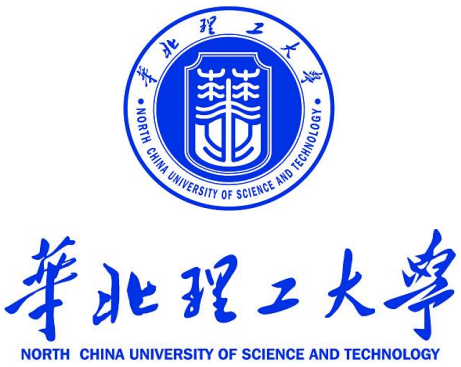
学 号：**201614420112**



毕业设计说明书

GRADUATE DESIGN

设计题目：**基于 Arduino 倒车雷达的设计与实现**

学生姓名：刘佳玮

专业班级：**16** 计科 **1** 班

学　　院：人工智能学院

指导教师：曾屹　　高级实验师

XXXX 年 **XX** 月 **XX** 日

摘　要

本文以实现汽车倒车安全性和便捷性作为设计背景，设计出一种基于超声波测距且含报警系统的倒车雷达系统，系统将测量倒车的距离并反馈给用户，最后与报警系统相结合，以增加驾驶者在倒车停车时候的安全性以及便捷性。本设计以 Arduino 为主要控制器，依照超声波测距理论，选用三组收发信息超声波传感器为测距工具，通过手机端 APP 发送指令操作车辆驾驶。当最小距离包含在危险值中时，系统将有报警提示并采取紧急制动。最后通过模拟测试，发现系统不仅能实现在水平正方向 5cm-100cm 的距离较准确测距，对于车体两侧的形状规则的大障碍物也有一定的提醒作用。本设计具有成本低，集成度高等优点，对于辅助驾驶员倒车有很好的帮助作用，因此具有一定的理论和实用价值的。

关键词 Arduino；倒车雷达；超声波测距；报警系统

Abstract

This article takes the realization of car back-up safety and convenience as a design background, and designs a back-up radar system based on ultrasonic ranging and an alarm system. The system will measure the back-up distance and feed it back to the user, and finally combine it with an alarm system to Increase driver safety and convenience when parking in reverse. This design uses Arduino as the main controller. According to the theory of ultrasonic ranging, three sets of ultrasonic sensors for transmitting and receiving information are selected as the ranging tools, and the mobile phone terminal APP sends instructions to operate the vehicle driving. When the minimum distance is included in the dangerous value, the system will give an alarm and take emergency braking. Finally, through simulation tests, it was found that the system can not only achieve a more accurate distance measurement in the horizontal positive direction from 5cm to 100cm, but also have a certain reminding effect on large obstacles with regular shapes on both sides of the vehicle body. The design has the advantages of low cost, high integration, etc., and has a good help function to assist the driver in reverse, so it has certain theoretical and practical value.

Keywords Arduino; parking sensor; ultrasonic ranging; alarm system

目　　录

1 绪论.....4
1.1 ARDUINO 的发展与使用情况阐述.....4
1.1.1 Arduino 现状研究.....4
1.1.2 Arduino 的可扩展性研究.....4
1.1.3 外设的可利用性研究.....5
1.2 倒车雷达的现状研究.....5
1.2.1 驾驶倒车现状.....5
1.2.2 倒车雷达现状.....5
1.3.本课题的来源.....6
1.4 本章小结.....6
2 倒车雷达系统设计.....7
2.1 硬件结构.....7
2.1.1 超声波模块.....7
2.1.2 直流电机模块.....8
2.1.3 电机驱动模块.....10
2.1.4 蓝牙控制模块.....11
2.2 本章小结.....12
3 倒车雷达软件设计.....13
3.1 系统可行性分析.....13
3.2 系统需求分析.....13
3.3.1 倒车雷达系统运行流程图.....13
3.3.2 系统功能模块的划分.....15
3.4 倒车阈值参考标准.....15
3.5 系统详细设计.....15
3.5.1 车辆行驶功能设计.....15
3.5.2 车辆减速功能设计.....17
3.5.3 感知功能设计.....18
3.5.4 制动功能设计.....19
3.5.5 通信功能设计.....20
3.6 倒车雷达系统功能实现.....22
3.7 本章小结.....22
4 结　论.....23
5 谢　辞.....24
6 参考文献.....25
7 附　录.....26

第 1 章 绪论

1.1 Arduino 的发展与使用情况阐述

1.1.1 Arduino 现状研究

Arduino 诞生于 2005 年，是一款包含以 AVR（Advanced Virtual RISC）单片机为核心控制器的单片机应用开发板和软件 Arduino IDE（Integrated Development Environment）的便捷灵活、方便上手的开源电子原型平台。Arduino 开发人员提供的简洁的 Arduino IDE 是一款十分方便用于写代码、编译、调试和下载的上位机软件，可以在 Windows、Macintosh OS X、Linux 三大主流操作系统上运行，所以 Arduino 可以实现跨平台开发。同时，Arduino 开发人员开发了简单的函数和众多的应用库，在不用直接操作寄存器的条件下，使得没有很好的单片机基础的人员也可以使用 Arduino 完成快速开发。

由于硬件电路图开源、编写程序语句简洁、编程调试环境便捷、库函数丰富的原因，原型开发平台 Arduino 被广大设计者应用于快速完成原型开发。因为 Arduino 的种种优势，越来越多的软件开发者使用 Arduino 进入硬件、物联网等开发领域来开发相关的项目、产品。

1.1.2 Arduino 的可扩展性研究

Arduino 采用的Creative Commons许可证，不需要付版权费用即可在原开发板上完成重新设计与二次开发。此外AVR处理器采用了RISC（Reduced Instruction Set Computer）技术，这一系列的处理器都是集成到单个芯片的独立的计算机。对于任何一种小型的控制或监视应用来说ACR处理器都是理想的选择。它们包含有一组内置的片上外设，还可以在片外扩充附加功能，所以价格相对低廉划算。

Arduino具有丰富的接口，可以利用多种通信机制对周边I/O设备进行编程进而完成逻辑控制，且很多常用的I/O设备都已经带有库文件或者样例程序，在此基础上进行简单的修改，即可编写出比较复杂的程序，完成功能多样化的作品。如可以快速使用Arduino与Adobe Flash，Processing，MaxMSP，SuperCollider等软件结合[1]，做出互动作品，也可以使用现有的电子元件例如：开关、传感器、其他控制器件、发光二极管、步进马达、其他输出装置，开发出更多令人赞叹的作品。考虑到Arduino具有低成本可易开发的优势，Arduino会有广阔的市场和受用人群。

1.1.3 外设的可利用性研究

在电子技术飞速发展的今天，积累了众多性能优良、功能众多的模块且配备了丰富的相关文档。而 Arduino 可以很好的和这些模块完成交互，如温湿度检测、距离判断、光检测和烟雾检测等，不必为了适配 Arduino 开发板而去重新开发模块，因为重新开发已有的子模块只会增加多余工作量完成重复的工作，提升了开发成本。而 Arduino 可以很好的利用已有模块，借助前人的智慧快速完成产品原型的开发[2]，因此系统开发的重点转变为软件开发，即在 Arduino 的控制下，各个模块之间的逻辑控制、功能实现与数据交互。

1.2 倒车雷达的现状研究

1.2.1 驾驶倒车现状

随着司机和车辆数量的迅速增长，道路变得越来越拥挤，停车条件也随之越来越困难。司机开车的难度更是增加很多，尤其是在特定的环境里，如黑夜、道路拥挤、大雾和地面环境恶劣的场所。坐在车上的司机不能及时和全面的了解汽车外部面临的各种情况，尽管所有的车辆都配备了后视镜装备，但并不能完全解决这方面的问题。逆向停车是一个复杂的系统，操作行为成功不仅取决于司机成熟的驾驶经验和技能，更取决于司机判断的灵敏度。但倒车雷达大大降低了驾驶员的体力和脑力消耗，逆向停车的司机，只需要坐在驾驶座上，就可以了解车辆周围的环境，为在视线受阻、地面环境恶劣的情况下倒车停车，提供了很大的安全性，从而提高行车安全，降低司机很多安全隐患。

1.2.2 倒车雷达现状

倒车雷达发展至今经历了五个阶段[3]，第一阶段为开始倒车时以大分贝喇叭提示行人注意避让；第二阶段的产品使用蜂鸣器进行警报提示；当于障碍物距离很近时发出警报；第三阶段的产品借助数码管显示与障碍物的距离；第四阶段的产品借助液晶屏显示与障碍物的距离，第五阶段的产品借助超声测距，提高了对障碍物距离判断的准确性。而现阶段将车载娱乐系统和第四阶段的液晶屏幕、第五阶段的超声测距相结合，汽车厂商为了使车主在倒车时能够更加安全、方便，于是在车载娱乐系统上集成了由倒车雷达、静态倒车辅助线、动态倒车辅助线、倒车影像、全景影像等功能组成的倒车辅助系统，但价格较为昂贵。至此，可以理解到倒车雷达是否稳定，主要取决于视频和声波两方面。

1.3. 本课题的来源

本课题来源于生活实际，用于辅助驾驶以解决倒车存在的安全隐患。如今，汽车的数目日渐增多，人们在日常生活中倒车停车的次数也是越来越多。因此，当倒车停车的时候，人们对汽车倒车停车系统的安全性和便捷性要求也逐步增加。现在的小汽车一般都配备有倒车影像，但有的车辆工作环境恶劣，使用倒车影像并不现实。即便是有倒车影像，由于影像镜头一般采取广角设计，画面会发生扭曲[4]。对于缺乏距离感的人来说，想要安全倒车依然有困难，这时配备一款超声波倒车雷达就显得很有必要。而新司机驾驶技术令人堪忧，尤其是在停车过程中，不能及时有效地处理避障情况，往往造成损害到汽车本身，甚至危及其他人的安全。汽车倒车需要可以检测驾驶员背后的驾驶育区，及车辆两侧情况，为驾驶员提供有效的提示，从而避免汽车的损坏和危险以及其他车辆的安全状况。

1.4 本章小结

随着科技发展，芯片技术已经越来越与人们的生活紧密联系，它给人们的生活带来了不少的便利。本章主要介绍了 Arduino 的开发优势：无需接触底层硬件，可利用库函数完成快速开发等；指出了需要掌握的相关技术：了解外设模块使用逻辑，掌握如何通过 Arduino 调用外设模块；最后指明了当今倒车雷达的研究现状，进而掌握本系统的设计方向，初步构想实现倒车雷达系统的关键步骤。

第 2 章 倒车雷达系统设计

2.1 硬件结构

在实现倒车雷达系统中，硬件结构将作为软件实施的载体。软件的成功运行离不开 Arduino 和各类传感器的连接、逻辑控制和分工合作，如选择直流电机模拟车辆轮胎的转动，选用蓝牙模块传输数据以完成模拟驾驶的功能等。搭建的倒车雷达系统的硬件结构如下文所示。

2.1.1 超声波模块

为使用超声波测量与障碍物的距离，选用 HC-SR04 超声波模块，本模块性能稳定，测度距离精确，能和国外的 SRF05，SRF02 等超声波测距模块相媲美，可应用于距离测量，机器人，防盗装置等。

HC-SR04 有两个超声波传输口，分别为 T 和 R。对 Trig 管脚赋予高电平 1，并保持 10us 以上再赋予低电平 0，此时会触发 HC-SR04 的开始信号，其中的 T 声波传输口会发出发送 8 个 40khz 的方波。在发送完毕后赋予 Echo 引脚高电平 1，此时另一个声波传输口 R 开始接受 T 刚发送的超声波，在接受结束后自动赋予 Echo 引脚低电平 0。此时 Echo 引脚保持高电平的时间就代表了这段超声波的往返时间[5]。因为是往返的时间，所以计算与障碍物的距离时需要减半来变成单程的时间，再将单程时间乘以声速即 340m/s，就会得到从当前位置到对面障碍物的距离。HC-SR04 的工作原理示意图如图 2.1 所示。

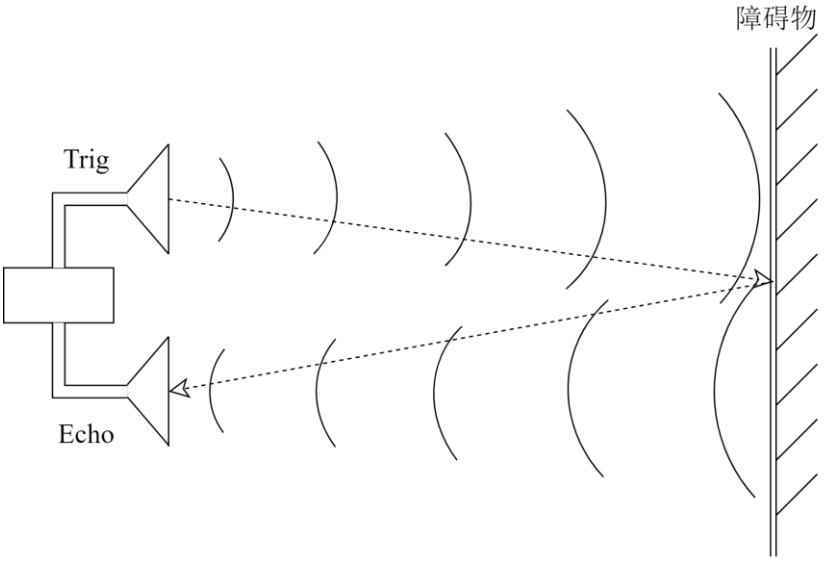


图 2.1 超声波工作原理示意图

HC-SR04 超声波模块的工作参数如表 2-1 所示：

表 2-1 超声波模块工作参数表

工作参数	参数阈值
使用电压	5V
工作电流	15 mA
感应角度	不大于 15°
探测距离	2cm 到 450cm
工作频率	40Hz

在了解完 HC-SR04 的工作原理与工作参数后，也很容易得到 Arduino 与 HC-SR04 的控制逻辑与连接方式。HC-SR04 共四个引脚，分别为：VCC，GND，Trig 和 Echo。将 HC-SR04 的 VCC 引脚连接到 Arduino 的+5V 引脚，HC-SR04 的 Trig 引脚连接到 Arduino 的 8-13 号的任意引脚作为信号的出口用于拉高电平，HC-SR04 的 Echo 引脚连接到 Arduino 的 8-13 号任意引脚作为信号的入口用于检测电平的变化，最后将 HC-SR04 的 GND 引脚连接到 Arduino 的 GND 引脚，其连接示意图如图 2.2 所示：

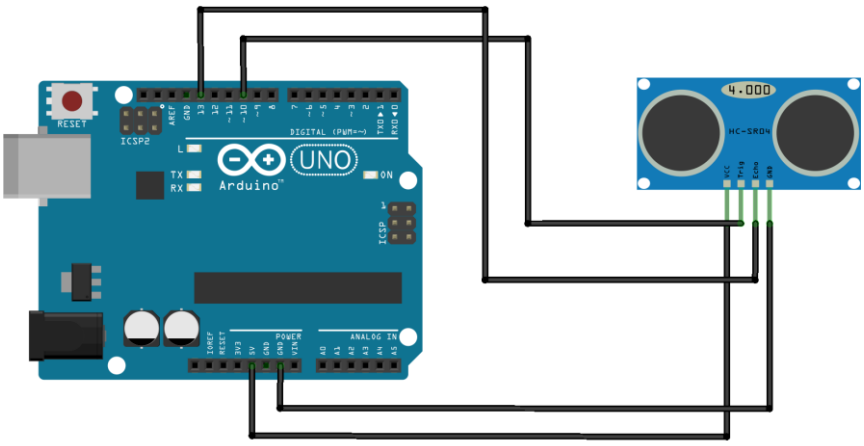


图 2.2 Arduino 与 HC-SR04 连接示意图

2.1.2 直流电机模块

直流电机是将直流电能转换为机械能的转动装置，为模拟车辆的运行，选用直流电机带动轮胎完成车辆前进、后退和转向等功能。

直流电动机的基本构造包括电枢、场磁铁、集电环和电刷。其中电枢是可以绕轴心转动的软铁芯缠绕多圈线圈；场磁铁用于产生磁场的强力永久磁铁或电磁铁；集电环是线圈约两端接至两片半圆形的集电环，随线圈转动，可供改变电流方向的变向器，每转动半圈（180 度），线圈上的电流方向就改变一次；电刷通

常使用碳制成，集电环接触固定位置的电刷，用以接至电源。直流电机结构示意图如图 2.3 所示：

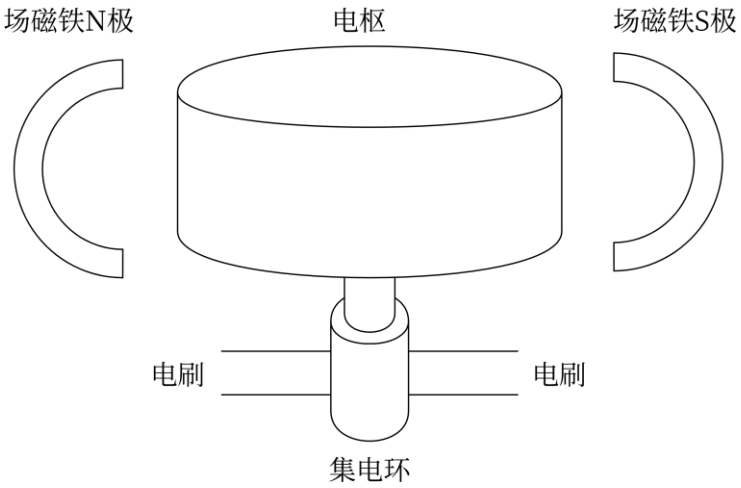


图 2.3 直流电机结构示意图

当电刷通电后，电枢周围产生磁场，带动转子的左侧被推离左侧的磁铁，并被吸引到右侧，从而产生转动。转子依靠惯性继续转动，当转子运行至水平位置时电流变换器将线圈的电流方向逆转，线圈所产生的磁场亦同时逆转，使这一过程得以重复，进而达到旋转的功能[6]。之后将电机接入轮胎，带动轮胎的旋转，完成小车的制动功能。

本次设计选用的电机型号为 HC02-48，其工作参数如表 2-2 所示：

表 2−2：直流电机工作参数表

工作参数	参数阈值
工作电压	3V 到 6V
工作电流	150mA
减速比	48:1
3V 空载转/分钟	125
5V 空载转/分钟	208
扭力	0.8kg・cm

据 HC02-48 的工作参数表明，在最小驱动电压 3V 的情况下，车速能满足对车辆完成前进和控制的要求。在根据扭力[7]进行计算，若此时直接连接半径 3cm 尺寸的轮胎，则经由车轮所发挥的推进力量约为 0.27kg 的力量，则两个电极发挥的推进力量约为 0.54kg。因此，可根据车身的重量预估所需电极的数量，在车身重量不超过 0.54kg 的情况下，选用两个电极即可。

但是 Arduino 不能直接驱动 HC02-48 电机完成工作，因为驱动电机需要较大电流，而 Arduino 的作用在于高低电平的控制而不是电流的输出。Arduino 的 IO 口输出电流在 40mA 左右，无法驱动最低要求 150mA 的电机。因此，只能采用

放大驱动的方式。但是也能得到驱动芯片的条件为驱动芯片直接驱动电机的前提是驱动芯片的输出电压和允许输出电流满足电机的要求。

2.1.3 电机驱动模块

经过不断的比对参数筛选驱动芯片，最终选用的电机驱动模块为 L298N 模块，共四个入口引脚，四个出口引脚，可同时控制两个电极。其工作参数如表 2-4 所示：

表 2−4：L298N 工作参数表

工作参数	参数阈值
逻辑部分输入电压	6.5V 到 12 V
驱动部分输入电压	4.8V 到 35V
逻辑部分工作电流	小于等于 36mA
驱动部分工作电流	小于等于 2A
控制信号输入电平	2.3V 到 5V
控制信号输入电平	-0.3V 到 1.5V
工作温度	-25℃到 130℃

在表 2-4 中，可以了解到逻辑部分输入口的输入电压最低为 6.5V，而 Arduino 的供电电压为 5V，因此需要增加额外的电池完成对 L298N 模块逻辑部分的供电功能，而用 Arduino 自身的供电口即可提供驱动部分的电压输入。在 L298N 的工作参数列表中可以了解到，在 L298N 驱动直流电机时可选择任何额定电压在 4.8~35V 和电流小于 2A 范围内的电机。而 HC02-48 电机的工作电压在 3-6V，且 L298N 的输出电流恰好能驱动电机模块，因此在硬件层面可完成驱动。

将 L298N 的四个入口引脚 IN1，IN2，IN3 和 IN4 接控制电平，而 L298N 模块的出口引脚中标明了正负极，此时 Out1，Out2 连接一个电机，Out3，Out4 出口引脚连接另外的电机。通过对 4 个入口引脚的赋值来更改出口引脚的电平输出，进而控制电机的正反转，完成前进、后退和转向的功能。此外可以通过总开关 ENA，ENB 两个使能端控制两个电机能否转动。使能端，IN1，IN2，IN3 和 IN4 的赋值情况与车辆运行情况的真值表如表 1-3 所示：

表 1−3：车轮转动真值表

直流电机状态	ENA/ENB	IN1/IN2	IN3/IN4
停止	0	X	X
制动	1	0	0
正转	1	1	0
反转	1	0	1
制动	1	1	1

在得到了可以使用 L298N 驱动电机的结论后，根据 L298N 与 Arduino 的连接方式，进而得到了 Arduino 通过 L298N 驱动电机模块的连接的电路原理图如图 2.4 所示

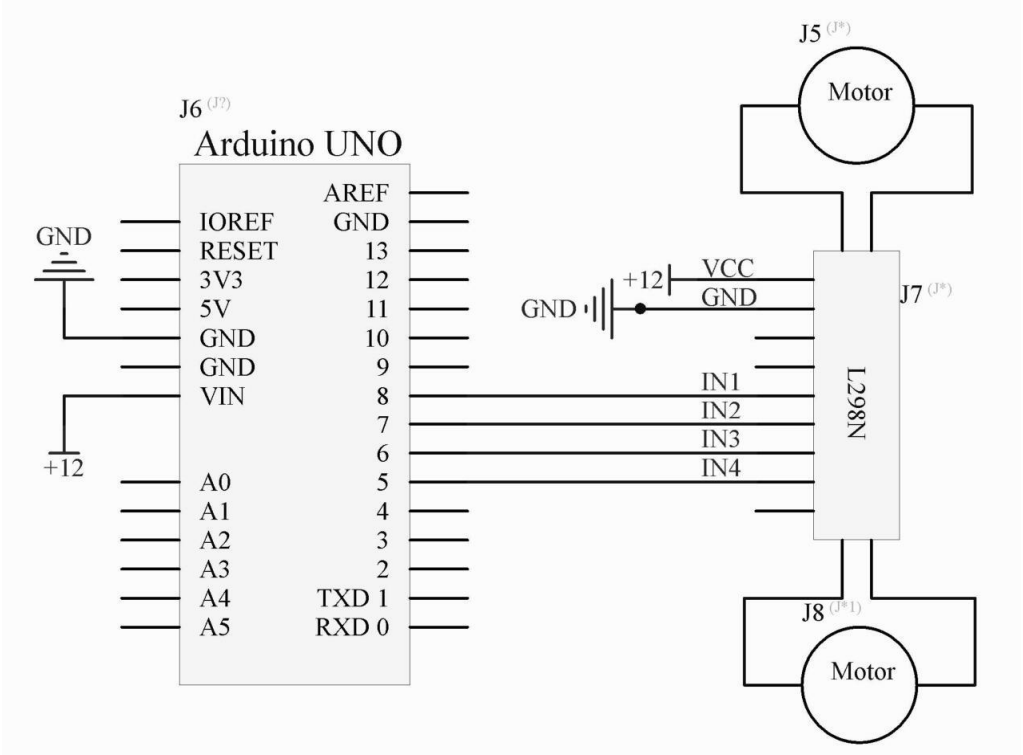


图 2.4 Arduino 驱动电机模块电路原理图

而电机的驱动需要分别给电机的两个电刷赋予高低两个不同的电平，以实现对电机的旋转。若不借助驱动芯片，一个电机会占用 Arduino 的两个 IO 口，两个电极会占用 4 个 IO 口。而一个 L298N 驱动芯片需要 4 个 IO 口的接入实现对两个电机的控制，因此尽管额外接入了 L298N 模块，但并没有造成 IO 口的浪费。

2.1.4 蓝牙控制模块

为合理模拟车辆的运行来测试倒车雷达的功能是否稳定，选择通过手机端发送指令控制车辆的前进、后退和转向功能，以此来模拟驾驶员驾驶车辆的行为，为此选用 HC-05 蓝牙模块。

HC-05 模块的主要引脚为：KEY 使能端、VCC 供电口、GND 地线、RXD 数据入口和 TXD 数据出口，并且主要通过 RXD、TXD 两引脚完成通信功能。且 HC-05 蓝牙模块的工作灵敏度为-85dBm@2Mbps。无线传输的接收灵敏度决定了设备能否良好的接收数据，提高信号的接收灵敏度可使无线产品具有更强地捕获弱信号的能力。随着传输距离的增加，接收信号变弱，高灵敏度的无线产品仍可以接收数据，维持稳定连接，大幅提高传输距离。市面上普通产品的接收灵

敏度一般为-85dBm，即 HC-05 能保证在短距离内以较低的误码率传输数据。

HC-05 蓝牙模块的工作参数如表 2-5 所示：

表 2−5：HC−05 工作参数表

工作参数	参数阈值
波特率	9600
通信距离	10 米
工作电压	3.2-6V
工作频段	2.4G
空中速率	2Mbps
工作电流	40mA
接收灵敏度	-85dBm@2Mbps

将 HC-05 蓝牙模块的 RXD 端连接 Arduino 的 TXD 端，将 HC-05 蓝牙模块的 TXD 端连接 Arduino 的 RXD 端。如此连接，会使得 HC-05 蓝牙模块的数据输出口与 Arduino 的数据读入口相连接，输出读入口与 Arduino 的数据输出口相连接，也是顺利完成两者通信的重要前提。连接图如图 2.5 所示

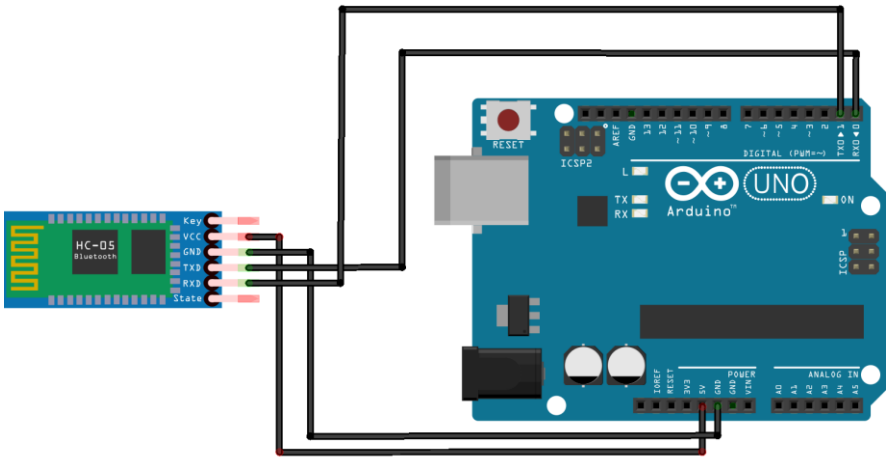


图 2.5 Arduino 与 HC-05 连接示意图

2. 2 本章小结

在本次倒车雷达系统的硬件设计中，选用 HC-SR04 超声波模块用于测量障碍为与车辆之间的距离；选用 HC-05 蓝牙模块并借助计算机中断技术完成手机端对车辆的控制，模拟驾驶员的实际驾驶情况；选用 L298N 电机驱动模块实现对直流电机的控制，进而完成车辆的前进、后退和转向功能。并给出相关电路原理图和器件的连接方式，为实现打车雷达系统的软件设计做好基础。

第 3 章 倒车雷达软件设计

3.1 系统可行性分析

系统的稳定运行离不开软件的设计，而本次设计的软件开发采用了 C++语言，具有丰富的库函数和面向对象编程的优势[8]，可以简洁快速的完成程序开发工作。此外，选用的模块具有丰富的文档，合理利用文档便可掌握如何使用模块及发挥功能，并结合 MCU 编程技术，可完成 Arduino 控制小车与衔接模块的功能。而本次系统的难点为蓝牙通信和危急情况的紧急制动，通信需要融入计算机网络中的通信技术，而紧急制动离不开计算机中断技术，合理利用通信和中断功能会对倒车雷达系统加以完善。

此外利用网络资源查询了较多 C++编程和 Arduino 控制相关技术资料，可顺利完成 MCU 的编程部分[9]；此外利用图书馆资源查询了微机接口、计算机网络通信和计算机原理相关书籍，对计算机网络和计算机原理的理论知识有所巩固，可以顺利突破设计中的难点。综上所述，本次设计具有可行性。

3.2 系统需求分析

倒车雷达系统是提高倒车安全性的必要措施。而出于系统的安全性的考虑，除在车辆尾部加装超声波模块外，也将在车辆的两侧加装超声波模块。因倒车过程中车辆两侧的安全性同样需要注意，所以在三个方位增加超声波提高倒车的安全性与便捷性。

而为了适应驾驶新手，系统将提供两种功能。第一种为辅助功能，驾驶员手动倒车，倒车雷达负责将距离信息反馈给驾驶员，辅助驾驶员根据提供的距离信息完成倒车功能；第二种为自动功能，即驾驶员可以手动选择自动倒车功能，车辆在接受指令后，开始根据超声波模块感知距离信息，而后车辆根据雷达反馈的信息自动完成倒车功能。

而无论选择哪种功能，均需要手动确定是否进入倒车模式和倒车是否结束或选择继续驾驶。

3.3 系统运行流程图

3.3.1 倒车雷达系统运行流程图

根据系统功能分析，得到系统运行的流程图，如图 3.1 所示：

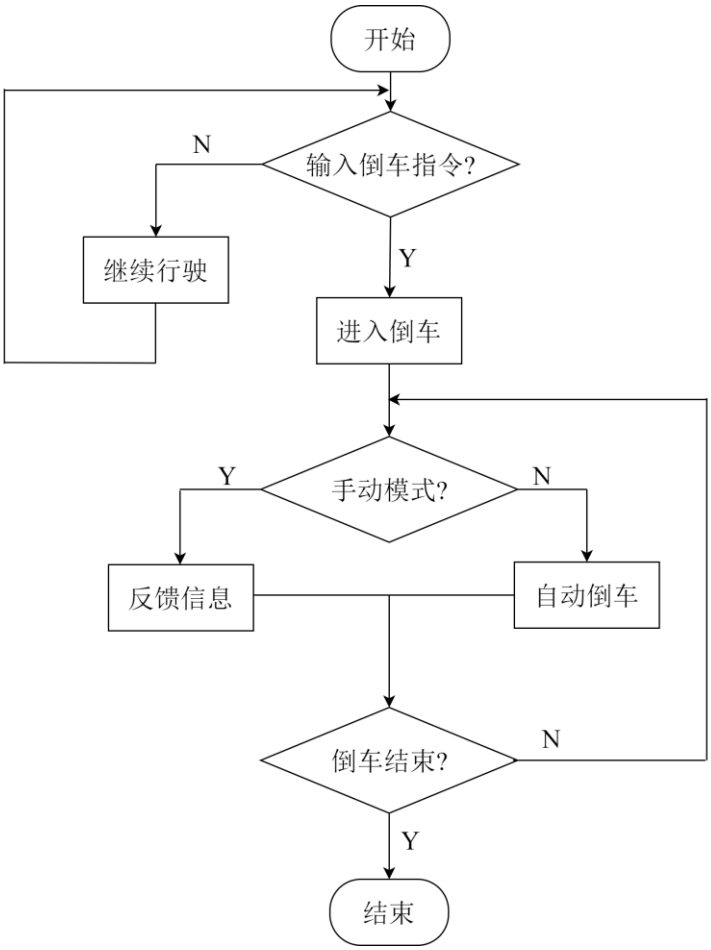


图 3.1 系统运行流程图

同时可以得到车辆紧急制动时的运行流程图，如图 3.2 所示：

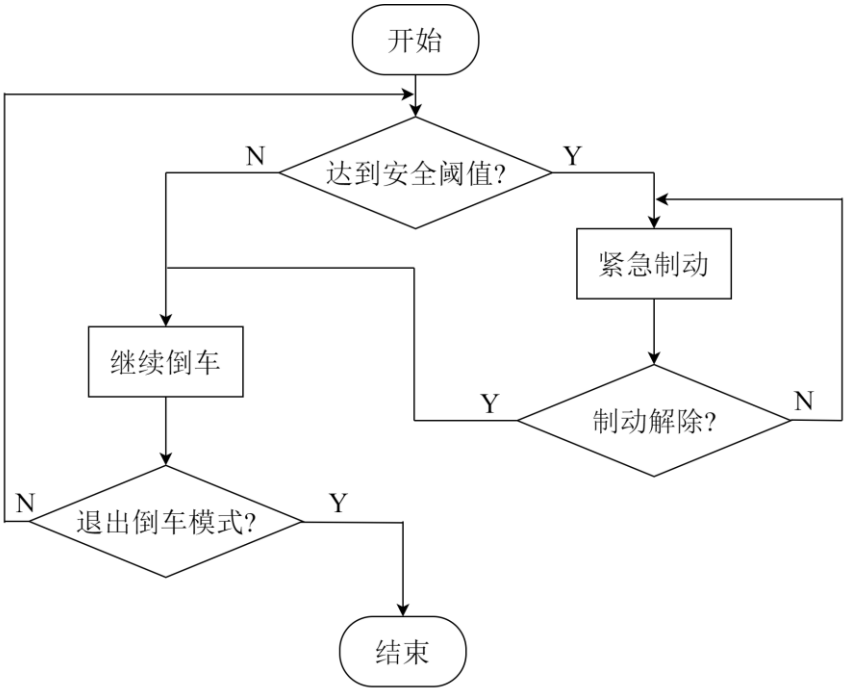


图 3.2 系统制动流程图

如图 3.2 所示，在倒车开始时，由超声波测量距离并给予驾驶员提示，辅助或自动完成倒车行为，而在测量到距离障碍物的距离过紧到达危险阈值后，由系统强行制动保证车辆和人身安全。而驾驶员意识到危险后，可手动解除制动状态并重新倒车。在倒车完毕后选择退出倒车模式，此时系统将关闭倒车雷达的超声波模块和制动中断等功能，降低系统功耗，节约资源保护环境。

3.3.2 系统功能模块的划分

将功能全部的功能划分为几个功能模块，有利于提升解决专一问题的效率。因此本次倒车雷达系统的功能模块划分为：感知功能模块、通信功能模块和控制功能模块。其中，感知功能模块由超声波构成，负责探测距离外部障碍物的距离，用于给驾驶员提供辅助信息。通信功能模块由蓝牙模块和手机端 APP（Application Program）组成，由手机端 APP 发送数据到蓝牙，蓝牙控制车辆完成相应指令。控制功能模块由 Arduino 构成，负责系统整体的运行，根据手机 APP 端发送的指令进入倒车模式、控制车辆运行、紧急制动或退出倒车模式。倒车雷达系统的功能模块划分图如图 3.3 所示：

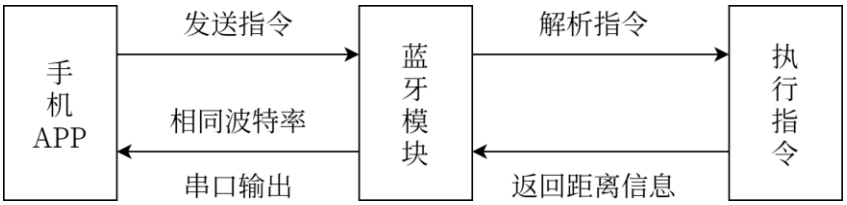


图 3.3 功能模块划分图

3.4 倒车阈值参考标准

倒车时的行驶速度会远远低于正常驾驶的速度，因此在进入倒车模式时会利用 PWM（Pulse width modulation）将车辆的行驶速度调制到低档；与此同时，当车身距离障碍物越近则认为越危险，当达到某危险阈值时，系统应立刻紧急制动，停止此时的倒车行为。经查阅文献，将实体车辆的车速和安全距离阈值映射到模型车辆，得到安全的倒车车速[10]为 2cm/s，距离障碍物的安全阈值[11]为 1cm。

3.5 系统详细设计

3.5.1 车辆行驶功能设计

倒车雷达系统中的基础为驾驶员驾驶车辆进行前进、后退、转向等移动功能。因此，首先需要完成 Arduino 控制直流电机实现正反转的设计。从物理角度出发，

当两个车轮同时向前旋转时，车身会前进；当两个车轮同时向后旋转时，车身会后退；当左侧车轮向后转动，右侧车轮向前转动时，此时车身将向左转向；同理，可以得到车身向右转向时，左侧车轮向前移动，右侧车轮将向后移动，其中又转向的示意图如图 3.4 所示。

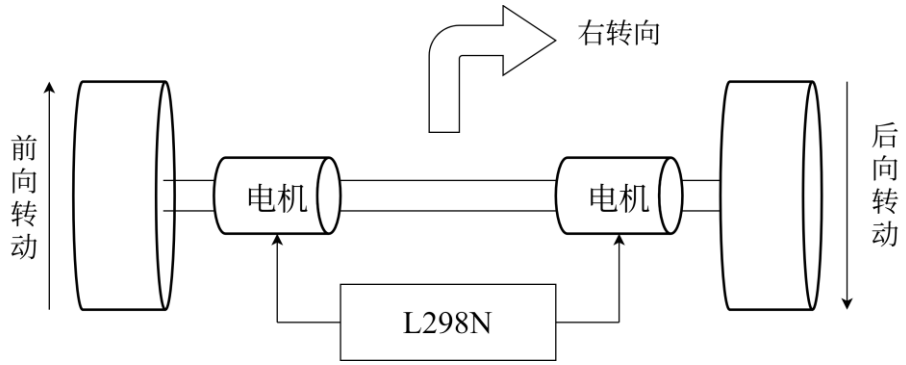


图 3.4 右转向示意图

而在电路层面，实现车轮的前向转动与后向转动的方式为更改电机两端电刷的正负极即可。最终对应到软件层面，更改车轮前后向转动的方式为更改 L298N 的出口引脚的电平。将 Arduino 的四个引脚 5，6，9，10 接入 L298N 的四个入口引脚，并将 5，6，9 和 10 这四个 IO 口定义为输出模式，即赋值高低电平输后出到 L298N 模块，由 L298N 模块的入口引脚读入指令并执行动作。对于两个电极 M1 和 M2，如 L298N 的出口引脚 Out1 和 Out2 分别接入 M1 电机的正极和负极，另外的出口引脚 Out3 和 Out4 接入 M2 电机的正极和负极。则实现右转的伪代码如下：

```
pinMode(5, OUTPUT); //初始化各 IO,模式为 OUTPUT  输出模式
pinMode(6, OUTPUT);
pinMode(9, OUTPUT);
pinMode(10, OUTPUT);
digitalWrite(input1, LOW); //  左轮前转动
digitalWrite(input2, HIGH);
digitalWrite(input3, HIGH); //  右轮后转
digitalWrite(input4, LOW);
delay(1000);
```

在代码中，可以明确看到先由 Arduino 的库函数 pinMode 声明引脚的逻辑为输出模式，即向外部的 L298N 发送数据。再由库函数 digitalWrite 向引脚发送高低电平，其中 LOW 的含义为低电平，HIGH 的含义为高电平，电流会由高电平一侧流向低电平一侧，以此来带动电机旋转。而 L298N 将输入引脚的高低电平信号进行处理，并在输出引脚进行输出。而电机按照 L298N 模块的输出进行转

动，以此来带动车身的运行。

3.5.2 车辆减速功能设计

而在倒车过程中，最重要的为减速慢性。因此需要设置车辆减速慢行的功能模块。而电机运行的快慢与提供的电压值大小有关，但是 L298N 的输出电压是不变的，因此需要使用脉宽调制技术来削弱 L298N 的输出电压。

脉宽调制技术是最常用的调速的工具，脉冲宽度调制（PWM）是一种对模拟信号电平进行数字编码的方法[12]。由于计算机不能输出模拟电压，且 L298N 的供电口只能输出固定大小的数字电压值。PWM 的控制方式是对逆变电路开关器件的通断进行控制，使用更改方波的占空比的调制方法使输出端得到一系列幅值相等的脉冲，用这些脉冲来代替正弦波或所需要的波形，达到对一个具体模拟信号的电平进行编码的目的。即在输出波形的一个周期中按一定的规则对各脉冲的宽度进行调制，即可改变逆变电路输出电压的大小，也可改变输出频率。但 PWM 信号仍然是数字的，因为在给定的任何时刻，满幅值的直流供电只可能是最大值（供电状态）或者 0（电源关闭）。

而在软件设计中使用时，仅了解需要输出电压的大小即可设置 PWM 值。以高电平输出 5V 为例，在输出 5V 电压时，只需一直输出高电平即可；输出 2V 电压时，需要在一个周期内（一个高电平和一个低电平为一个周期）保证有 20%时间输出高电平，80%时间输出低电平；同理，在输出 2.5V 电压时，仅需要在一个周期内 50%时间输出高电平，50%时间输出低电平即可。不同电压的方波占空比示意图如图 3.5 所示

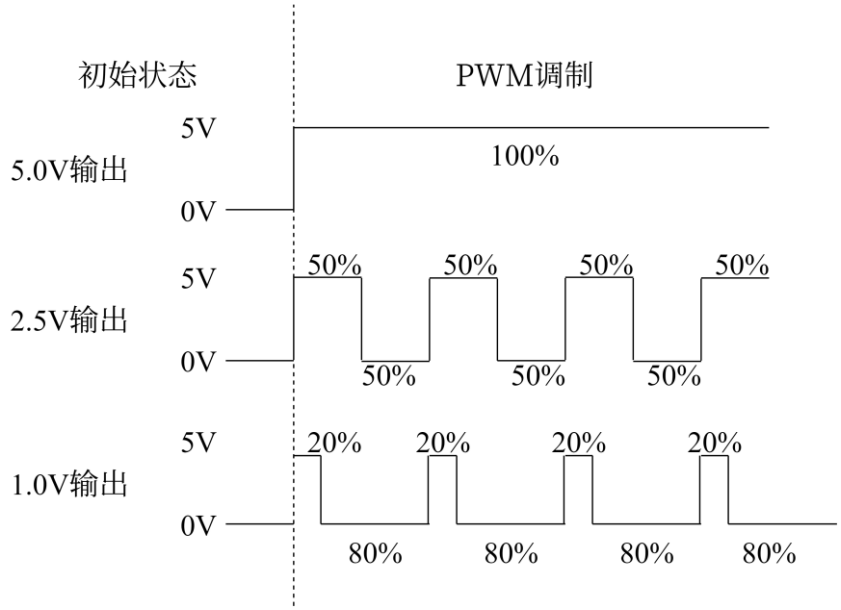


图 3.5 不同输出电压占空比示意图

Arduino 提供了 5 个 PWM 调制 IO 口，分别为 3，5，6，10 和 11。在得到对 PWM 的客观认知后，同样可以借助 Arduino 的库函数简易实现电压的调制输出。首先将标号为 5 和 6 的口作为输出口，伪代码如下所示：

```
int leftPWM = 5;
int rightPWM = 6;
pinMode(leftPWM, OUTPUT);
pinMode(rightPWM, OUTPUT);
```

而借助 Arduino 的 analogWrite()函数，第一个参数传入要调制的引脚，第二个参数为占空比，取值范围为[0,255]，0 对应 OFF，255 对应 ON，则数值 204 表示 80%的占空比，即 5V 为最高电压时输出 4V 的电压。为代码如下所示：

```
analogWrite(leftPWM, 204);
analogWrite(rightPWM, 204);
```

3.5.3 感知功能设计

感知功能主要由三组超声波模块模块，负责在倒车过程中探测左侧、右侧和后侧的距离，并将距离信息反馈给驾驶员。

为了提升安全系数，将距离声明为 float 类型的变量把距离精度提升到小数点后一位。在超声波模块的软件设计中，首先连接 Arduino 的 VCC 端和 HC-SR04 的 VCC 端，并连接两者的 GND 端构成电路通路，并将 Echo 和 Trig 引脚连接至 Arduino。在上一章节的硬件设计中，了解到 HC-SR04 的工作模式为：将 Trig 赋予高电平 10us 以上后将 Trig 赋予低电平，之后赋予 Echo 高电平并开始计时，Echo 接收到反馈声波，变为低电平时结束计时。因 Trig 引脚向外发送数据所以定义为输出模式，Echo 引脚接受数据定义为接收模式。通过时间计算与障碍物的距离，伪代码如下所示：

```
pinMode(trigPin, OUTPUT);                // 定义 Trig 为输出模式
pinMode(echoPin, INPUT);                 // 定义 Echo 为输入模式
digitalWrite(trigPin, LOW);              // 初始化 Trig 引脚
delayMicroseconds(5);
digitalWrite(trigPin, HIGH);             // 拉高 Trig 电平
delayMicroseconds(10);                   // 持续 10ms
digitalWrite(trigPin, LOW);              // 拉底 Trig 电平
duration = pulseIn(echoPin, HIGH);       // 记录 Echo 端接受声波的时间
cm = (duration/2) / 29.1;                // 将时间转换为距离
```

3.5.4 制动功能设计

中断适合执行需要不断检查的工作，如检查一个引脚上连接的按键开关是否被按下，检查一个引脚的电平是否变化。因此，也可使用中断来检测三组超声波中的任何一个探测到的距离是否低于安全阈值，而后采取紧急制动。当达到安全阈值时，中断函数应立刻发出中断相应，执行对应的中断服务程序，如暂停车辆的运行，等待驾驶员的反馈调整。而单重中断仅能相应单次中断，不能嵌套中断导致程序扩展性不强[13]。因此，考虑到未来系统可能会扩展其他功能，应使用多重中断。多重中断与单重中断的对比如表 3-1 所示：

表 3-1：单重中断与多重中断对比表

执行指令	单重中断	多重中断
中断隐指令	关中断	关中断
	保存断点	保存断点
	送中断向量	送中断向量
中断服务程序	保护现场	保护现场和屏蔽字
	--	开中断
	执行中断服务程序	执行中断服务程序
	--	关中断
	恢复现场	恢复现场和屏蔽字
	开中断	开中断
	中断返回	中断返回

Arduino 有两种中断方式，方式一为定时器中断，即设定一个时间限制，当到达时间后自动触发中断，开始执行中断函数方式。方式二为外部中断，由外部事件引起系统中断。这种方案事先编写中断捕获程序，如电平检测，当电平变化时自动触发中断，开始执行中断服务程序。因为倒车是连续的过程，并非可按离散的时间踩点判断是否制动，应该为到达临界的安全阈值时立刻触发系统中断执行对应函数，所以紧急制动应采取第二种中断方式。

无路哪种中断方式，最终目的均为执行中断服务程序，即中断 Arduino 当前正在执行的程序而优先去执行中断服务程序。当中断服务程序完成以后，再返回断点继续执行刚才执行的程序。对于 Arduino 开发板来说，中断服务程序是一种特殊的函数，具有其它类型函数所不具备的限制和特点。如不能有任何参数，且没有任何返回值，因此中断服务程序函数里所使用的变量应声明为 volatile 类型的全局变量以代替传参和返回值的功能。且中断服务程序执行期间，程序会进入到对应子函数中，其他函数会暂停执行也为重要特点。包括主函数在内的其他函数中的 delay()函数不会工作且 millis()函数返回值也不再增长，接收到的串口数据也可能丢失。

Arduino提供的中断函数attachInterrupt可实现中断触发与进入中断服务程序的功能。但 attachInterrupt 函数只能检测电平和脉冲变化引起的中断，因此，需要将距离与电平的高低相关联。如将 2 号引脚初始为高电平，若在每次捕获完距离并发现距离低于安全阈值时，将 2 号引脚改为低电平以触发中断，进而进入中断服务程序引发车辆制动。伪代码如下所示：

```
// 当与障碍物的距离低于安全距离时触发
if (distance_back < threshold || distance_left < threshold || distance_right < thershold){
    // 当引脚电平发生变化时触发中断服务程序
    attachInterrupt(digitalPinToInterrupt(interruptPin), stop, CHANGE);
}
//中断服务程序 停止车辆运行
void stop() {
    car.stop();
}
```

3.5.5 通信功能设计

通信功能主要由蓝牙模块实现。在模拟驾驶员驾驶车辆时，选取的模式为手机 APP 端连接蓝牙，发送指令控制车辆的移动。在将蓝牙模块与 Arduino 按第二章的方案正确连接后，即可完成通信功能，其示意图如图 3.6 所示：

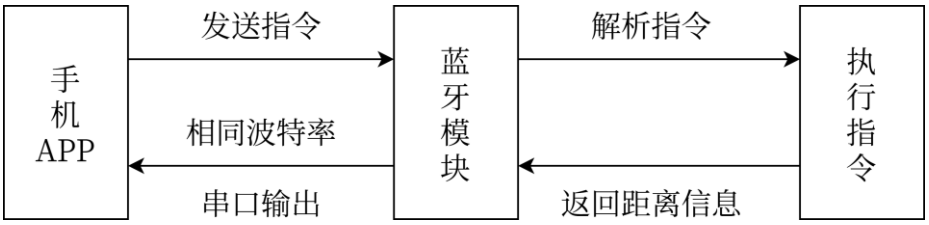


图 3.6 通信功能示意图

如图所示，蓝牙模块在处于数据通信的交通枢纽，位于倒车雷达系统的中央位置，负责手机端与车辆的数据通信。而在这个过程中，通信的网络协议可保证数据以透明形式在发送端与接收端进行。以手机 APP 发送数据为例，由应用层规定传输数据的格式，经由传输层形成 UDP（User Datagram Protocol）或 TCP（Transmission Control Protocol）报文经过端口传输至网络层，因连接好蓝牙后可捕获蓝牙模块的 MAC（Media Access Control Address）地址且默认在同一子网内，通过 CSMACD（Carrier-sense multiple access with collision detection）广播协议[14]将数据帧通过数据链路层进行发送，且数据帧内含有检验信息，而后数据帧通过 2.4G 频段的物理层传播至接收端的数据链路层。接收端由数据链路层解析数据并传播至网络层，并解析物理层传播的数据，将电信号转换为具体数值，且

仅 MAC 地址相匹配才能接受发送的数据帧。经传输层到达应用层，而后处理应用层的数据。其中，数据链路层、网络层、传输层具有完备的数据检验方式和差错处理机制。如在网络层、数据链路层会在数据报文或数据帧中加入校检数据，接收端可根据校检数据判断数据是否接受正确。此外，数据链路层的 CSMACD 协议用于保证广播通信时数据的被指定接收端接收，而传输层的慢启动、拥塞控制和流量控制[15]等算法可保证数据以合理的速度进行传输，防止传输过快或过慢导致的网络拥堵或资源过剩现象。完整的计算机网络通信模型如图 3.8 所示：

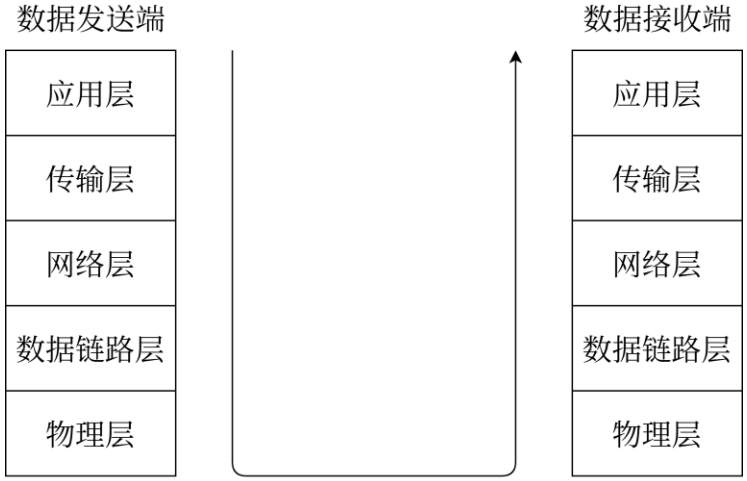


图 3.8 计算机网络通信模型示意图

而发送的指令集如表 3-2 所示：

表 3−2：发送端指令集

指令内容	指令类型	含义
F	字符型	车辆前进
B	字符型	车辆后退
S	字符型	车辆停止
L	字符型	车辆左转
R	字符型	车辆右转
1	整型	解除倒车模式
H	字符型	手动倒车
A	字符型	自动倒车

而车辆要不断获取手机端发送的指令获取下一步行动，因此应将蓝牙的通信功能作为主函数中的主循环部分，时刻接收手机端发出的指令。以 loop 函数为主循环，并检测蓝牙端是否有数据传输，伪代码如下所示：

```
void loop(){
while(Serial.available())
    char c=Serial.read();
```

而距离信息的反馈选择串口输出的形式，将倒车距离反馈给驾驶员，此时数据的发射端蓝牙模块与接收端主机需要设置同样的波特率。在计算机通信中，数据在物理层的传输均为电信号，而携带数据信息的电信号单元为码元，一个码元表示一个脉冲信号，波特率表示的为每秒钟传送的码元符号的个数。若发射端和接收端的波特率不同，两者会工作在不同的带宽中，且接收端将无法正确的对发送端的波形进行解调，将无法收到数据。为适配 HC-05 蓝牙模块，将接收端的波特率指定为 9600。

3.6 倒车雷达系统功能实现

在详细设计完各个模块的功能后，将各模块的功能加以组合运用，便能实现最终的倒车雷达系统。

对于选择手动倒车的用户，系统将开启制动功能，同时感知功能模块开始运转，将左侧、右侧和后侧的距离信息通过通信功能模块反馈给用户，对用户的倒车起到一定辅助作用。对于选择自动驾驶的用户，启动将通过距离感知模块自动探测左侧、右侧、后侧障碍物的距离。当后侧空旷而左右狭窄时，车辆将直行倒入；当左侧距离狭窄时，车辆将执行右转行为以矫正车身姿势到左右两侧距离相等的状态，利于后续的倒车行为。同理，当右侧距离狭窄时将采取左转行为以矫正车身。

而无路那种选择，当距离感知模块探测到与障碍物的距离小于安全距离时，车辆将采取紧急制动而非继续前进。而制动的另一情况为倒车结束，此时车尾与障碍物距离过近，驾驶员可选择结束自动倒车以继续驾驶或确认停止倒车，倒车正式结束。

3.7 本章小结

本章主要介绍倒车雷达系统的需求分析和设计过程，分为倒车雷达系统的可行性分析、需求分析，提出倒车雷达系统详细设计和最终实现三个部分。首先进行了系统可行性的分析，确认倒车雷达系统的设计具有可行性。而后分析倒车雷达系统的功能，做出需求分析。针对需求分析的结论，将倒车雷达系统的实现划分为几个模块，包含感知模块、通信模块和控制模块。之后对每个模块下的功能实现进行了探讨和设计，包括感知模块的距离探测功能，通信模块的数据传输功能，控制模块的制动、调速和行驶功能。最终将多个模块进行组合，实现了最终的倒车雷达系统的全部功能。

结　论

对系统的运行进行模拟实验，发现具有良好的性能。能够准确的接收手机端发送的指令并按指令执行。在倒车期间能及时减速，精准的测量各个方位的障碍物，且遇到危险情况时及时发出警报。

本次设计并实现了一种车辆的倒车雷达系统，此系统不仅可以完成车后方物体的探测，还可以对大型车辆两侧盲区探测。在整个过程中，首先研究超声测距的原理和相关技术，在此基础上根据倒车操作时的具体需要而选定了其他功能与模块，包括数据传输功能、PWM调速功能和制动报警功能，最终设计出了基于Arduino控制系统的整体设计方案。系统采用了HC-05蓝牙模块用于收发数据，该装置不但可以显示数字还可以显示字符，能够满足系统发送数据与接收数据的需求，以给驾驶员更多的倒车提示。为了提高系统的安全性，系统还加入了PWM调速功能，用于降低车辆倒车时的车速，且不断检测与障碍物的距离，能及时触发报警系统采取紧急制动，更加有利于保障人身安全。

通过对倒车雷达系统的研究和设计，本课题主要完成的工作有：（1）根据系统需求提出了系统的整体设计方案，同时确定了相关模块的选取与使用方法。（2）完成了整个雷达系统的硬件设计。系统以Arduino为控制核心，控制三组超声波传感器测距支路，通过对测出的三组最小数值可以得到与障碍物的安全距离。（3）采用C++语言对系统的软件部分进行了编写。软件部分主要包括系统运行主程序、超声波测距程序、手机收发数据程序、车辆调速程序和报警制动程序，以及一些相关子程序。

但本系统仍然存在可扩展的方向，包括：（1）测距精度。超声波的传播速度与温度、湿度具有一定相关度，但本系统没有考虑到温度和湿度的因素，因此整个系统没有达到理想的精度。为了使系统可以在任何温度、湿度下都能够准确的测量，可以考虑在外围电路增加温湿度传感器模块，然后控制芯片根据实测的温湿度修正声波的传输速度，从而增加系统测距的精度。（2）电磁辐射干扰。倒车雷达系统的运行环境是相对比较复杂的，如：发动机高压点火而产生的电磁辐射和其他车辆的声波等，外界的电磁波可能会对系统的信号接收与发送造成不良的影响。所以在进行硬件搭建和软件设计时可以考虑采取适当的抗干扰措施，以提高系统的工作可靠性。

以上为倒车雷达系统的总结。在设计该系统的过程中参考了市场上已经存在的倒车雷达系统的特点。虽然整个系统存在很多的不足，但是作为相对简洁并且成本比较低的车辆雷达系统，还是具有一定实用价值的。

谢　辞

本此的设计是在我的导师曾屹的悉心指导和严格要求下完成的。曾屹老师在电路设计、硬件仿真和软件设计等方面造诣颇深，在我设计的期间给予了许多意义深远的指导。同时，他将理论知识和生产实际的背景相结合，对我的设计项目提出了宝贵的建议和意见，使我在项目设计中有更多的收获。曾屹老师丰富的专业知识、资深的实验经验和求实创新的工作作风深深的影响着我。在此，谨向曾屹老师致以我最崇高的敬意和真挚的感谢！

感谢我的家人和朋友对我生活上的关心，学习和工作的支持，这些使得我能够安心的完成我的设计工作。

最后，对在我的学习和成长道路上给予帮助的所有老师和朋友们表示深深地感谢，对评阅该论文的所有专家表示最崇高的敬意和真挚的感谢！

参考文献

[1] 张军朝.Arduino技术及应用[M].上海交通大学出版社,2018:8-30,38.

[2] 史记征,梁晶,崔俊.基于Arduino的《传感器技术及应用》课程教学改革探索[J].教育现代化,2019,6(A5):98-99+115.

[3] 汤传国. 基于超声波测距的倒车雷达系统研究[D].长安大学,2015.

[4] 潘康福. 基于超声波测距的倒车防撞报警系统研究[D].南京邮电大学,2018.

[5] Information Technology - Data Analytics; New Findings from Shenyang Agricultural University Describe Advances in Data Analytics (A Crop Canopy Localization Method Based on Ultrasonic Ranging and Iterative Self-Organizing Data Analysis Technique Algorithm)[J]. Information Technology Newsweekly,2020.

[6] 朱静.有刷直流电机旋转工作噪声分析及研究[J].无线互联科技,2019,16(16):149-150.

[7] 田中灵,高大威,方平,林阳,李杰.基于K&C试验台的扭力梁与多连杆悬架研究[J].上海理工大学学报,2019,41(06):552-556.

[8] 周风顺,王林章,李宣东.C/C++程序缺陷自动修复与确认方法[J].软件学报,2019,30(05):1243-1255.

[9] 熊慧,邱博文,刘近贞.开源平台Arduino硬件生态扩充研究[J].实验室研究与探索,2019,38(06):103-106.

[10] 魏金丽.BRT安全适应性车速建模与分析[J].科学技术与工程,2016,16(29):126-131.

[11] 刘贵如,周鸣争,王陆林,王海.城市工况下最小安全车距控制模型和避撞算法[J].汽车工程,2016,38(10):1200-1205+1176.

[12] 边春元,段鹏飞,肖鸿权,李晓霞,张争强.一种用于无刷直流电机回馈制动的PWM调制方式[J].中国电机工程学报,2019,39(17):5247-5256+5305.

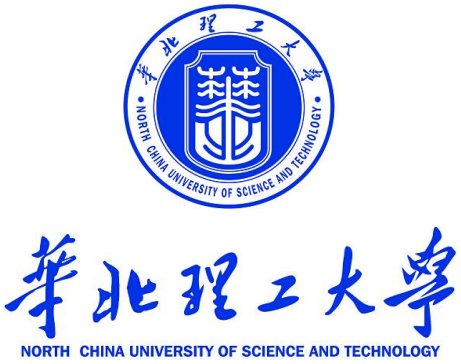
[13] 肖裕辉. 虚拟飞机管理计算机控制管理系统软件设计与实现[D].电子科技大学,2017.

[14] 黄旭方,王旭阳,孙鑫,陈冰雪.有效减小时延的自适应p-坚持CSMA协议研究[J].计算机工程与应用,2017,53(12):99-104.

[15] 杨华甫,倪子伟.基于TCP的流量控制和拥塞控制分析[J].计算机系统应用,2005(10):50-53.

附　录

学　号：**201614420112**



毕业设计说明书

附录

设计题目：**基于 Arduino 倒车雷达的设计与实现**

学生姓名：刘佳玮

专业班级：**16** 计科 **1** 班

学　　院：人工智能学院

指导教师：曾屹　　高级实验师

XXXX 年 **XX** 月 **XX** 日

在 Arduino 编程中，最常见的为使用 pinMode()函数定义引脚为输出模式或输入模式。如定义超声波的 Echo 引脚为输入模式，定义 Trig 引脚为输出模式，只有在定义完模式后方可读写数据。其实现方法为：

```
void pinMode(uint8_t pin, uint8_t mode)
{
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);
        volatile uint8_t *reg, *out;
        if (port == NOT_A_PIN) return;
        // JWS: can I let the optimizer do this?
        reg = portModeRegister(port);
        out = portOutputRegister(port);
        if (mode == INPUT) {
                uint8_t oldSREG = SREG;
                cli();
                *reg &= ~bit;
                *out &= ~bit;
                SREG = oldSREG;
        } else if (mode == INPUT_PULLUP) {
                uint8_t oldSREG = SREG;
                cli();
                *reg &= ~bit;
                *out |= bit;
                SREG = oldSREG;
        } else {
                uint8_t oldSREG = SREG;
                cli();
                *reg |= bit;
                SREG = oldSREG;
        }
}
```

其中，pinMode()调用的 portModeRegister()是一个宏定义，用于返回模式寄存器用于控制当前引脚的模式，其实现方法如下：

```
#define portModeRegister(P) ( (volatile uint8_t *)
```

```
( pgm_read_word( port_to_mode_PGM + (P))) )
const uint16_t PROGMEM port_to_mode_PGM[] = {
        NOT_A_PORT,
        NOT_A_PORT,
        (uint16_t) &DDRB,
        (uint16_t) &DDRC,
        (uint16_t) &DDRD,
};
```

与之对应的，portOutputRegister()函数用于返回对应引脚的模式，其实现方法为：

```
#define portOutputRegister(P) ( (volatile uint8_t *)
( pgm_read_word( port_to_output_PGM + (P))) )
const uint16_t PROGMEM port_to_output_PGM[] = {
    NOT_A_PORT,
    NOT_A_PORT,
    (uint16_t) &PORTB,
    (uint16_t) &PORTC,
    (uint16_t) &PORTD,
};
```

其中的 cli()为重要的宏，其功能是执行汇编程序指令以禁止中断。常用于执行编译好的指令，防止其他程序的中断。其声明为：

```
# define cli()    __asm__ __volatile__ ("cli" ::)
```

而在完成定义引脚的输入输出模式后，需要使用 digitalWrite()函数对引脚的电平进行赋值以完成初始化工作，其实现方法如下：

```
void digitalWrite(uint8_t pin, uint8_t val)
{
        uint8_t timer = digitalPinToTimer(pin);
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);
        volatile uint8_t *out;
        if (port == NOT_A_PIN) return;
        // If the pin that support PWM output, we need to turn it off
        // before doing a digital write.
```

```
        if (timer != NOT_ON_TIMER) turnOffPWM(timer);
        out = portOutputRegister(port);
        uint8_t oldSREG = SREG;
        cli();
        if (val == LOW) {
                *out &= ~bit;
        } else {
                *out |= bit;
        }
        SREG = oldSREG;
}
```

而在完成赋值后，仍然需要 digitalRead()函数读取引脚的电平，以检测是否达到指定电平，其实现方法如下：

```
int digitalRead(uint8_t pin)
{
        uint8_t timer = digitalPinToTimer(pin);
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);
        if (port == NOT_A_PIN) return LOW;
        // If the pin that support PWM output, we need to turn it off
        // before getting a digital reading.
        if (timer != NOT_ON_TIMER) turnOffPWM(timer);
        if (*portInputRegister(port) & bit) return HIGH;
        return LOW;
}
```

除去数字 IO 外，另外重要的 IO 方式为模拟量的读写，如小车调速的输出为模拟输出，模拟平缓的波形以降低电压输出，达到减速的目的。其 analogWrite()函数的实现方法为：

```
// Right now, PWM output only works on the pins with
// hardware support.  These are defined in the appropriate
// pins_*.c file.  For the rest of the pins, we default
// to digital output.
void analogWrite(uint8_t pin, int val)
{
```

```
// We need to make sure the PWM output is enabled for those pins
// that support it, as we turn it off when digitally reading or
// writing with them.  Also, make sure the pin is in output mode
// for consistenty with Wiring, which doesn't require a pinMode
// call for the analog output pins.
pinMode(pin, OUTPUT);
if (val == 0)
{
    digitalWrite(pin, LOW);
}
else if (val == 255)
{
    digitalWrite(pin, HIGH);
}
else
{
    switch(digitalPinToTimer(pin))
    {
        case TIMER0A:
            // connect pwm to pin on timer 0, channel A
            sbi(TCCR0A, COM0A1);
            OCR0A = val; // set pwm duty
            break;
        case TIMER0B:
            // connect pwm to pin on timer 0, channel B
            sbi(TCCR0A, COM0B1);
            OCR0B = val; // set pwm duty
            break;
        case TIMER1A:
            // connect pwm to pin on timer 1, channel A
            sbi(TCCR1A, COM1A1);
            OCR1A = val; // set pwm duty
            break;
        case TIMER1B:
```

```
                    // connect pwm to pin on timer 1, channel B
                    sbi(TCCR1A, COM1B1);
                    OCR1B = val; // set pwm duty
                    break;
                case TIMER2A:
                    // connect pwm to pin on timer 2, channel A
                    sbi(TCCR2A, COM2A1);
                    OCR2A = val; // set pwm duty
                    break;
                case TIMER2B:
                    // connect pwm to pin on timer 2, channel B
                    sbi(TCCR2A, COM2B1);
                    OCR2B = val; // set pwm duty
                    break;
                case NOT_ON_TIMER:
                default:
                    if (val < 128) {
                        digitalWrite(pin, LOW);
                    } else {
                        digitalWrite(pin, HIGH);
                    }
            }
        }
    }
```

其调用的 sbi()宏负责将指定地址位的数据用 1 表示，声明为：

```
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
```

而 analogWrite 仅读取特殊寄存器内的模拟量的数值，而产生模拟量数值到指定寄存器的 tone()函数的实现方法为：

```
volatile long timer0_toggle_count;
volatile uint8_t *timer0_pin_port;
volatile uint8_t timer0_pin_mask;
volatile long timer1_toggle_count;
volatile uint8_t *timer1_pin_port;
volatile uint8_t timer1_pin_mask;
```

```
volatile long timer2_toggle_count;
volatile uint8_t *timer2_pin_port;
volatile uint8_t timer2_pin_mask;
void tone(uint8_t _pin, unsigned int frequency, unsigned long duration)
{
  uint8_t prescalarbits = 0b001;
  long toggle_count = 0;
  uint32_t ocr = 0;
  int8_t _timer;
  _timer = toneBegin(_pin);
  if (_timer >= 0)
  {
    // Set the pinMode as OUTPUT
    pinMode(_pin, OUTPUT);
    // if we are using an 8 bit timer, scan through prescalars to find the best fit
    if (_timer == 0 || _timer == 2)
    {
      ocr = F_CPU / frequency / 2 - 1;
      prescalarbits = 0b001;    // ck/1: same for both timers
      if (ocr > 255)
      {
        ocr = F_CPU / frequency / 2 / 8 - 1;
        prescalarbits = 0b010;    // ck/8: same for both timers
        if (_timer == 2 && ocr > 255)
        {
          ocr = F_CPU / frequency / 2 / 32 - 1;
          prescalarbits = 0b011;
        }
        if (ocr > 255)
        {
          ocr = F_CPU / frequency / 2 / 64 - 1;
          prescalarbits = _timer == 0 ? 0b011 : 0b100;
          if (_timer == 2 && ocr > 255)
          {
```

```
            ocr = F_CPU / frequency / 2 / 128 - 1;
            prescalarbits = 0b101;
        }
        if (ocr > 255)
        {
            ocr = F_CPU / frequency / 2 / 256 - 1;
            prescalarbits = _timer == 0 ? 0b100 : 0b110;
            if (ocr > 255)
            {
                // can't do any better than /1024
                ocr = F_CPU / frequency / 2 / 1024 - 1;
                prescalarbits = _timer == 0 ? 0b101 : 0b111;
            }
        }
    }
  }
  if (_timer == 0)
  {
    TCCR0B = (TCCR0B & 0b11111000) | prescalarbits;
  }
  else
  {
    TCCR2B = (TCCR2B & 0b11111000) | prescalarbits;
  }
}
else
{
  // two choices for the 16 bit timers: ck/1 or ck/64
  ocr = F_CPU / frequency / 2 - 1;
  prescalarbits = 0b001;
  if (ocr > 0xffff)
  {
    ocr = F_CPU / frequency / 2 / 64 - 1;
    prescalarbits = 0b011;
```

```
    }
    if (_timer == 1)
    {
        TCCR1B = (TCCR1B & 0b11111000) | prescalarbits;
    }
}
// Calculate the toggle count
if (duration > 0)
{
    toggle_count = 2 * frequency * duration / 1000;
}
else
{
    toggle_count = -1;
}
// Set the OCR for the given timer,
// set the toggle count,
// then turn on the interrupts
switch (_timer)
{
    case 0:
        OCR0A = ocr;
        timer0_toggle_count = toggle_count;
        bitWrite(TIMSK0, OCIE0A, 1);
        break;
    case 1:
        OCR1A = ocr;
        timer1_toggle_count = toggle_count;
        bitWrite(TIMSK1, OCIE1A, 1);
        break;
    case 2:
        OCR2A = ocr;
        timer2_toggle_count = toggle_count;
        bitWrite(TIMSK2, OCIE2A, 1);
```

```
            break;
        }
    }
}
```

在车辆正常驾驶的条件下，可以选择最高电压输出，关闭 PWM 调制以达到节能的目的，实现代码如下：

```
static void turnOffPWM(uint8_t timer)
{
    switch (timer)
    {
        #if defined(TCCR1A) && defined(COM1A1)
        case TIMER1A:   cbi(TCCR1A, COM1A1);    break;
        #endif
        #if defined(TCCR1A) && defined(COM1B1)
        case TIMER1B:   cbi(TCCR1A, COM1B1);    break;
        #endif
        #if defined(TCCR1A) && defined(COM1C1)
        case TIMER1C:   cbi(TCCR1A, COM1C1);    break;
        #endif
        #if defined(TCCR2) && defined(COM21)
        case  TIMER2:   cbi(TCCR2, COM21);      break;
        #endif
        #if defined(TCCR0A) && defined(COM0A1)
        case  TIMER0A:  cbi(TCCR0A, COM0A1);    break;
        #endif
        #if defined(TCCR0A) && defined(COM0B1)
        case  TIMER0B:  cbi(TCCR0A, COM0B1);    break;
        #endif
        #if defined(TCCR2A) && defined(COM2A1)
        case  TIMER2A:  cbi(TCCR2A, COM2A1);    break;
        #endif
        #if defined(TCCR2A) && defined(COM2B1)
        case  TIMER2B:  cbi(TCCR2A, COM2B1);    break;
        #endif
```

```
            #if defined(TCCR3A) && defined(COM3A1)
            case   TIMER3A:   cbi(TCCR3A, COM3A1);      break;
            #endif
            #if defined(TCCR3A) && defined(COM3B1)
            case   TIMER3B:   cbi(TCCR3A, COM3B1);      break;
            #endif
            #if defined(TCCR3A) && defined(COM3C1)
            case   TIMER3C:   cbi(TCCR3A, COM3C1);      break;
            #endif
            #if defined(TCCR4A) && defined(COM4A1)
            case   TIMER4A:   cbi(TCCR4A, COM4A1);      break;
            #endif
            #if defined(TCCR4A) && defined(COM4B1)
            case   TIMER4B:   cbi(TCCR4A, COM4B1);      break;
            #endif
            #if defined(TCCR4A) && defined(COM4C1)
            case   TIMER4C:   cbi(TCCR4A, COM4C1);      break;
            #endif
            #if defined(TCCR4C) && defined(COM4D1)
            case TIMER4D:  cbi(TCCR4C, COM4D1);break;
            #endif
            #if defined(TCCR5A)
            case   TIMER5A:   cbi(TCCR5A, COM5A1);      break;
            case   TIMER5B:   cbi(TCCR5A, COM5B1);      break;
            case   TIMER5C:   cbi(TCCR5A, COM5C1);      break;
            #endif
        }
    }
```

此外，在超声波测距中，检测 IO 口的电平变化以完成测距的目的，其中检测函数 pulseIn()的实现方法如下所示：

```
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
    // cache the port and bit of the pin in order to speed up the
```

```
    // pulse width measuring loop and achieve finer resolution. calling
    // digitalRead() instead yields much coarser resolution.
    uint8_t bit = digitalPinToBitMask(pin);
    uint8_t port = digitalPinToPort(pin);
    uint8_t stateMask = (state ? bit : 0);
    // convert the timeout from microseconds to a number of times through
    // the initial loop; it takes approximately 16 clock cycles per iteration
    unsigned long maxloops = microsecondsToClockCycles(timeout)/16;
    unsigned long width =
    countPulseASM(portInputRegister(port), bit, stateMask, maxloops);
    // prevent clockCyclesToMicroseconds to return bogus values
    // if countPulseASM timed out
    if (width)
        return clockCyclesToMicroseconds(width * 16 + 16);
    else
        return 0;
}
```

在 pulseIn 函数中调用了 clockCyclesToMicroseconds 宏，以完成数字和时间的转换，其声明如下：

```
#define clockCyclesPerMicrosecond() ( F_CPU / 1000000L )
#define clockCyclesToMicroseconds(a) ( (a) / clockCyclesPerMicrosecond() )
#define microsecondsToClockCycles(a) ( (a) * clockCyclesPerMicrosecond() )
```

但其中难点为计算脉冲长度函数 countOulseASM()函数，以判断当前电平是否为低电平，其实现方法为：

```
#include <avr/io.h>
.section .text
.global countPulseASM
countPulseASM:
.LM0:
.LFBB1:
    push r12    ;    ;    130 pushqi1/1 [length = 1]
    push r13    ;    ;    131 pushqi1/1 [length = 1]
    push r14    ;    ;    132 pushqi1/1 [length = 1]
    push r15    ;    ;    133 pushqi1/1 [length = 1]
```

```
	push r16	;	;	134 pushqi1/1 [length = 1]
	push r17	;	;	135 pushqi1/1 [length = 1]
/* prologue: function */
/* frame size = 0 */
/* stack size = 6 */
.L__stack_usage = 6
	mov r30,r24	;	port, port	;	2 *movhi/1	[length = 2]
	mov r31,r25	;	port, port
/*		unsigned long width = 0;
***		// wait for any previous pulse to end
***		while ((*port & bit) == stateMask)
*/
.LM1:
	rjmp .L2	;	;	181 jump	[length = 1]
.L4:
/*			if (--maxloops == 0) */
.LM2:
	subi r16,1	;	maxloops,	;	17	addsi3/2	[length = 4]
	sbc r17, r1	;	maxloops
	sbc r18, r1	;	maxloops
	sbc r19, r1	;	maxloops
	breq .L13	; ,	;	19	branch	[length = 1]
.L2:
/*			if (--maxloops == 0) */
.LM3:
	ld r25,Z	;	D.1554, *port_7(D)	;	22	movqi_insn/4	[length = 1]
	and r25,r22	;	D.1554, bit	;	24	andqi3/1	[length = 1]
	cp r25,r20	;	D.1554, stateMask	;	25	*cmpqi/2	[length = 1]
	breq .L4	; ,	;	26	branch	[length = 1]
	rjmp .L6	;	;	184 jump	[length = 1]
.L7:
/*				return 0;
***
***		// wait for the pulse to start
```

```
***         while ((*port & bit) != stateMask)
***                 if (--maxloops == 0)
*/
.LM4:
        subi r16,1      ;    maxloops,    ;    31    addsi3/2    [length = 4]
        sbc r17, r1     ;    maxloops
        sbc r18, r1     ;    maxloops
        sbc r19, r1     ;    maxloops
        breq .L13    ; ,     ;    33    branch    [length = 1]
.L6:
/*                 if (--maxloops == 0) */
.LM5:
        ld r25,Z      ;    D.1554, *port_7(D)      ;    41    movqi_insn/4    [length = 1]
        and r25,r22    ;    D.1554, bit    ;    43    andqi3/1    [length = 1]
        cpse r25,r20 ;    D.1554, stateMask    ;    44 enable_interrupt-3 [length = 1]
        rjmp .L7      ;
        mov r12, r1     ;    width    ;    7 *movsi/2    [length = 4]
        mov r13, r1     ;    width
        mov r14, r1     ;    width
        mov r15, r1     ;    width
        rjmp .L9      ;     ;    186 jump    [length = 1]
.L10:
/*                      return 0;
***
***         // wait for the pulse to stop
***         while ((*port & bit) == stateMask) {
***                 if (++width == maxloops)
*/
.LM6:
        ldi r24,-1      ; ,     ;    50    addsi3/3    [length = 5]
        sub r12,r24    ;    width,
        sbc r13,r24    ;    width,
        sbc r14,r24    ;    width,
        sbc r15,r24    ;    width,
```

```
	cp r16,r12	;	maxloops, width	;	51	*cmpsi/2	[length = 4]
	cpc r17,r13	;	maxloops, width
	cpc r18,r14	;	maxloops, width
	cpc r19,r15	;	maxloops, width
	breq .L13	; ,	;	52	branch	[length = 1]
.L9:
/*		if (++width == maxloops) */
.LM7:
	ld r24,Z	;	D.1554, *port_7(D)	;	60	movqi_insn/4	[length = 1]
	and r24,r22	;	D.1554, bit	;	62	andqi3/1	[length = 1]
	cp r24,r20	;	D.1554, stateMask	;	63	*cmpqi/2	[length = 1]
	breq .L10	; ,	;	64	branch	[length = 1]
/*			return 0;
***	}
***	return width;
*/
.LM8:
	mov r22,r12	;	D.1553, width	;	108 movqi_insn/1	[length = 1]
	mov r23,r13	;	D.1553, width	;	109 movqi_insn/1	[length = 1]
	mov r24,r14	;	D.1553, width	;	110 movqi_insn/1	[length = 1]
	mov r25,r15	;	D.1553, width	;	111 movqi_insn/1	[length = 1]
/* epilogue start */
.LM9:
	pop r17	;	;	171 popqi [length = 1]
	pop r16	;	;	172 popqi [length = 1]
	pop r15	;	;	173 popqi [length = 1]
	pop r14	;	;	174 popqi [length = 1]
	pop r13	;	;	175 popqi [length = 1]
	pop r12	;	;	176 popqi [length = 1]
	ret	;	177 return_from_epilogue	[length = 1]
.L13:
.LM10:
	ldi r22,0	;	D.1553	;	120 movqi_insn/1	[length = 1]
	ldi r23,0	;	D.1553	;	121 movqi_insn/1	[length = 1]
```

```
        ldi r24,0   ;   D.1553     ;   122 movqi_insn/1    [length = 1]
        ldi r25,0   ;   D.1553     ;   123 movqi_insn/1    [length = 1]
    /* epilogue start */
    .LM11:
        pop r17   ;     ;   138 popqi [length = 1]
        pop r16   ;     ;   139 popqi [length = 1]
        pop r15   ;     ;   140 popqi [length = 1]
        pop r14   ;     ;   141 popqi [length = 1]
        pop r13   ;     ;   142 popqi [length = 1]
        pop r12   ;     ;   143 popqi [length = 1]
        ret   ;   144 return_from_epilogue    [length = 1]
```

而在 Arduino 运行期间，另一个重要的函数为延时函数。因 CPU 的运行时间和外部设备的运行时间并非同一数量级，前者的单位是微秒，而后者可能需要用秒来衡量。因此，常用延时函数来延缓 CPU 的运行以达到等待外部设备准备就绪的目的。常用的 delay 函数的实现方法为：

```
#include "delay.h"
#include "Arduino.h"
#ifdef __cplusplus
extern "C" {
#endif
/** Tick Counter united by ms */
static volatile uint32_t _ulTickCount=0 ;
unsigned long millis( void )
{
// todo: ensure no interrupts
  return _ulTickCount ;
}
// Interrupt-compatible version of micros
// Theory: repeatedly take readings of SysTick counter, millis counter and SysTick interrupt pending flag.
// When it appears that millis counter and pending is stable and SysTick hasn't rolled over, use these
// values to calculate micros. If there is a pending SysTick, add one to the millis counter in the calculation.
```

```
unsigned long micros( void )
{
   uint32_t ticks, ticks2;
   uint32_t pend, pend2;
   uint32_t count, count2;
   ticks2  = SysTick->VAL;
   pend2   = !!(SCB->ICSR & SCB_ICSR_PENDSTSET_Msk)  ;
   count2  = _ulTickCount ;
   do
   {
      ticks=ticks2;
      pend=pend2;
      count=count2;
      ticks2  = SysTick->VAL;
      pend2   = !!(SCB->ICSR & SCB_ICSR_PENDSTSET_Msk)  ;
      count2  = _ulTickCount ;
   } while ((pend != pend2) || (count != count2) || (ticks < ticks2));
   return ((count+pend) * 1000) + (((SysTick->LOAD - ticks)*(1048576/(VARIANT_MCK/1000000)))>>20) ;
   // this is an optimization to turn a runtime division into two compile-time divisions and
   // a runtime multiplication and shift, saving a few cycles
}
void delay( unsigned long ms )
{
   if (ms == 0)
   {
      return;
   }
   uint32_t start = micros();
   while (ms > 0)
   {
      yield();
      while (ms > 0 && (micros() - start) >= 1000)
```

```
    {
      ms--;
      start += 1000;
    }
  }
}
#include "Reset.h" // for tickReset()
void SysTick_DefaultHandler(void)
{
  // Increment tick count each ms
  _ulTickCount++;
  tickReset();
}
#ifdef __cplusplus
}
#endif
```

不仅可以以秒为单位进行延时操作，在超声波模块中，延时的单位为毫秒，此时便可借助 delayMicroseconds()函数实现毫秒单位的延时，其实现方法如下：

```
static inline void delayMicroseconds(uint16_t) __attribute__((always_inline, unused));
static inline void delayMicroseconds(uint16_t usec)
{
    if (__builtin_constant_p(usec)) {
        #if F_CPU == 16000000L
        uint16_t tmp = usec * 4;
        #elif F_CPU == 8000000L
        uint16_t tmp = usec * 2;
        #elif F_CPU == 4000000L
        uint16_t tmp = usec;
        #elif F_CPU == 2000000L
        uint16_t tmp = usec / 2;
        if (usec == 1) {
            asm volatile("rjmp L%=\nL%=:\n" ::);
        }
```

```
#elif F_CPU == 1000000L
uint16_t tmp = usec / 4;
if (usec == 1) {
        asm volatile("nop\n");
} else if (usec == 2) {
        asm volatile("rjmp L%=\nL%=:\n" ::);
} else if (usec == 3) {
        asm volatile("rjmp L%=\nL%=:\n" ::);
        asm volatile("nop\n");
}
#else
#error "Clock must be 16, 8, 4, 2 or 1 MHz"
#endif
if (tmp > 0) {
    if (tmp < 256) {
        uint8_t tmp2 = tmp;
        asm volatile(
            "L_%=_loop:"            // 1 to load
            "subi   %0, 1"           "\n\t"   // 1
            "nop"                    "\n\t"   // 1
            "brne   L_%=_loop"       "\n\t"   // 2 (1 on last)
                    : "=d" (tmp2)
                    : "0" (tmp2)
                );
    } else {
        asm volatile(
            "L_%=_loop:"             // 2 to load
            "sbiw   %A0, 1"          "\n\t"   // 2
            "brne   L_%=_loop"       "\n\t"   // 2 (1 on last)
                    : "=w" (tmp)
                    : "0" (tmp)
                );
        }
}
```

```
} else {
    asm volatile(
    #if F_CPU == 16000000L
            "sbiw    %A0, 2"                    "\n\t"    // 2
            "brcs    L_%=_end"                  "\n\t"    // 1
            "breq    L_%=_end"                  "\n\t"    // 1
            "lsl     %A0"                       "\n\t"    // 1
            "rol     %B0"                       "\n\t"    // 1
            "lsl     %A0"                       "\n\t"    // 1
            "rol     %B0"          "\n\t"    // 1    overhead: (8)/4 = 2us
        #elif F_CPU == 8000000L
            "sbiw    %A0, 3"                    "\n\t"    // 2
            "brcs    L_%=_end"                  "\n\t"    // 1
            "breq    L_%=_end"                  "\n\t"    // 1
            "lsl     %A0"                       "\n\t"    // 1
            "rol     %B0"        "\n\t"    // 1    overhead: (6)/2 = 3 us
        #elif F_CPU == 4000000L
            "sbiw    %A0, 4"                    "\n\t"    // 2
            "brcs    L_%=_end"                  "\n\t"    // 1
            "breq    L_%=_end"      "\n\t"    // 1    overhead: (4) = 4 us
        #elif F_CPU == 2000000L
            "sbiw    %A0, 12"                   "\n\t"    // 2
            "brcs    L_%=_end"                  "\n\t"    // 1
            "breq    L_%=_end"                  "\n\t"    // 1
            "lsr     %B0"                       "\n\t"    // 1
            "ror     %A0"        "\n\t"    // 1    overhead: (6)*2 = 12 us
        #elif F_CPU == 1000000L
            "sbiw    %A0, 32"                   "\n\t"    // 2
            "brcs    L_%=_end"                  "\n\t"    // 1
            "breq    L_%=_end"                  "\n\t"    // 1
            "lsr     %B0"                       "\n\t"    // 1
            "ror     %A0"                       "\n\t"    // 1
            "lsr     %B0"                       "\n\t"    // 1
            "ror     %A0"        "\n\t"    // 1    overhead: (8)*4 = 32 us
```

```
                #endif
                "L_%=_loop:"
                    "sbiw     %A0, 1"                          "\n\t"    // 2
                    "brne     L_%=_loop"                       "\n\t"    // 2
                "L_%=_end:"
                    : "=w" (usec)
                    : "0" (usec)
                );
        }
    }
```

车辆与障碍物的距离到达安全阈值后，紧急制动通过 attachInterrupt()中断函数完成，其实现方法如下：

```
    static void nothing(void) {
    }
    static volatile voidFuncPtr intFunc[EXTERNAL_NUM_INTERRUPTS] = {
      nothing,
      nothing,
    };
    void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode) {
      if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
        intFunc[interruptNum] = userFunc;
        // Configure the interrupt mode (trigger on low input, any change, rising
        // edge, or falling edge).    The mode constants were chosen to correspond
        // to the configuration bits in the hardware register, so we simply shift
        // the mode into place.
        // Enable the interrupt.
        switch (interruptNum) {
        case 0:
          EICRA = (EICRA & ~((1 << ISC00) | (1 << ISC01))) | (mode << ISC00);
          EIMSK |= (1 << INT0);
           break;
        case 1:
          EICRA = (EICRA & ~((1 << ISC10) | (1 << ISC11))) | (mode << ISC10);
          EIMSK |= (1 << INT1);
```

```
            break;
        case 2:
            break;
        }
      }
    }
```

解除中断的 detachInterrupt()函数的实现方法为：

```
void detachInterrupt(uint8_t interruptNum) {
  if(interruptNum < EXTERNAL_NUM_INTERRUPTS) {
    // Disable the interrupt.    (We can't assume that interruptNum is equal
    // to the number of the EIMSK bit to clear, as this isn't true on the
    // ATmega8.    There, INT0 is 6 and INT1 is 7.)
    switch (interruptNum) {
    case 0:
      EIMSK &= ~(1 << INT0);
      break;
    case 1:
      EIMSK &= ~(1 << INT1);
      break;
    case 2:
      break;
    }
    intFunc[interruptNum] = nothing;
  }
}
```

而中断通过中断向量被计算机识别，中断向量的定义如下：

```
#define INT0_vect_num         1
#define INT0_vect             _VECTOR(1)
/* External Interrupt Request 0 */
#define INT1_vect_num         2
#define INT1_vect             _VECTOR(2)
/* External Interrupt Request 1 */
#define PCINT0_vect_num       3
#define PCINT0_vect           _VECTOR(3)
```

```
/* Pin Change Interrupt Request 0 */
#define PCINT1_vect_num      4
#define PCINT1_vect          _VECTOR(4)
/* Pin Change Interrupt Request 0 */
#define PCINT2_vect_num      5
#define PCINT2_vect          _VECTOR(5)
/* Pin Change Interrupt Request 1 */
#define WDT_vect_num         6
#define WDT_vect             _VECTOR(6)
/* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect_num 7
#define TIMER2_COMPA_vect _VECTOR(7)
/* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect_num 8
#define TIMER2_COMPB_vect _VECTOR(8)
/* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect_num      9
#define TIMER2_OVF_vect      _VECTOR(9)
/* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect_num    10
#define TIMER1_CAPT_vect    _VECTOR(10)
/* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect_num 11
#define TIMER1_COMPA_vect _VECTOR(11)
/* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect_num 12
#define TIMER1_COMPB_vect _VECTOR(12)
/* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect_num      13
#define TIMER1_OVF_vect      _VECTOR(13)
/* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect_num 14
#define TIMER0_COMPA_vect _VECTOR(14)
/* TimerCounter0 Compare Match A */
```

```
#define TIMER0_COMPB_vect_num 15
#define TIMER0_COMPB_vect _VECTOR(15)
/* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect_num    16
#define TIMER0_OVF_vect      _VECTOR(16)
/* Timer/Couner0 Overflow */
#define SPI_STC_vect_num    17
#define SPI_STC_vect           _VECTOR(17)
/* SPI Serial Transfer Complete */
#define USART_RX_vect_num 18
#define USART_RX_vect         _VECTOR(18)
/* USART Rx Complete */
#define USART_UDRE_vect_num      19
#define USART_UDRE_vect      _VECTOR(19)
/* USART, Data Register Empty */
#define USART_TX_vect_num 20
#define USART_TX_vect         _VECTOR(20)
/* USART Tx Complete */
#define ADC_vect_num           21
#define ADC_vect                  _VECTOR(21)
/* ADC Conversion Complete */
#define EE_READY_vect_num 22
#define EE_READY_vect         _VECTOR(22)
/* EEPROM Ready */
#define ANALOG_COMP_vect_num    23
#define ANALOG_COMP_vect    _VECTOR(23)
/* Analog Comparator */
#define TWI_vect_num           24
#define TWI_vect                  _VECTOR(24)
/* Two-wire Serial Interface */
#define SPM_READY_vect_num        25
#define SPM_READY_vect        _VECTOR(25)
/* Store Program Memory Read */
#define _VECTORS_SIZE (26 * 4)
```

此时便可以根据中断向量表查找对应的中断服务程序的寄存器，并执行寄存器内的程序，根据中断向量查找寄存器的实现方法如下所示：

```
#    define __INTR_ATTRS used, externally_visible
#    define ISR(vector, ...)                    \
extern "C" void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
        void vector (void)
```

而车辆的运行最重要的为串口通信，如手机端发送数据到蓝牙模块，以及车辆将距离信息以串口通信返回，均涉及到 Serial 类的各种方法，下文列出每个方法的实现原理。

判断当前是否允许串口通信的 available 方法：

```
int HardwareSerial::available(void)
{
    return ((unsigned int)
    (SERIAL_RX_BUFFER_SIZE + _rx_buffer_head - _rx_buffer_tail))
    % SERIAL_RX_BUFFER_SIZE;
}
```

指定当前通信波特率的 begin 方法：

```
void HardwareSerial::begin(unsigned long baud, byte config)
{
  // Try u2x mode first
  uint16_t baud_setting = (F_CPU / 4 / baud - 1) / 2;
  *_ucsra = 1 << U2X0;
  // hardcoded exception for 57600 for compatibility with the bootloader
  // shipped with the Duemilanove and previous boards and the firmware
  // on the 8U2 on the Uno and Mega 2560. Also, The baud_setting cannot
  // be > 4095, so switch back to non-u2x mode if the baud rate is too
  // low.
  if (((F_CPU == 16000000UL) && (baud == 57600)) || (baud_setting >4095))
  {
    *_ucsra = 0;
    baud_setting = (F_CPU / 8 / baud - 1) / 2;
  }
  // assign the baud_setting, a.k.a. ubrr (USART Baud Rate Register)
```

```
  *_ubrrh = baud_setting >> 8;
  *_ubrrl = baud_setting;
  _written = false;
  //set the data bits, parity, and stop bits
  *_ucsrc = config;
  sbi(*_ucsrb, RXEN0);
  sbi(*_ucsrb, TXEN0);
  sbi(*_ucsrb, RXCIE0);
  cbi(*_ucsrb, UDRIE0);
}
```

同理，中止串口通信的 end()函数实现方法如下：

```
void HardwareSerial::end()
{
  // wait for transmission of outgoing data
  flush();

  cbi(*_ucsrb, RXEN0);
  cbi(*_ucsrb, TXEN0);
  cbi(*_ucsrb, RXCIE0);
  cbi(*_ucsrb, UDRIE0);

  // clear any received data
  _rx_buffer_head = _rx_buffer_tail;
}
```

通过串口读入数据 read()函数的实现方法：

```
int HardwareSerial::read(void)
{
  // if the head isn't ahead of the tail, we don't have any characters
  if (_rx_buffer_head == _rx_buffer_tail) {
    return -1;
  } else {
    unsigned char c = _rx_buffer[_rx_buffer_tail];
    _rx_buffer_tail =
    (rx_buffer_index_t)(_rx_buffer_tail + 1) % SERIAL_RX_BUFFER_SIZE;
```

```
    return c;
  }
}
```

而 read()函数在读缓冲区时会更改缓冲区内的数据，相对应 peek()函数则以不修改缓冲区的形式读取缓冲区内的数据，其实现方法如下：

```
int HardwareSerial::read(void)
{
  // if the head isn't ahead of the tail, we don't have any characters
  if (_rx_buffer_head == _rx_buffer_tail) {
    return -1;
  } else {
    unsigned char c = _rx_buffer[_rx_buffer_tail];
    _rx_buffer_tail = (rx_buffer_index_t)(_rx_buffer_tail + 1)
% SERIAL_RX_BUFFER_SIZE;
    return c;
  }
}
```

在串口初始化的过程中，以防上次操作后停留在串口缓冲区的数据对本次操作造成不良影响，通常使用 flush()函数完成对串口的清空，其实现方法如下：

```
void HardwareSerial::flush()
{
  // If we have never written a byte, no need to flush. This special
  // case is needed since there is no way to force the TXC (transmit
  // complete) bit to 1 during initialization
  if (!_written)
    return;
  while (bit_is_set(*_ucsrb, UDRIE0) || bit_is_clear(*_ucsra, TXC0)) {
    if (bit_is_clear(SREG, SREG_I) && bit_is_set(*_ucsrb, UDRIE0))
    // Interrupts are globally disabled, but the DR empty
    // interrupt should be enabled, so poll the DR empty flag to
    // prevent deadlock
    if (bit_is_set(*_ucsra, UDRE0))
      _tx_udr_empty_irq();
  }
```

```
  // If we get here, nothing is queued anymore (DRIE is disabled) and
  // the hardware finished tranmission (TXC is set).
}
```

在 flush()函数执行中，首先判断串口的缓冲区是否暂存的写数据，写数据的实现函数为 write()，而!write()表示不曾写数据，此时便可不用清空以加速系统处理速度，write()函数的实现方法如下：

```
size_t HardwareSerial::write(uint8_t c)
{
  _written = true;
  // If the buffer and the data register is empty, just write the byte
  // to the data register and be done. This shortcut helps
  // significantly improve the effective datarate at high (>
  // 500kbit/s) bitrates, where interrupt overhead becomes a slowdown.
  if (_tx_buffer_head == _tx_buffer_tail && bit_is_set(*_ucsra, UDRE0)) {
    *_udr = c;
    sbi(*_ucsra, TXC0);
    return 1;
  }
  tx_buffer_index_t i = (_tx_buffer_head + 1) % SERIAL_TX_BUFFER_SIZE;
  // If the output buffer is full, there's nothing for it other than to
  // wait for the interrupt handler to empty it a bit
  while (i == _tx_buffer_tail) {
    if (bit_is_clear(SREG, SREG_I)) {
      // Interrupts are disabled, so we'll have to poll the data
      // register empty flag ourselves. If it is set, pretend an
      // interrupt has happened and call the handler to free up
      // space for us.
      if(bit_is_set(*_ucsra, UDRE0))
   _tx_udr_empty_irq();
    } else {
      // nop, the interrupt handler will free up space for us
    }
  }
  _tx_buffer[_tx_buffer_head] = c;
```

```
    _tx_buffer_head = i;
    sbi(*_ucsrb, UDRIE0);
    return 1;
  }
```

最后需要使用 bit_is_set() 和 bit_is_clear()两个宏函数检测缓存区寄存器内的数据是否清理完毕，其声明如下：

```
  #define bit_is_set(sfr, bit) (_SFR_BYTE(sfr) & _BV(bit))
  #define bit_is_clear(sfr, bit) (!(_SFR_BYTE(sfr) & _BV(bit)))
  #define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))
  #define _SFR_ADDR(sfr) _SFR_MEM_ADDR(sfr)
  #define _SFR_MEM_ADDR(sfr) ((uint16_t) &(sfr))
  #define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
  #define _BV(bit) (1 << (bit))
```

而项目的难点为串口数据的输出与显示，需要借助 C++的其他工具将输出串口数据解析后才能输出到屏幕，用于驾驶员的辅助驾驶，串口输出函数的实现方法为：

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <Arduino.h>
#include "Print.h"
// Public Methods //////////////////////////////////////////////////////////////
/* default implementation: may be overridden */
size_t Print::write(const uint8_t *buffer, size_t size) {
#ifdef DEBUG_ESP_CORE
  static char not_the_best_way [] PROGMEM STORE_ATTR = "Print::write(data,len) should be overridden for better efficiency\r\n";
  static bool once = false;
  if (!once) {
    once = true;
    os_printf_plus(not_the_best_way);
  }
#endif
```

```
        size_t n = 0;
        while (size--) {
                size_t ret = write(pgm_read_byte(buffer++));
                if (ret == 0) {
                        // Write of last byte didn't complete, abort additional processing
                        break;
                }
                n += ret;
        }
        return n;
}
size_t Print::printf(const char *format, ...) {
        va_list arg;
        va_start(arg, format);
        char temp[64];
        char* buffer = temp;
        size_t len = vsnprintf(temp, sizeof(temp), format, arg);
        va_end(arg);
        if (len > sizeof(temp) - 1) {
                buffer = new char[len + 1];
                if (!buffer) {
                        return 0;
                }
                va_start(arg, format);
                vsnprintf(buffer, len + 1, format, arg);
                va_end(arg);
        }
        len = write((const uint8_t*) buffer, len);
        if (buffer != temp) {
                delete[] buffer;
        }
        return len;
}
size_t Print::printf_P(PGM_P format, ...) {
```

```
	va_list arg;
	va_start(arg, format);
	char temp[64];
	char* buffer = temp;
	size_t len = vsnprintf_P(temp, sizeof(temp), format, arg);
	va_end(arg);
	if (len > sizeof(temp) - 1) {
		buffer = new char[len + 1];
		if (!buffer) {
			return 0;
		}
		va_start(arg, format);
		vsnprintf_P(buffer, len + 1, format, arg);
		va_end(arg);
	}
	len = write((const uint8_t*) buffer, len);
	if (buffer != temp) {
		delete[] buffer;
	}
	return len;
}
size_t Print::print(const __FlashStringHelper *ifsh) {
	PGM_P p = reinterpret_cast<PGM_P>(ifsh);
	char buff[128] __attribute__ ((aligned(4)));
	auto len = strlen_P(p);
	size_t n = 0;
	while (n < len) {
		int to_write = std::min(sizeof(buff), len - n);
		memcpy_P(buff, p, to_write);
		auto written = write(buff, to_write);
		n += written;
		p += written;
		if (!written) {
			// Some error, write() should write at least 1 byte before returning
```

```
                break;
            }
        }
        return n;
}
size_t Print::print(const String &s) {
        return write(s.c_str(), s.length());
}
size_t Print::print(const char str[]) {
        return write(str);
}
size_t Print::print(char c) {
        return write(c);
}
size_t Print::print(unsigned char b, int base) {
        return print((unsigned long) b, base);
}
size_t Print::print(int n, int base) {
        return print((long) n, base);
}
size_t Print::print(unsigned int n, int base) {
        return print((unsigned long) n, base);
}
size_t Print::print(long n, int base) {
        if(base == 0) {
            return write(n);
        } else if(base == 10) {
            if(n < 0) {
                int t = print('-');
                n = -n;
                return printNumber(n, 10) + t;
            }
            return printNumber(n, 10);
        } else {
```

```
            return printNumber(n, base);
        }
}
size_t Print::print(unsigned long n, int base) {
        if(base == 0)
            return write(n);
        else
            return printNumber(n, base);
}
size_t Print::print(double n, int digits) {
        return printFloat(n, digits);
}
size_t Print::println(const __FlashStringHelper *ifsh) {
        size_t n = print(ifsh);
        n += println();
        return n;
}
size_t Print::print(const Printable& x) {
        return x.printTo(*this);
}
size_t Print::println(void) {
        return print("\r\n");
}
size_t Print::println(const String &s) {
        size_t n = print(s);
        n += println();
        return n;
}
size_t Print::println(const char c[]) {
        size_t n = print(c);
        n += println();
        return n;
}
size_t Print::println(char c) {
```

```
	size_t n = print(c);
	n += println();
	return n;
}
size_t Print::println(unsigned char b, int base) {
	size_t n = print(b, base);
	n += println();
	return n;
}
size_t Print::println(int num, int base) {
	size_t n = print(num, base);
	n += println();
	return n;
}
size_t Print::println(unsigned int num, int base) {
	size_t n = print(num, base);
	n += println();
	return n;
}
size_t Print::println(long num, int base) {
	size_t n = print(num, base);
	n += println();
	return n;
}
size_t Print::println(unsigned long num, int base) {
	size_t n = print(num, base);
	n += println();
	return n;
}
size_t Print::println(double num, int digits) {
	size_t n = print(num, digits);
	n += println();
	return n;
}
```

```
size_t Print::println(const Printable& x) {
    size_t n = print(x);
    n += println();
    return n;
}
// Private Methods /////////////////////////////////////////////////////////////
size_t Print::printNumber(unsigned long n, uint8_t base) {
    char buf[8 * sizeof(long) + 1]; // Assumes 8-bit chars plus zero byte.
    char *str = &buf[sizeof(buf) - 1];
    *str = '\0';
    // prevent crash if called with base == 1
    if(base < 2)
        base = 10;
    do {
        unsigned long m = n;
        n /= base;
        char c = m - base * n;
        *--str = c < 10 ? c + '0' : c + 'A' - 10;
    } while(n);
    return write(str);
}
size_t Print::printFloat(double number, uint8_t digits) {
    char buf[40];
    return write(dtostrf(number, 0, digits, buf));
}
```

考虑到超声波的探测距离的限制和优化用户体验，使用 constrain()函数将距离映射到规定区间内，将距离的安全与否直观的反馈给用户，其实现方法如下：

```
    template<class T>
    const T& constrain(const T& x, const T& a, const T& b) {
        if(x < a) {
            return a;
        }
        else if(b < x) {
            return b;
```

```
        }
        else
            return x;
    }
```

而无论哪一侧的超声波距离达到安全阈值时均应发出警报，因此使用 min() 函数求得三组距离中的最小值，其实现方法如下：

```
    template<class ForwardIt, class Compare>
    ForwardIt min_element(ForwardIt first, ForwardIt last, Compare comp)
    {
        if (first == last) return last;
        ForwardIt smallest = first;
        ++first;
        for (; first != last; ++first) {
            if (comp(*first, *smallest)) {
                smallest = first;
            }
        }
        return smallest;
    }
```

Arduino 驱动超声波模块的程序如下所示：

```
int trigPin = 11;       //Trig
int echoPin = 12;       //Echo
long duration, cm, inches;
void setup() {
  //Serial Port begin
  Serial.begin (9600);
  //Define inputs and outputs
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
}
void loop()
{
  // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
  // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
```

```
  digitalWrite(trigPin, LOW);
  delayMicroseconds(5);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Read the signal from the sensor: a HIGH pulse whose
  // duration is the time (in microseconds) from the sending
  // of the ping to the reception of its echo off of an object.
  duration = pulseIn(echoPin, HIGH);
  // convert the time into a distance
  cm = (duration/2) / 29.1;
  inches = (duration/2) / 74;
  Serial.print(inches);
  Serial.print("in, ");
  Serial.print(cm);
  Serial.print("cm");
  Serial.println();
  delay(1000);
}
```

Arduino 实现与手机端蓝牙通信的示例代码如下所示：

```
#include <SoftwareSerial.h>
// Pin10 is RX，connect TXD of HC-05
// Pin11 is TX，connect RXD of HC-05
SoftwareSerial BT(10, 11);
char val;
void setup() {
  Serial.begin(38400);
  Serial.println("BT is ready!");
  // HC-05 default 38400
  BT.begin(38400);
}
void loop() {
  if (Serial.available()) {
    val = Serial.read();
```

```
    BT.print(val);
  }
  if (BT.available()) {
    val = BT.read();
    Serial.print(val);
  }
}
```

Arduino 通过控制 L298N 电机驱动模块实现车身运转的程序如下所示：

```
int input1 = 5; //  定义 uno 的 pin 5  向  input1  输出
int input2 = 6; //  定义 uno 的 pin 6  向  input2  输出
int input3 = 9; //  定义 uno 的 pin 9  向  input3  输出
int input4 = 10; //  定义 uno 的 pin 10  向  input4  输出
void setup() {
//   Serial.begin (9600);
//初始化各 IO,模式为 OUTPUT  输出模式
pinMode(input1, OUTPUT);
pinMode(input2, OUTPUT);
pinMode(input3, OUTPUT);
pinMode(input4, OUTPUT);
}
void loop() {
  //forward  向前转
  digitalWrite(input1, HIGH); //给高电平
  digitalWrite(input2, LOW);   //给低电平
  digitalWrite(input3, HIGH); //给高电平
  digitalWrite(input4, LOW);   //给低电平
  delay(1000);     //延时 1 秒
  //stop  停止
  digitalWrite(input1, LOW);
  digitalWrite(input2, LOW);
  digitalWrite(input3, LOW);
  digitalWrite(input4, LOW);
  delay(500);   //延时 0.5 秒
  //back  向后转
```

```
  digitalWrite(input1, LOW);
  digitalWrite(input2, HIGH);
  digitalWrite(input3, LOW);
  digitalWrite(input4, HIGH);
  delay(1000);
}
```

Arduino 通过模拟 IO 口进行 PWM 调制对外输出以达到使车辆减速的目的，其中，PWM 调速的示例程序如下所示：

```
int pinI1=8;//定义 I1 接口
int pinI2=9;//定义 I2 接口
int pwmPin=10;//定义 EA(PWM 调速)接口
int speed=200;
int vTest=2;//定义中断测速接口
int nWheel = 0;   //记录测速模块的次数
float omega = 0;   //角速度
float velocity(int n)
{
    //角速度的计算公式为（n/20）*（2pi)，即 n*0.31415
    float vel = n*0.31415;
    return vel;
}
void flash()
{
    int nr;
    nr = nWheel;
    omega = velocity(nr);
    nWheel = 0;
    Serial.print(omega);
    Serial.println("rad/s");
}
void setup()
{
  Serial.begin(9600);   //串口波特率为 9600
  attachInterrupt(0,count, FALLING);
```

```
    pinMode(pinI1,OUTPUT);//定义该接口为输出接口
    pinMode(pinI2,OUTPUT);
    pinMode(pwmPin,OUTPUT);
    pinMode(vTest,INPUT);
    pinMode(13, OUTPUT);
    digitalWrite(pinI1,LOW);//使直流电机顺时针转
    digitalWrite(pinI2,HIGH);
    // 中断设置函数，每 1s 进入一次中断
    MsTimer2::set(1000, flash);
    MsTimer2::start();
  }
  void loop()
  {
    analogWrite(pwmPin,speed);
    digitalWrite(13, HIGH);
  }
  void count()
  {
    nWheel++;
  }
```

Arduino 检测中断并执行中断服务程序的示例代码如下所示：

```
int pinInterrupt = 2; //接中断信号的脚
void onChange()
{
  if ( digitalRead(pinInterrupt) == LOW )
    Serial.println("Key Down");
  else
    Serial.println("Key UP");
}
void setup()
{
  Serial.begin(9600); //打开串口
  pinMode( pinInterrupt, INPUT);//设置管脚为输入
  //Enable 中断管脚，中断服务程序为 onChange()，监视引脚变化
```

```
    attachInterrupt( digitalPinToInterrupt(pinInterrupt), onChange, CHANGE);
}
void loop()
{
  // 模拟长时间运行的进程或复杂的任务。
  for (int i = 0; i < 100; i++)
  {
    // 什么都不做，等待 10 毫秒
    delay(10);
  }
}
```