥控攀爬车，所以车模并没有预留编码器安装位置。所以拆除了抬头架，使用其螺丝孔加装了第五轮（如图 8.2）。

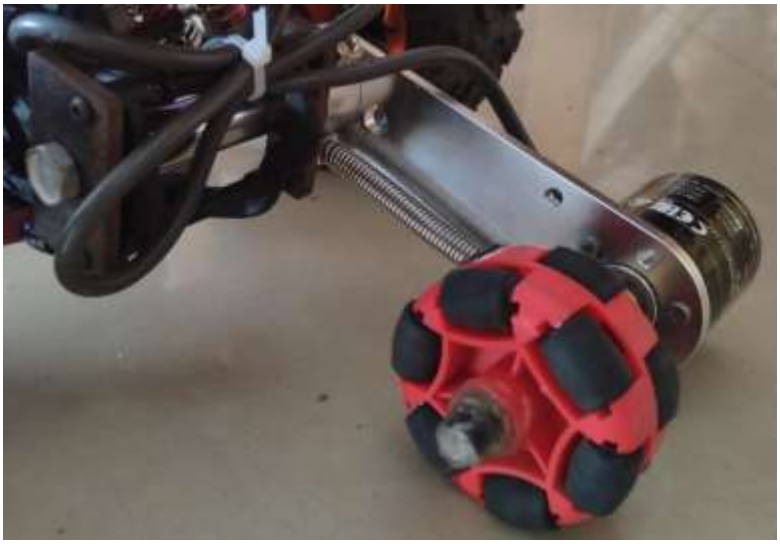


图 8.2 第五轮

编码器与第五轮通过联轴器连接，并固定在具有减震弹簧的支架上。第五轮选用 F 车的万向轮，有效减小其对转向的影响。

8.3 减震器结构改进。

G 车模是 RC 室外越野车模的一种，前后轮共装有八个减震器，调节减震器可以使车模减震性能、转向能力等发生改变。该车模的减震器在颠簸路段可以缓解路面带来的冲击，迅速吸收颠簸时产生的震动，使车辆恢复到正常行驶状态。但是对于平跑，减震器会导致不必要的倾斜，使传感器测量数据产生误差。在不更换减震的前提下，通过扎带以及尼龙柱，可以对减震器进行固定（图 8.4），消除减震器的负面影响。

在测试中我们发现，车辆前轮在转向的过程中，底部转向杆会触碰到前轮后方两个减震的底端，这导致了车辆在高速运行中转向杆左右拉伸受到阻碍，车辆的转向顺滑度会一定程度的降低。经过讨论，我们使用垫片将减震的底端抬高，使它脱离转向杆运动轨迹，车辆的转向性能有了显著的提高。（如图 8.3）。（后来我们按要求去除了轧带和螺母，但此处还是强调这样的措施以及更换更硬的弹簧可以有效的改善转向时的侧倾与加减速时的俯仰）。

图 8.3 减震器处理

8.4 防撞设计

室外光电组别要求智能车在运动过程中躲避障碍。虽然软件设计上已经很好的完成了该功能，但是由于比赛场地的随机因素太多，仍然有撞上障碍的可能性。根据规则，室外光电组别的障碍物为填满沙土的锥桶，车辆难以撞开，一旦证明撞上某障碍，就不能完赛。为了保证完赛率，我们设计了前防撞（图8.4）。

图 8.4 防撞实物图

防撞条成一定夹角可以在正面撞上障碍时，强行给车辆一个引导，使车辆朝着某一个方向继续行驶，避免了推着障碍走的情况，提高完赛率。防撞条的滑轮则是为了减小引导所需要的力。经过实践，效果理想。

第九章　单片机与 PC 端通信

9.1 单片机软件设计

车辆的底盘控制主要是以基于 Cortex-M4 内核的 K66 单片机为核心，通过串口中断接收 PC 端发送的速度指令与转向指令，利用 FTM 定时器输出相应的 PWM，从而改变电调以及舵机的脉冲信号，实现对车辆纵向速度与转向角度的控制。

对于底盘控制的核心是利用串口中断对上位机发送的信息进行存储与处理。由于车辆在高速行驶，对于实时性的要求极高，是一个强实时性系统，我们的串口中断处理方法是利用中断服务程序中对接收到的数据帧进行就地识别计算，不采用缓冲区机制，串口数据的接收与识别同时在中断服务子程序中进行，对数据帧的分析与识别在主循环中进行解码。该算法的流程图如图所示：

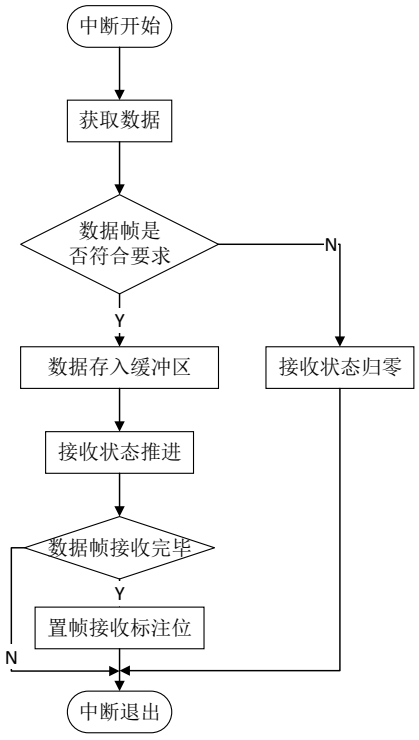


图 9.1

利用在中断程序中识别数据帧的技术最大的好处就是减轻了处理器的负担，对单片机程序的实时性有的显著的提高，其好处是：在收到数据帧之后就对数据进行识别，节省了处理器识别数据帧的时间以及管理缓冲区的工作量；对于错误的数据帧不再进行接收，无需再消耗系统资源。

串口通信采用 RS-232 串口通信协议（ANSI/EIA-232 标准）。

RS-232 串口通信参数：

(a) 波特率：RS-232-C 标准规定的数据传输速率为每秒 50、75、 100、150、300、600、1200、2400、4800、9600、19200 波特。

(b) 数据位：标准的值是 5、7 和 8 位，如何设置取决于你想传送的信息。比如，标准的 ASCII 码是 0～127（7 位）；扩展的 ASCII 码是 0～255（8 位）。

(c) 停止位：用于表示单个包的最后一位，典型的值为 1，1.5 和 2 位。由于数是在传输线上定时的，并且每一个设备有其自己的时钟，很可能在通信中两台设备间出现了小小的不同步。因此停止位不仅仅是表示传输的结束，并且提供计算机校正时钟同步的机会。

(d) 奇偶校验位：在串口通信中一种简单的检错方式。对于偶和奇校验的情况，串口会设置校验位（数据位后面的一位），用一个值确保传输的数据有偶个或者奇个逻辑高位。例如，如果数据是 011，那么对于偶校验，校验位为 0，保证逻辑高的位数是偶数个。如果是奇校验，校验位位 1，这样就有 3 个逻辑高位。

底层控制的另一个基本核心就是 PWM 输出的计算，计算结果的优良直接决定了舵机与电机的性能表现。我们采用的是最常见又比较好用的 PID 控制。

PID 控制：对偏差信号 e(t)进行比例、积分和微分运算变换后形成的一种控制规律[错误!未找到引用源。]。在工程实际中，应用最为广泛的调节器控制规律为比例、

积分、微分控制，简称 PID 控制，又称 PID 调节。PID 控制器问世至今已有近 70 年历史，它以其结构简单、稳定性好、工作可靠、调整方便而成为工业控制的主要技术之一。当被控对象的结构和参数不能完全掌握，或得不到精确的数学模型时，控制理论的其它技术难以采用时，系统控制器的结构和参数必须依靠经验和现场调试来确定，这时应用 PID 控制技术最为方便。在实际应用中，也有时根据不同应用环境采用 PI 和 PD 控制。

PID 控制器的输入输出关系为：

$$u(t) = K_p\left(e(t) + \frac{1}{T_i}\int_0^t e(t)dt + \frac{T_d de(t)}{dt}\right)$$

（公式 9.1）

（$e(t)$为输入的误差信号，Kp 为比例系数，Ti 为积分时间常数，Td 为微分时间常数，$u(t)$为控制器输出。）

在单片机中，我们仅能对数字信号处理，即数字 PID 控制。将公式 1 离散化，得

$$u(k) = K_p\left(e(k) + \frac{T}{T_i}\sum_{j=0}^{k} e(j) + \frac{T_d\,(e(k)-e(k-1))}{T}\right) = K_p \times e(k) + K_i T\sum_{j=0}^{k} e(j) + K_d\frac{e(k)-e(k-1)}{T}$$

（公式 9.2）

上式中：$Ki=Kp/Ti$，$Kd=Kp*Td$，T 为采样周期，k 为采样序号，$k=1,2,...$，$e(k-1)$和 $e(k)$分别为第（$k-1$）和第 k 时刻所得到的系统偏差信号。

增量式 PID 控制是指控制器的输出是控制量的增量 $\Delta u(k)$，当执行机构需要的是控制量的增量而不是位置量的绝对数值时，可以使用增量式 PID 控制算法进行控制。

增量式 PID 控制算法公式：

$$\Delta u(k) = u(k) - u(k-1)$$

$$\Delta u(k) = K_p(e(k)-e(k-1)) + K_i T e(k) + \frac{K_d(e(k)-2e(k-1)+e(k-2))}{T}$$

(公式 9.3)

以上各式中，*Ki=Kp/Ti*，*Kd=Kp*Td*，*T* 为采样周期，*k* 为采样序号，*k*=1,2,…，*e(k-2)*、*e(k-1)*以及 *e(k)*分别为第 *（k-2）*、第*(k-1)*和第 *k* 时刻所得到的系统偏差。

9.2 PC 端软件设计

Boost::Asio 是一个跨平台的 C++库，用于网络和底层 I/O 编程，是异步输入输出的核心。该库可以让 C++ 异步地处理数据，且平台独立。 异步数据处理就是指，任务触发后不需要等待它们完成。 相反，Boost.Asio 会在任务完成时触发一个应用。 异步任务的主要优点在于，在等待任务完成时不需要阻塞应用程序，可以去执行其它任务。

使用 Boost.Asio 进行异步数据处理的应用程序基于两个概念：I/O 服务和 I/O 对象。 I/O 服务抽象了操作系统的接口，允许第一时间进行异步数据处理，而 I/O 对象则用于初始化特定的操作。Boost.Asio 只提供了一个名为 boost::asio::io_service 的类作为 I/O 服务，它针对所支持的每一个操作系统都分别实现了优化的类。

C++的BOOST库中，通信库都在asio下，串口通信由asio组件的serial_port类完成。BOOST 库下的串口通信 serial_port 类的使用跟网络通信相似，只是在进行串口通信前需要初始化串口。

PC 端通信部份的软件设计结构如下：

- 对 serial_port 类头文件声明：

```
#include   <boost/asio.hpp>
```

- 创建串口对象：

```
boost::asio::serial_port* serial_port = 0;
```

- 传入 io_service 对象，打开串口

在 art_racecar_init(int speed,char *dev) 函数中定义了一个 I/O 服务 io_service

io_service 对象是使用 boost::asio 库的必有对象。通过 serial_port(io_service)传入 io_service 对象，再 open(dev)打开串口。一旦串口被打开，则此串口就会被当成流来被使用。如果串口端口不存在，则 try-catch 能够获取“系统找不到指定的文件”文件异常。如果串口端口没有和实际的串口连接，则 try-catch 能够获取“设备没有连接”的异常。

- 串口初始化

通过函数 set_ooption 对串口参数波特率、流量控制、奇偶校验、停止位和数据位进行设置。

- 调用 write()函数进行串口通信

write(fd_,buf,6);

如先前所述，serial_port 类成功打开串口后，串口对于 serial_port 类来说就成了一种文件流。就可以使用 boost::asio 下的函数往“流”读写数据。向串口发送数据时是采用 boost::asio 下 write()函数将程序中的完整的数据串写入到串口。write()的第一个参数表示 serial_port 对象，第二个参数是写向(传输)串口的数据，第三个参数是 boost 库下的异常参数，如果 write 函数传输数据发生什么异常就会自动抛出此异常给 catch。

第十章　结论

经过几个月的努力，我们的车达到了直道 3.1m/s 的速度，但由于激光雷达性能无法提升与弯道定位的晃动，导致代价地图和路径规划的更新频率以及弯道速度无法满足更高水平的要求。同时，处理器的性能比较一般，不便于移植更复杂的算法。其避震系统的机械结构也导致车身侧倾和俯仰时激光雷达的数据发生较大的误差，导致障碍物定位会发生移位。其转向结构及轮胎安装方式有着严重的转向虚位，限制了其转向能力，转向半径过大。因此在以下方面，仍有改进的空间：

（1） 可以通过更换激光雷达，提高数据的采集频率以及精度，提高车辆避障的实时性。

（2） 可以通过增加摄像头，增强识别障碍的能力，并给出可行域。

（3） 更换硬质弹簧改善侧倾；改良转向改善转向虚位的情况。

（4） 对于路径规划方面的研究可以进一步深入，对于规划出的路径可以考虑利用二次优化的方法使路径更加符合车辆的动力约束。

（5） 考虑如何自行判断何时需要加减速及如何更及时更恰当的加减速。

（6） 弯道处定位很晃，导致弯道处和出弯后的避障会有一定问题，应想办法解决定位晃的问题

（7） 去掉测速轮，将编码器直接与电机齿轮连接。

参考文献

[1] Mariano Jaimez, Javier G. Monroy and Javier Gonzalez-Jimenez. Planar Odometry from a Radial Laser Scanner. A Range Flow-based Approach. ICRA,2016.

[2] 朱葛峻,王盼宝,等.智能车制作:从元器件、机电系统、控制算法到完整的智能车设计.清华大学出版社,2018.

[3] O. J. Woodman, “An introduction to inertial navigation,” University of Cambridge, Computer Laboratory, Tech. Rep. UCAMCL-TR-696, vol. 14, 2007.

[4] J. Gonzalez-Jimenez and R. Gutierrez, “Direct motion estimation from a range scan sequence,” Journal of Robotic Systems, vol. 16, no. 2, pp. 73–80, 1999.

[5] A. Censi, “An ICP variant using a point-to-line metric,” in IEEE Int. Conference on Robotics and Automation (ICRA), 2008, pp. 19–25.

[6] A. Diosi and L. Kleeman, “Fast laser scan matching using polar coordinates,” The International Journal of Robotics Research, vol. 26, no. 10, pp. 1125–1153, 2007.

[7] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, “Robust Monte Carlo localization for mobile robots,” Artificial intelligence, vol. 128, no. 1, pp. 99–141, 2001.

[8] J. Huang and E. Konaka. Multiple look-ahead distance scheme for lateral control of vision-based vehicles. the 19th World Congress on Intelligent Transport Systems, AP-00106, 2012

[9] Al Sweigart,王海鹏.Python 编程快速上手：让繁琐工作自动化.人民邮电出版社,2016.

[10] Roesmann, Christoph,Feiten, Wendelin,Woesch, Thomas,Hoffmann, Frank,Bertram, Torsten. Trajectory modification considering dynamic constraints of autonomous robots[P]. Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on,2012.

[11] Rosmann, C.,Feiten, W.,Wosch, T.,Hoffmann, F.,Bertram, T.. Efficient trajectory optimization using a sparse model[P]. ,2013.

致谢

今年能够参加第十四届恩智浦杯全国大学生智能汽车赛首先要感谢队友间的相互支持和包容以及江毅骅学长前期的帮助。其次要感谢王击老师的悉心指导，黎群辉的场地支持和关怀，杨建老师、周汉斌学长、李永攀学长的帮忙寻找和借用场地还有送来的水果。然后还要感谢和太原工业、武汉大学、哈工大等学校的友好交流。最后感谢学校的资金支持。

附录

源代码

```
<launch>
    <!-- Map server -->
    <node name="map_server" pkg="map_server" type="map_server"
args="$(find csu_racecar)/map/test_amcl.yaml">
        <remap from="map" to="map_amcl"/>
    </node>
    <node name="map_server_nav" pkg="map_server"
type="map_server" args="$(find
csu_racecar)/map/test_nav.yaml"/>

    <!-- ODOMETRY -->
    <!--rf2o_Laser_Odometry-->
    <include file="$(find
csu_racecar)/launch/includes/rf2o.launch.xml" />
    <!-- Robot_Localization -->
    <node pkg="robot_localization" type="ekf_localization_node"
name="ekf_se" clear_params="true">
    <!--node pkg="robot_localization"
type="ukf_localization_node" name="ukf_se"
clear_params="true"-->
        <!--rosparam command="load" file="$(find
csu_racecar)/param/ukf_params.yaml" /-->
        <rosparam command="load" file="$(find
csu_racecar)/param/ekf_params.yaml" />
    </node>

    <!-- Localization -->
    <!-- AMCL -->
    <include file="$(find
csu_racecar)/launch/includes/amcl.launch.xml">
        <!--arg name="init_x" value="$(arg init_x)"/-->
        <!--arg name="init_y" value="$(arg init_y)"/-->
        <!--arg name="init_a" value="$(arg init_a)"/-->
    </include>
```

```
    <!-- Navstack -->
    <node pkg="move_base" type="move_base" respawn="false"
name="move_base" >
        <!-- local planner -->
        <param name="base_global_planner"
value="global_planner/GlobalPlanner"/>
        <param name="base_local_planner"
value="base_local_planner/TrajectoryPlannerROS"/>

        <!-- costmap layers -->
        <rosparam file="$(find
csu_racecar)/param/costmap_common_params.yaml" command="load"
ns="global_costmap"/>
        <rosparam file="$(find
csu_racecar)/param/costmap_common_params.yaml" command="load"
ns="local_costmap"/>
        <rosparam file="$(find
csu_racecar)/param/local_costmap_params.yaml" command="load"/>
        <rosparam file="$(find
csu_racecar)/param/global_costmap_params.yaml" command="load"/>
        <!-- move_base params -->
        <!--rosparam file="$(find
csu_racecar)/param/teb_local_planner_params.yaml"
command="load"/-->
        <rosparam file="$(find
csu_racecar)/param/base_global_planner_params.yaml"
command="load"/>
        <rosparam file="$(find
csu_racecar)/param/move_base_params.yaml" command="load"/>
        <remap from="/odom" to="/odometry/filtered"/>
        <remap from="/move_base_simple/goal"
to="/multi_point/goal"/>

    </node>

    <!-- pure_pursuit controller -->
    <node pkg="csu_racecar" type="pure_pursuit" respawn="false"
name="pure_pursuit" output="screen">
        <rosparam file="$(find
csu_racecar)/param/pure_pursuit_params.yaml" command="load"/>
```

```
        <remap from="/pure_pursuit/odom" to="/odometry/filtered"
/>
        <remap from="/pure_pursuit/global_planner"
to="/move_base/GlobalPlanner/plan" />
        <remap from="/pure_pursuit/amcl" to="/amcl_pose" />
        <remap from="/pure_pursuit/goal"
to="/move_base_simple/goal" />
    </node>

</launch>
```