

Ubuntu串口通讯

查看串口设备

1. 打开终端，插入串口设备，输入下面指令

```
1 | ls /dev/tty*
```

2. 通过对比分配设备前后/dev/ 目录下的 tty* 文件，可以了解到插入的 USB 转串口线对应的是哪个设备文件。

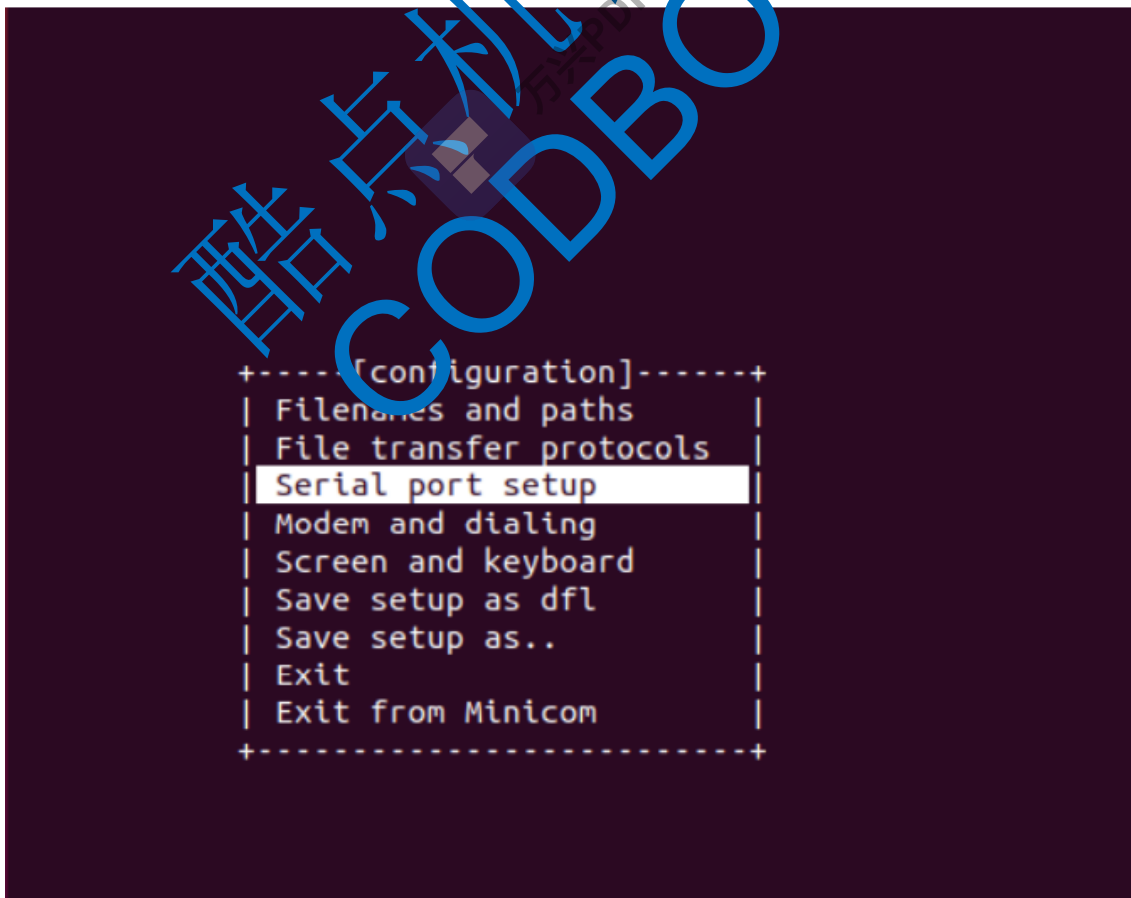
安装和配置 minicom

1. 输入以下指令，安装minicom

```
1 | sudo apt install minicom
```

2. 安装成功后，赋予root权限, 进入minicom配置界面

```
1 | sudo minicom -s
```



上图是 minicom 运行时的配置界面，注意执行 minicom 命令时需要使用 sudo 获取权限，否则无法修改设备的参数。在该界面中使用键盘的上下方向键和回车键可以进入菜单进行配置。下面我们选择“Serial port setup”菜单配置串口参数，如下图所示：

```
+-----+
| A -   Serial Device       : /dev/ttyS6
| B - Lockfile Location    : /var/lock
| C -   Callin Program     :
| D - Callout Program      :
| E -   Bps/Par/Bits       : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
|
| Change which setting? █
+-----+
|
| Screen and keyboard
| Save setup as dfl
| Save setup as..
| Exit
| Exit from Minicom
|
+-----+
```

在配置串口参数页面中根据提示的按键“A”、“E”、“F”配置串口设备为“/dev/ttyS6”（根据自己的电脑设备选择）、波特率为“115200”、以及不使用硬件流控“No”。配置完成后按回车键退出当前菜单。然后再选择“Save setup as dfl”菜单保存配置（见下图（若提示无法保存，请确保前面是使用“sudo”权限运行 minicom 的），保存完成后选择“Exit”菜单或按键盘的“Esc”键即可进入终端界面。如下图所示：

```
Welcome to minicom 2.7.1
OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyS6, 11:01:30
Press CTRL-A Z for help on special keys
█

CTRL-A Z for help | 115200 8N1 | N0R | Minicom 2.7.1 | VT102 | Offline | ttyS6
```

在 minicom 的终端界面中，按下 Ctrl+A 键再按下 Z 键可以查看帮助，按下 Ctrl+A 键再按下 X 键可以退出

串口通讯

ubuntu上串口通讯一共分为三个部分，终端设备配置、串口发送数据、串口读取数据。

1. 终端串口配置

在终端串口配置中，主要用到一个结构体termios,这个结构体用于设置终端设备的参数，包括波特率、数据 位数、校验位等，具体配置使用如下：

```
1  int fd;
2  /*获取串口文件描述符,并使用读写权限打开串口*/
3  fd = open(path, O_RDWR);
4
5  if(fd < 0)
6  {
7      printf("fail to open %s device \n", path);
8      return 0;
9  }
10
11 /*用于串口波特率、数据 位数、校验位配置*/
12 struct termios opt;
13
14 /*清空串口接收数据缓冲区*/
15 tcflush(fd, TCIOFLUSH);
16
17 //获取串口参数 opt
18 tcgetattr(fd, &opt);
19
20 //设置串口输出波特率为115200
21 cfsetospeed(&opt, B115200);
22
23 //设置串口输入波特率为115200
24 cfsetispeed(&opt, B115200);
25
26 //设置串口数据位
27 opt.c_cflag &= ~CSIZE;
28 opt.c_cflag |= CS8; /*设置数据位为8位*/
29
30 //校验位
31 opt.c_cflag &= ~PARENB; /*不使用奇偶校验*/
32 opt.c_cflag &= ~IPCK; /*禁止输入奇偶检测*/
33
34 //设置停止位
35 opt.c_cflag &= ~CSTOPB; /*设置停止位为1位*/
36
37 opt.c_iflag &= ~(INLCR); /*禁止将输入中的换行符NL映射为回车-换行CR*/
38 opt.c_iflag &= ~(IXON | IXOFF | IXANY); //不要软件流控制
39 opt.c_oflag &= ~OPOST;
40 opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
41 //更新配置
42 tcsetattr(fd, TCSANOW, &opt);
```

以上代码就是对终端串口的参数配置

2. 串口写数据

串口写数据主要使用 write(), 该函数原型如下:

```
1  ssize_t write(int fd, const void *buf, size_t nbyte)
2
3  fd: 文件描述符,
4  buf: 指定的缓冲区, 即指针, 指向一段内存单元;
5  nbyte: 要写入文件指定的字节数
6  返回值: 写入文档的字节数 (成功); -1 (出错)
7
8  该函数在使用时, 一定要打开串口设备。
```

3. 串口读数据

串口读取数据主要使用read(),该函数原型如下:

```
1 ssize_t read(int fd, void *buf, size_t count);
2
3 参数
4 count
5 是请求读取的字节数, 读上来的数据保存在缓冲区buf中, 同时文件的当前读写位置向后移。注意
  这个读写位置和使用C标准I/O库时的读写位置有可能不同, 这个读写位置是记在内核中的, 而使用
  C标准I/O库时的读写位置是用户空间I/O缓冲区中的位置。比如用fgetc读一个字节, fgetc有可能
  从内核中预读1024个字节到I/O缓冲区中, 再返回第一个字节, 这时该文件在内核中记录的读写
  位置是1024, 而在FILE结构体中记录的读写位置是1。注意返回值类型是ssize_t, 表示有符号
  的size_t, 这样既可以返回正的字节数、0 (表示到达文件末尾) 也可以返回负值-1 (表示出
  错)。
6 read函数返回时, 返回值说明了buf中前多少个字节是刚读上来的。有些情况下, 实际读到的字节
  数(返回值)会小于请求读的字节数count, 例如: 读常规文件时, 在读到count个字节之前已到
  达文件末尾。例如, 距文件末尾还有30个字节而请求读100个字节, 则read返回30, 下次read将
  返回0。
7
8 返回值
9 成功返回读取的字节数, 出错返回-1并设置errno, 如果在调用read之前已到达文件末尾, 则这次
  read返回0
10
11
```

串口完整数据发送接收程序

下面是完整串口数据接收发送程序, 可以直接在Ubuntu上运行使用, 运行之前请给串口设备赋予读写权限。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <fcntl.h>
4 #include <unistd.h>
5 #include <sys/stat.h>
6 #include <sys/types.h>
7 #include <termios.h>
8 #include <string.h>
9 #include <sys/ioctl.h>
10 #include <stdint.h>
11
12 /*根据串口对应的设备, 进行修改*/
13 const char default_path[] = "/dev/ttyS6";
14 int main(int argc, char *argv[])
15 {
16     int fd;
17     int res;
18     char *path;
19     char buf[1024] = "Embedfire tty send test\n";
20
21     if(argc > 1)
22         path = argv[1];
23     else
24         path = (char *)default_path;
25
26     printf("this is tty/usart demo\n");
```

```
27
28  /*获取串口文件描述符*/
29  fd = open(path, O_RDWR);
30
31  if(fd < 0)
32  {
33      printf("fail to open %s device \n", path);
34      return 0;
35  }
36
37  struct termios opt;
38
39  /*清空串口接收数据缓冲区*/
40  tcflush(fd, TCIOFLUSH);
41
42  //获取串口参数 opt
43  tcgetattr(fd, &opt);
44
45  //设置串口输出波特率
46  cfsetospeed(&opt, B115200);
47
48  //设置串口输入波特率
49  cfsetispeed(&opt, B115200);
50
51  //设置串口数据位
52  opt.c_cflag &= ~CSIZE;
53  opt.c_cflag |= CS8; /*设置数据位为8位*/
54
55  //校验位
56  opt.c_cflag &= ~PARENB; /*不使用奇偶校验*/
57  opt.c_iflag &= ~INPCK; /*禁止输入奇偶校验*/
58
59  //设置停止位
60  opt.c_cflag &= ~CSTOPB; /*设置停止位为1位*/
61
62  opt.c_iflag &= ~(INLCR); /*禁止将输入中的换行符NL映射为回车-换行CR*/
63
64  opt.c_iflag &= ~(IXON | TXOFF | IXANY); //不要软件流控制
65  opt.c_oflag &= ~OPOST;
66  opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
67  //更新配置
68  tcsetattr(fd, TCSANOW, &opt);
69
70  printf("device %s is set to 115200bps, 8N1", path);
71
72  do{
73      write(fd, buf, strlen(buf));
74
75      res = read(fd, buf, 1024);
76      if(res > 0)
77      {
78          buf[res] = '\0';
79          for(int i = 0; i < res; i ++){
80              printf("%x ", buf[i]);
81              packet_unpack(buf, res);
82          }
83      }
84      usleep(200000);
```

```

85     }while(res >= 0);
86
87     printf("read error,res = %d",res);
88
89     close(fd);
90
91     return 0;
92
93 }
94

```

运行该程序，即可看到接收的16进制的数据

```

00 57 00 43 00 26 4c 00 27 e 00 00 4a 00 3a 00 70 50 7d 1f 6 00 00 00 00 00 00 1 00 57 00 43 00 26
4c 00 27 e 00 00 4a 00 3a 00 70 50 7d 1f 6 1 0 1 1 0 1 1 0 1 00 18 00 8 0 26 60 00 27 e 00 4a 0
0 3a 00 70 79 7d a 2 0 0 0 0 0 0 0 1 0 0 3 7d 9 3 00 00 00 00 00 00 3 7d a 5 00 41 00 55 00 1 10 7
d 1f 6 00 00 00 00 1 0 2 0 0 66 00 30 0 26 11 0 27 e 00 4a 00 39 00 70 4e 7d 1f 6 1 0 1 1 0 2
1 0 2 0 0 2a 00 21 0 26 7 0 27 e 00 4a 00 3a 00 70 5 7d 1f 6 00 00 00 00 1 1 0 1 00 7e 00 26 0
26 60 00 27 e 00 4a 00 3a 00 70 30 7d a 2 0 0 0 0 0 0 0 1 00 3 7d 9 3 00 00 00 00 00 00 3 7d a 5 0
0 42 00 56 00 1 10 7d 2 4 29 2d 7d 2 7 1 6 7d 1f 6 00 00 00 00 00 00 1 00 7a 00 6a 00 26 11 0 27
e 00 4a 00 3a 00 70 9 7d 1f 6 00 0 1 0 0 00 00 1 00 44 00 18 0 26 7c 0 27 e 00 4a 00 3a 00
70 29 7d 1f 6 00 0 1 0 0 00 00 00 00 35 0 3 0 26 4c 0 27 e 00 4a 00 3a 00 70 72 7d a 2 0 0 0
0 00 1 00 3 7d 9 3 00 00 00 00 3 7d a 5 00 41 00 56 00 1 13 7d 1f 6 00 00 00 1 00 1 00 0
7a 00 4e 0 26 5b 0 27 e 00 4a 00 3a 00 70 66 7d 1f 6 1 0 2 1 0 1 00 1 00 d 00 57 00 26 47 0
27 e 00 4a 00 3a 00 70 16 7d 1f 6 1 0 1 0 0 00 00 1 00 1 00 17 0 26 72 0 27 e 00 4a 00 3a
00 70 71 7d a 2 0 0 0 0 0 0 0 1 00 3 7d 9 3 00 00 00 00 00 00 3 7d a 5 00 42 00 55 00 1 13 7d 2
4 29 2d 7d 2 7 1 6 7d 1f 6 00 00 00 00 00 1 00 56 00 60 0 26 4b 0 27 e 00 4b 00 3b 00 70 75
7d 1f 6 1 0 2 1 0 1 00 00 00 4e 00 34 0 26 c 0 27 e 00 4b 00 3b 00 70 7c 7d 1f 6 1 0 1 1 0 2
00 1 00 6b 00 3e 0 26 69 0 27 e 00 4b 00 3b 00 70 7d 37 7d a 2 0 0 0 00 00 1 00 3 7d 9 3 00 0
0 00 00 0 3 7d a 5 00 43 00 57 00 1 10 7d 1f 6 1 0 2 1 0 2 00 1 00 22 00 6b 0 26 5a 0 27 e
00 4b 00 3b 00 70 18 7d 1f 6 00 0 1 0 0 0 1 0 1 0 27 0 6 30 0 0 59 0 27 e 00 4b 00 3b 00 7
0 76 7d 1f 6 1 0 1 1 0 1 1 0 1 00 39 00 26 0 26 5b 0 27 e 00 4a 00 3a 00 70 4d 7d a 2 0 0 0
00 1 00 3 7d 9 3 00 00 00 00 00 00 3 7d a 5 00 42 00 56 00 1 10 7d 2 4 29 2d 7d 2 7 1 6 7d 1f
6 1 0 2 1 0 1 00 00 00 26 00 7a 0 26 5b 0 27 e 00 4b 00 3b 00 70 c 7d 1f 6 00 00 00 00 00 1
00 7e 00 7a 0 25 52 0 27 e 00 4b 00 3b 00 70 5d 7d 1f 6 1 0 1 1 0 1 00 00 00 26 00 4e 0 26
55 0 27 e 00 4b 00 3b 00 70 34 7d a 2 0 0 0 00 00 1 00 3 7d 9 3 00 00 00 00 3 7d a 5 00 4
3 00 56 00 1 11 7d 1f 6 1 0 1 1 0 3 00 0 0 0 3 00 0 26 4f 0 27 e 00 4a 00 3b 00 70 7a 7d
1f 6 1 0 1 1 0 1 00 00 00 1c 00 57 0 25 6b 0 27 e 00 4b 00 3b 00 70 2b 7d 1f 6 1 0 1 1 0 1 1
0 2 0 0 4e 00 70 0 26 2f 0 27 e 00 4a 00 3b 00 70 41 7d 2 0 0 0 00 00 1 00 3 7d 9 3 00 0
0 00 00 3 7d a 5 00 43 00 56 00 1 11 7d 2 4 29 2d 7d 2 7 1 6 7d 1f 6 1 0 1 1 0 2 0 0 1 00 12
00 61 0 26 55 0 27 e 00 4a 00 3a 00 70 2d 7d 1f 6 0 0 00 00 00 1 0 1 00 21 0

```

车辆底层通讯

车辆底层串口通讯主要通过循环读取一个字节数据，然后根据具体的协议，判断该字节属于那个指令，并将他放进对应指令的缓冲区中。具体解析程序如下：

数据接收部分：

```

1     while(1)
2     {
3         /*发送控制命令，速度100， 左转向10°*/
4         sendCarControlCmd( &fd, 100, 0, 10);
5         // 接收字符串
6         res = read(fd, &buf, 1);
7         if (res > 0) {
8
9             /*数据解析*/
10            packet_unpack(buf);
11        }
12
13    }

```

数据解析部分：

packet_unpack ()

```
1  /*缓存每一帧数据，并缓存下来*/
2  void packet_unpack(uint8_t _buf)
3  {
4      static uint8_t uart_flag = 1;
5      static uint8_t s_uartBuf[100];
6      static uint8_t s_len = 0;
7      if (_buf == 0xFD)
8      {
9          s_uartBuf[0] = 0xFD;
10         uart_flag = 1;
11         s_len++;
12     }
13     else
14     {
15         if (uart_flag == 1)
16         {
17             if (s_len > s_uartBuf[1] + 1)
18             {
19                 s_uartBuf[s_len] = _buf;
20                 s_len++;
21
22                 carInfoParse(s_uartBuf, s_len);
23                 uart_flag = 0;
24                 s_len = 0;
25                 return;
26             }
27             else
28             {
29                 s_uartBuf[s_len] = _buf;
30                 s_len++;
31             }
32         }
33     }
34 }
```

carInfoParse()

```
1  /*根据缓存的数据，分别存入各个缓冲区*/
2  uint8_t carInfoParse(uint8_t *_buf, uint8_t _len)
3  {
4      //根据异或校验判断数据是否存在接收错误
5      if (_buf[_len - 1] == xor_check(&_amp;_buf[2], _len - 3))
6      {
7
8          switch (_buf[2])
9          {
10             case 0x02: // 速度、转向
11
12                 g_tCarMoveInfo.x_dir = _buf[3];
13                 g_tCarMoveInfo.x_linespeed = _buf[4] << 8 | _buf[5];
14                 g_tCarMoveInfo.y_dir = _buf[6];
15                 g_tCarMoveInfo.y_linespeed = _buf[7] << 8 | _buf[8];
16                 g_tCarMoveInfo.steerDir = _buf[9];
17                 g_tCarMoveInfo.steerAngle = _buf[10] << 8 | _buf[11];
18
19                 break;
```

```
20
21 case 0x03: //电机转速
22     g_tCarMotorInfo.motor1Speed = _buf[3] << 8 | _buf[4];
23     g_tCarMotorInfo.motor2Speed = _buf[5] << 8 | _buf[6];
24     g_tCarMotorInfo.motor3Speed = _buf[7] << 8 | _buf[8];
25     g_tCarMotorInfo.motor4Speed = _buf[9] << 8 | _buf[10];
26
27     break;
28
29 case 0x04: //电压
30     g_tCarBatteryInfo.voltage = _buf[3];
31     //printf("%x\n", g_tCarBatteryInfo.voltage );
32     break;
33
34 case 0x05: //IMU pitch roll yaw
35     g_tCarImuAttitudeInfo.pitchSymbol = _buf[3];
36     g_tCarImuAttitudeInfo.pitch = _buf[4] << 8 | _buf[5];
37     g_tCarImuAttitudeInfo.rollSymbol = _buf[6];
38     g_tCarImuAttitudeInfo.roll = _buf[7] << 8 | _buf[8];
39     g_tCarImuAttitudeInfo.yawSymbol = _buf[9];
40     g_tCarImuAttitudeInfo.yaw = _buf[10] << 8 | _buf[11];
41
42 case 0x06: //imu 原始数据
43     g_tCarImuRawInfo.gyroxSymbol = _buf[3];
44     g_tCarImuRawInfo.gyrox = _buf[4] << 8 | _buf[5];
45     g_tCarImuRawInfo.gyroYSymbol = _buf[6];
46     g_tCarImuRawInfo.gyroY = _buf[7] << 8 | _buf[8];
47     g_tCarImuRawInfo.gyrozSymbol = _buf[9];
48     g_tCarImuRawInfo.gyroz = _buf[10] << 8 | _buf[11];
49     g_tCarImuRawInfo.accelxSymbol = _buf[12];
50     g_tCarImuRawInfo.accelx = _buf[13] << 8 | _buf[14];
51     g_tCarImuRawInfo.acceLySymbol = _buf[15];
52     g_tCarImuRawInfo.acceLy = _buf[16] << 8 | _buf[17];
53     g_tCarImuRawInfo.accelzSymbol = _buf[18];
54     g_tCarImuRawInfo.accelz = _buf[19] << 8 | _buf[20];
55     g_tCarImuRawInfo.quatwSymbol = _buf[21];
56     g_tCarImuRawInfo.quatw = _buf[22] << 8 | _buf[23];
57     g_tCarImuRawInfo.quatxSymbol = _buf[24];
58     g_tCarImuRawInfo.quatx = _buf[25] << 8 | _buf[26];
59     g_tCarImuRawInfo.quatySymbol = _buf[27];
60     g_tCarImuRawInfo.quaty = _buf[28] << 8 | _buf[29];
61     g_tCarImuRawInfo.quatzSymbol = _buf[30];
62     g_tCarImuRawInfo.quatz = _buf[31] << 8 | _buf[32];
63
64     break;
65
66 case 0x07: //车辆类型
67     g_tCarTypeInfo.carType = _buf[3];
68
69     break;
70
71 default:
72
73     break;
74 }
75 }
76 }
77
```



```
78  /*缓存每一帧数据，并缓存下来*/
79  void packet_unpack(uint8_t _buf)
80  {
81      static uint8_t uart_flag = 1;
82      static uint8_t s_uartBuf[100];
83      static uint8_t s_len = 0;
84      if (_buf == 0xFD)
85      {
86          s_uartBuf[0] = 0xFD;
87          uart_flag = 1;
88          s_len++;
89      }
90      else
91      {
92          if (uart_flag == 1)
93          {
94              if (s_len > s_uartBuf[1] + 1)
95              {
96                  s_uartBuf[s_len] = _buf;
97                  s_len++;
98              }
99              carInfoParse(s_uartBuf, s_len);
100             uart_flag = 0;
101             s_len = 0;
102             return;
103         }
104         else
105         {
106             s_uartBuf[s_len] = _buf;
107             s_len++;
108         }
109     }
110 }
111 }
```

小车控制部分:

```
1  void sendCarControlCmd(int *fd, int16_t _xLineSpeed, int16_t _yLineSpeed,
2  int16_t _steerAngle)
3  {
4      uint8_t _buf[13] = {0};
5      _buf[0] = 0xCD;
6      _buf[1] = 0x0a;
7      _buf[2] = 0x01;
8
9      if (_xLineSpeed > 0)
10     {
11         _buf[3] = 0x00;
12     }
13     else
14     {
15         _buf[3] = 0x01;
16     }
17
18     _buf[4] = (abs(_xLineSpeed) & 0xff00) >> 8;
19     _buf[5] = (abs(_xLineSpeed) & 0x00ff);
20 }
```

```
20     if (_yLinespeed > 0)
21     {
22         _buf[6] = 0x00;
23     }
24     else
25     {
26         _buf[6] = 0x01;
27     }
28
29     _buf[7] = (abs(_yLinespeed) & 0xff00) >> 8;
30     _buf[8] = (abs(_yLinespeed) & 0x00ff);
31
32     if (_steerAngle > 0)
33     {
34         _buf[9] = 0x00;
35     }
36     else
37     {
38         _buf[9] = 0x01;
39     }
40
41     _buf[10] = (abs(_steerAngle) & 0xff00) >> 8;
42     _buf[11] = (abs(_steerAngle) & 0x00ff);
43
44     _buf[12] = xor_check(&_amp;buf[2], _buf[11]);
45
46     write(*fd, _buf, 13);
47 }
```

完整代码如下：

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <unistd.h>
5  #include <sys/stat.h>
6  #include <sys/types.h>
7  #include <termios.h>
8  #include <string.h>
9  #include <sys/ioctl.h>
10 #include <stdint.h>
11
12 typedef struct
13 {
14     uint8_t head;
15     uint8_t frame_len;
16     uint8_t frame_code;
17     uint8_t *frame_data;
18     uint8_t frame_crc;
19 }
20 } uart_frame_t;
21
22 typedef struct
23 {
24
25     uint8_t x_dir;          //x轴方向
26     uint16_t x_linespeed;  //x轴线速度
```

```
27     uint8_t y_dir;           //Y轴方向
28     uint16_t y_lineSpeed;    //y轴线速度
29     uint8_t steerDir;        //转向方向
30     uint16_t steerAngle;     //转向角度
31
32 } carMoveInfo_t;
33
34 typedef struct
35 {
36     uint16_t motor1Speed;    // 电机1转速
37     uint16_t motor2Speed;    // 电机2转速
38     uint16_t motor3Speed;    // 电机3转速
39     uint16_t motor4Speed;    // 电机4转速
40
41 } carMotorInfo_t;
42
43 typedef struct
44 {
45     uint8_t voltage;         //电压大小
46 } carBatteryInfo_t;
47
48 typedef struct
49 {
50
51     uint8_t pitchSymbol;     //pitch 正负
52     uint16_t pitch;
53     uint8_t rollSymbol;      //roll 正负
54     uint16_t roll;
55     uint8_t yawSymbol;       //yaw 正负
56     uint16_t yaw;
57
58 } carImuAttitude_t;
59
60 typedef struct
61 {
62     uint8_t gyroXSymbol;
63     uint8_t gyroX;
64
65     uint8_t gyroYSymbol;
66     uint8_t gyroY;
67
68     uint8_t gyroZSymbol;
69     uint8_t gyroZ;
70
71     uint8_t accelXSymbol;
72     uint8_t accelX;
73
74     uint8_t accelYSymbol;
75     uint8_t accelY;
76
77     uint8_t accelZSymbol;
78     uint8_t accelZ;
79
80     uint8_t quatwSymbol;
81     uint8_t quatw;
82
83     uint8_t quatxSymbol;
84     uint8_t quatx;
```

```
85
86     uint8_t quatysymbol;
87     uint8_t quaty;
88
89     uint8_t quatzsymbol;
90     uint8_t quatz;
91
92 } carImuRaw_t;
93
94 typedef struct
95 {
96
97     uint8_t carType;
98
99 } carTypeInfo_t;
100
101 /*底盘反馈数据缓冲区*/
102 carMoveInfo_t g_tCarMoveInfo;
103 carMotorInfo_t g_tCarMotorInfo;
104 carBatteryInfo_t g_tCarBatteryInfo;
105 carImuAttitude_t g_tCarImuAttitudeInfo;
106 carImuRaw_t g_tCarImuRawInfo;
107 carTypeInfo_t g_tCarTypeInfo;
108
109 /*根据串口对应的设备, 进行修改*/
110 const char default_path[] = "/dev/ttyS6";
111 /*串口解析*/
112 void packet_unpack(uint8_t buf);
113
114 int main(int argc, char *argv[])
115 {
116
117     int fd;
118     int res;
119     char *path;
120     char buf;
121
122     //若无输入参数则使用默认终端设备
123     if (argc > 1)
124         path = argv[1];
125     else
126         path = (char *)default_path;
127
128     //获取串口设备描述符
129     printf("This is tty/usart demo.\n");
130     fd = open(path, O_RDWR);
131     if (fd < 0)
132     {
133         printf("Fail to Open %s device\n", path);
134         return 0;
135     }
136
137     struct termios opt;
138
139     //清空串口接收缓冲区
140     tcflush(fd, TCIOFLUSH);
141     // 获取串口参数 opt
142     tcgetattr(fd, &opt);
```

```
143
144 //设置串口输出波特率
145 cfsetospeed(&opt, B115200);
146 //设置串口输入波特率
147 cfsetispeed(&opt, B115200);
148 //设置数据位数
149 opt.c_cflag &= ~CSIZE;
150 opt.c_cflag |= CS8;
151 //校验位
152 opt.c_cflag &= ~PARENB;
153 opt.c_iflag &= ~INPCK;
154 //设置停止位
155 opt.c_cflag &= ~CSTOPB;
156
157 //更新配置
158 tcsetattr(fd, TCSANOW, &opt);
159
160 opt.c_iflag &= ~(INLCR); /*禁止将输入中的换行符NL映射为回车-换行CR*/
161
162 opt.c_iflag &= ~(IXON | IXOFF | IXANY); //不要软件流控制
163 opt.c_oflag &= ~OPOST;
164 opt.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG); //原始模式
165
166 tcsetattr(fd, TCSANOW, &opt); //更新终端配置
167
168 printf("device %s is set to 115200bps, 8N1\n", path);
169
170 while (1)
171 {
172     /*发送控制命令, 速度100, 左转向100 */
173     sendCarControlCmd(&fd, 100, 0, 100);
174     // 接收字符串
175     res = read(fd, &buf, 1);
176     if (res > 0)
177     {
178         /*数据解包*/
179         packet_unpack(buf);
180     }
181 }
182
183 printf("read error,res = %d", res);
184
185 close(fd);
186 return 0;
187 }
188
189 //异或校验
190 uint8_t xor_check(uint8_t *_buf, uint8_t len)
191 {
192     uint8_t xorTemp = _buf[0];
193
194     for (int i = 1; i < len; i++)
195     {
196         xorTemp ^= (*_buf + i);
197     }
198     return xorTemp;
199 }
200 }
```

```
201
202 /*根据缓存的数据，分别存入各个缓冲区*/
203 uint8_t carInfoParse(uint8_t *_buf, uint8_t _len)
204 {
205     //根据异或校验判断数据是否存在接收错误
206     if (_buf[_len - 1] == xor_check(&_amp;_buf[2], _len - 3))
207     {
208
209         switch (_buf[2])
210         {
211             case 0x02: // 速度、转向
212
213                 g_tCarMoveInfo.x_dir = _buf[3];
214                 g_tCarMoveInfo.x_linespeed = _buf[4] << 8 | _buf[5];
215                 g_tCarMoveInfo.y_dir = _buf[6];
216                 g_tCarMoveInfo.y_linespeed = _buf[7] << 8 | _buf[8];
217                 g_tCarMoveInfo.steerDir = _buf[9];
218                 g_tCarMoveInfo.steerAngle = _buf[10] << 8 | _buf[11];
219
220                 break;
221
222             case 0x03: //电机转速
223                 g_tCarMotorInfo.motor1Speed = _buf[3] << 8 | _buf[4];
224                 g_tCarMotorInfo.motor2Speed = _buf[5] << 8 | _buf[6];
225                 g_tCarMotorInfo.motor3Speed = _buf[7] << 8 | _buf[8];
226                 g_tCarMotorInfo.motor4Speed = _buf[9] << 8 | _buf[10];
227
228                 break;
229
230             case 0x04: //电压
231                 g_tCarBatteryInfo.voltage = _buf[3];
232                 //printf("%x\n", g_tCarBatteryInfo.voltage );
233                 break;
234
235             case 0x05: //IMU pitch roll yaw
236                 g_tCarImuAttitudeInfo.pitchSymbol = _buf[3];
237                 g_tCarImuAttitudeInfo.pitch = _buf[4] << 8 | _buf[5];
238                 g_tCarImuAttitudeInfo.rollSymbol = _buf[6];
239                 g_tCarImuAttitudeInfo.roll = _buf[7] << 8 | _buf[8];
240                 g_tCarImuAttitudeInfo.yawSymbol = _buf[9];
241                 g_tCarImuAttitudeInfo.yaw = _buf[10] << 8 | _buf[11];
242
243             case 0x06: //imu 原始数据
244                 g_tCarImuRawInfo.gyroxSymbol = _buf[3];
245                 g_tCarImuRawInfo.gyrox = _buf[4] << 8 | _buf[5];
246                 g_tCarImuRawInfo.gyroYSymbol = _buf[6];
247                 g_tCarImuRawInfo.gyroY = _buf[7] << 8 | _buf[8];
248                 g_tCarImuRawInfo.gyrozSymbol = _buf[9];
249                 g_tCarImuRawInfo.gyroz = _buf[10] << 8 | _buf[11];
250                 g_tCarImuRawInfo.accelxSymbol = _buf[12];
251                 g_tCarImuRawInfo.accelx = _buf[13] << 8 | _buf[14];
252                 g_tCarImuRawInfo.accelYSymbol = _buf[15];
253                 g_tCarImuRawInfo.accelY = _buf[16] << 8 | _buf[17];
254                 g_tCarImuRawInfo.accelzSymbol = _buf[18];
255                 g_tCarImuRawInfo.accelz = _buf[19] << 8 | _buf[20];
256                 g_tCarImuRawInfo.quatwSymbol = _buf[21];
257                 g_tCarImuRawInfo.quatw = _buf[22] << 8 | _buf[23];
258                 g_tCarImuRawInfo.quatxSymbol = _buf[24];
```

```
259     g_tCarImuRawInfo.quatx = _buf[25] << 8 | _buf[26];
260     g_tCarImuRawInfo.quatySymbol = _buf[27];
261     g_tCarImuRawInfo.quaty = _buf[28] << 8 | _buf[29];
262     g_tCarImuRawInfo.quatzSymbol = _buf[30];
263     g_tCarImuRawInfo.quatz = _buf[31] << 8 | _buf[32];
264     ;
265     break;
266
267     case 0x07: //车辆类型
268         g_tCarTypeInfo.carType = _buf[3];
269
270         break;
271
272     default:
273
274         break;
275     }
276 }
277 }
278
279 /*缓存每一帧数据，并缓存下来*/
280 void packet_unpack(uint8_t _buf)
281 {
282     static uint8_t uart_flag = 1;
283     static uint8_t s_uartBuf[100];
284     static uint8_t s_len = 0;
285     if (_buf == 0xFD)
286     {
287         s_uartBuf[0] = 0xFD;
288         uart_flag = 1;
289         s_len++;
290     }
291     else
292     {
293         if (uart_flag == 1)
294         {
295             if (s_len > s_uartBuf[1] + 1)
296             {
297                 s_uartBuf[s_len] = _buf;
298                 s_len++;
299
300                 carInfoParse(s_uartBuf, s_len);
301                 uart_flag = 0;
302                 s_len = 0;
303                 return;
304             }
305             else
306             {
307                 s_uartBuf[s_len] = _buf;
308                 s_len++;
309             }
310         }
311     }
312 }
313
314 /*小车控制函数，
315    fd : 串口设备的文件描述符
316    _xLinespeed: 小车x轴的线速度
```

```
317     _yLinespeed: 小车y轴的线速度
318     _steerAngle: 小车的转向速度
319
320     */
321 void sendCarControlCmd(int *fd, int16_t _xLinespeed, int16_t _yLinespeed,
322 int16_t _steerAngle)
323 {
324     uint8_t _buf[13] = {0};
325     _buf[0] = 0xCD;
326     _buf[1] = 0x0a;
327     _buf[2] = 0x01;
328
329     if (_xLinespeed > 0)
330     {
331         _buf[3] = 0x00;
332     }
333     else
334     {
335         _buf[3] = 0x01;
336     }
337
338     _buf[4] = (abs(_xLinespeed) & 0xff00) >> 8;
339     _buf[5] = (abs(_xLinespeed) & 0x00ff);
340
341     if (_yLinespeed > 0)
342     {
343         _buf[6] = 0x00;
344     }
345     else
346     {
347         _buf[6] = 0x01;
348     }
349
350     _buf[7] = (abs(_yLinespeed) & 0xff00) >> 8;
351     _buf[8] = (abs(_yLinespeed) & 0x00ff);
352
353     if (_steerAngle > 0)
354     {
355         _buf[9] = 0x00;
356     }
357     else
358     {
359         _buf[9] = 0x01;
360     }
361
362     _buf[10] = (abs(_steerAngle) & 0xff00) >> 8;
363     _buf[11] = (abs(_steerAngle) & 0x00ff);
364
365     _buf[12] = xor_check(&_amp;_buf[2], _buf[1]);
366
367     write(*fd, _buf, 13);
368 }
```




醒点机器人
CODBOT