从 0 开始单片机 ROS 小车

TAOOOOOAT

一、设计目的及系统功能

1.1 研究背景

在现有的机器人应用中，常常采用上位机控制单片机，再由单片机控制一些外设（例如：电机、传感器等）的方式来实现整体的功能。采用以上方式有以下四个优点：

1. 分工和协作：上位机和单片机各自具有不同的功能和任务。上位机通常负责高级决策、图形界面、数据处理和用户交互等复杂任务，而单片机则负责实时控制和底层硬件交互。这样的分工可以将复杂的任务分解为更小的子任务，提高系统的灵活性和可维护性。

2. 处理能力和资源限制：上位机通常具有更强大的计算能力和更丰富的资源，例如更大的内存和处理器性能。这使得上位机可以处理更复杂的算法和任务，而单片机则更适合处理实时控制和低级硬件操作。通过将部分任务分配给单片机，可以充分利用资源，提高整个系统的性能。

3. 系统稳定性和可靠性：在机器人的应用中，往往需要实时响应和高稳定性，使用单片机进行硬件控制可以提供更可靠的实时性能。单片机通常采用硬实时系统设计，具有更低的延迟和更高的可靠性，适合处理对时间要求较高的任务，如传感器数据采集和执行器控制。

4. 硬件接口和兼容性：很多机器人系统需要与多个硬件设备进行交互，如传感器、执行器、摄像头等。单片机通常具有丰富的输入输出接口和通信接口，可以方便地连接和控制各种硬件设备。上位机通过与单片机进行通信，可以实现与多个硬件设备的交互和控制，提供更灵活的系统架构。

同时，在上位机中最常用的机器人操作系统为 ROS（Robot Operating System）。其是一个开源的、灵活的机器人操作系统。它提供了一套软件工具和库，用于帮助开发者创建、控制和部署机器人系统。以下是 ROS 优点：

1. 灵活性：ROS 的设计目标之一是提供灵活性，使开发者能够轻松构建各种类型

的机器人应用。ROS 采用了模块化的架构，将功能划分为独立的节点，这些节点可以独立运行、通信和组合，以实现复杂的机器人行为。

2. 开源和共享：ROS 是一个开源项目，具有活跃的社区支持。这意味着开发者可以访问大量的开源软件包和工具，以加速机器人应用的开发。同时，ROS 鼓励开发者共享他们的代码和解决方案，促进了知识的共享和合作。

3. 强大的工具和库：ROS 提供了丰富的工具和库，用于构建机器人应用。它包括了用于传感器数据处理、导航、机器人建模和仿真、图像处理等方面的库，使开发者能够快速实现各种功能。

总体而言，ROS 的目标是提供一个强大、灵活且易于使用的机器人操作系统，它的开源性、模块化设计和丰富的工具使得机器人开发变得更加高效和便捷。由于以上原因其常常被作为机器人的操作系统。

1.2 研究目标

整体上采用上位机（树莓派，Raspberry）控制下位机（Tiva™ C Series TM4C123G LaunchPad），再通过下位机控制电机驱动模块（AT8236）和直流电机。最终实现网联、闭环控制小车，使其具备基础但是完整的轮式移动机器人整体架构。

在功能上可以实现联网远程键盘控制小车进行：前进后退、左右转、加减速和停止的功能。

在实现细节上主要有四个目标：

1. 使用 TM4C 控制电机驱动模块（AT8236）通过 PWM 驱动直流电机，并通过编码器读取电机实际转速，对电机实现 PID 闭环控制；

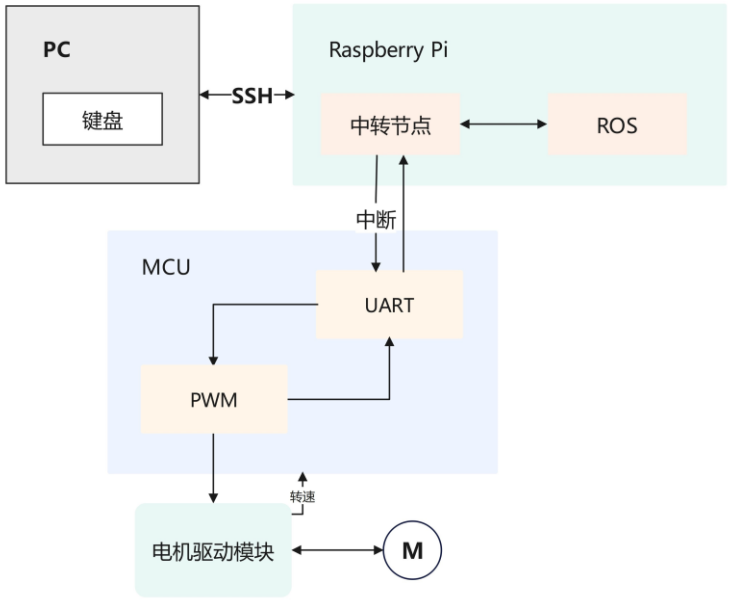
2. 实现上下位机之间的 UART（Universal Asynchronous Receiver/Transmitter）通信；

3. 在上位机部署 ROS 系统，并成功向下位机读取/发送信息；

4. 实现键盘控制节点。

二、总体设计

2.1 智能小车总体结构

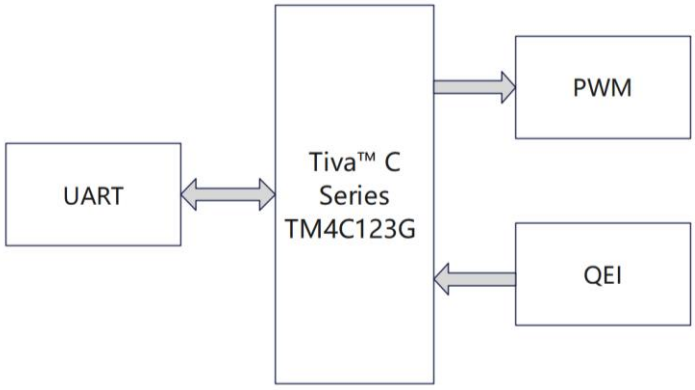


图片 1 总体结构

如图 1 所示为小车的整体结构由三个模块组成：上位机、下位机与电机模块。在 PC 端通过 SSH 工具联网控制上位机，传输控制命令。

上位机上运行 ROS 系统，接收/发送来自用户的控制信息。通过调用中转节点通过 UART 与下位机进行通讯发出期望速度，并实时接收 MCU 发出的电机转速。

下位机接收电机驱动模块发出的编码器信号，并通过 PID 计算得出当前 PWM 的占空比，并发送给电机驱动模块。接收上位机的期望速度。向上位机实时发出电机转速信息。如下图为单片机的方框图。

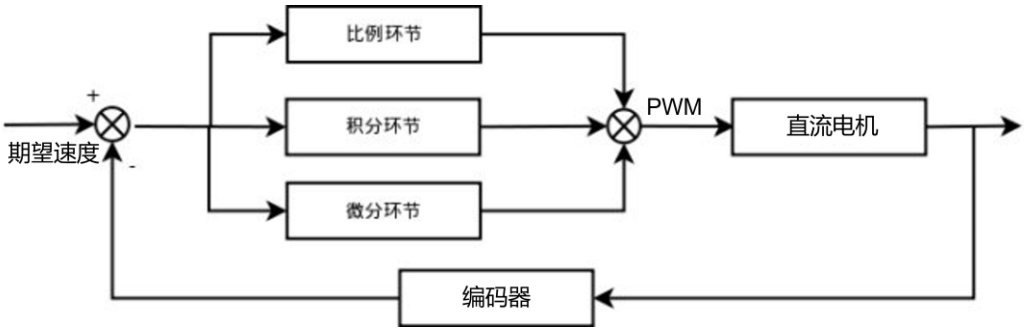


图片 2

电机模块中：

1. 电机驱动模块接收 PWM，通过 H 桥电路对电机输入电压进行控制。接受编码器发出的信号，并发送到 MCU。
2. 电机随着输入电压的变化转速改变，同时编码器发送脉冲信号。

2.2 控制系统设计方案



图片 3 PID 控制

如图 3 所示为整体的控制方案，采用增量式 PID 闭环反馈控制。其中速度作为被控对象，通过改变 PWM 占空比控制直流电机转速，编码器作为测量元件测量电机实际转速。

下式为增量式 PID，其中 $\Delta u(k)$ 为输出变化量，K_p 为比例系数，K_i 为积分系数 K_d 为微分系数，$e(k-1)$ 为上一次的目标和实际的误差值，$e(k)$ 为这次的目标和实际的误差值。

$$\Delta u(k) = K_p \times e(k-1) + K_i \times e(k) + K_d \times \left(e(k) - 2e(k-1) + e(k-2)\right) \qquad (1)$$

三、硬件设计

3.1 主要元件选型

1. 直流电机（2 个，含编码器）
2. 有刷的电机驱动器件 AT8236（2 个）组成 D107A 模块

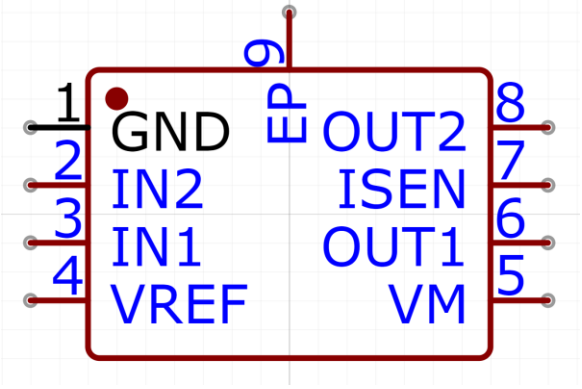
3. 单片机：Tiva™ C Series TM4C123G LaunchPad

4. 上位机：Raspberry Pi 4

3.2 主要元件工作原理

AT8236 是一款直流有刷的电机驱动器件，能够以高达 6A 的峰值电流双向控制电机。利用电流衰减模式，可通过对输入信号进行脉宽调制（PWM）来控制电机的转速，同时具备低功耗休眠功能。芯片原理图如图 4 所示。

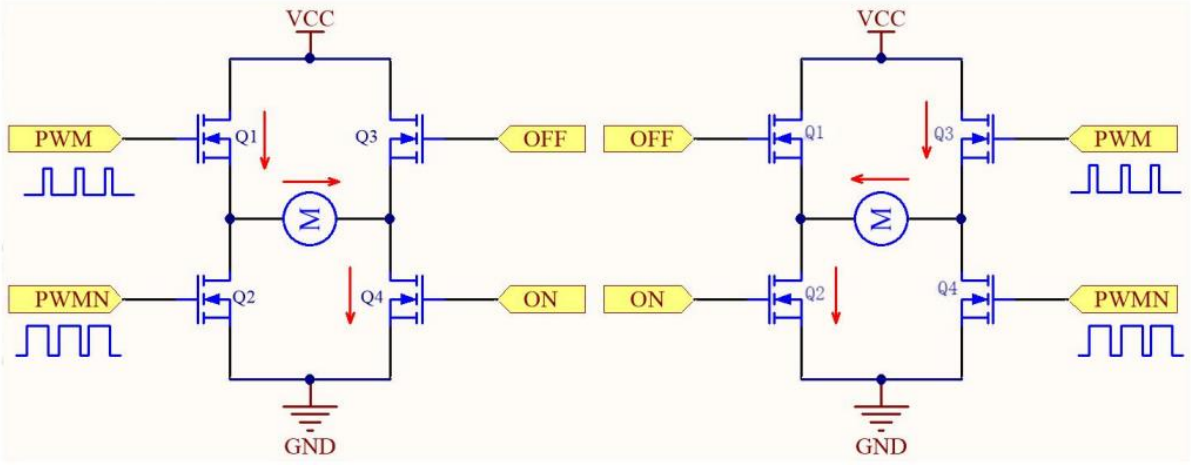
可以实现单通道 H 桥电机驱动，并且实现 PWM 接口控制。



图片 4 AT8236 芯片原理图

IN1	IN2	功能
PWM	0	正转 PWM，快衰竭
1	PWM	正转 PWM，慢衰竭
0	PWM	反转 PWM，快衰竭
PWM	1	反转 PWM，慢衰竭

表格 1 AT8236PWM 调速



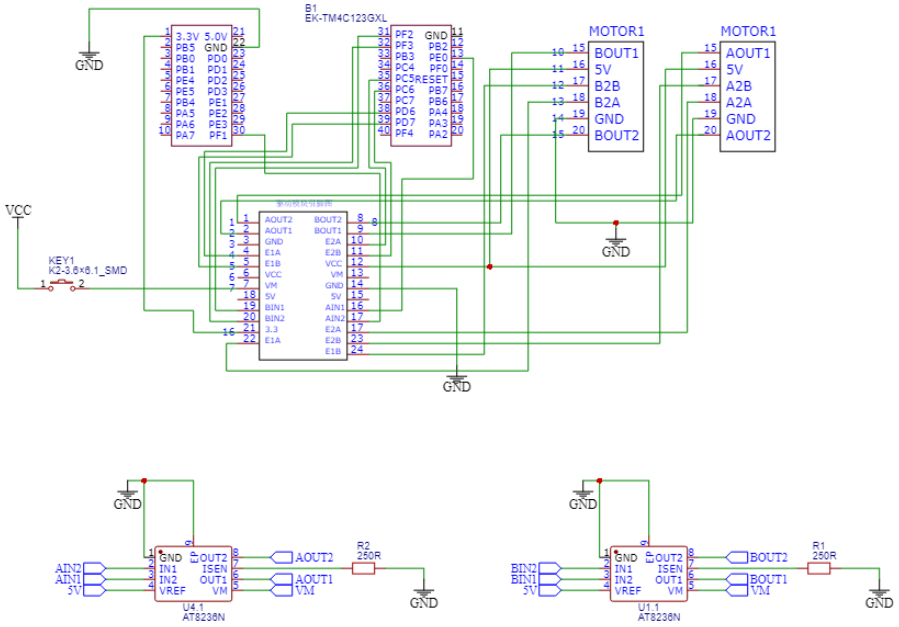
图片 5 单极性模式

如表格 1 为 AT8236 PWM 调速时的输入表，图 5 为单极性模式下的原理图，当 PWM 为高电平时：MOS 管 Q1 和 Q4 导通，MOS 管 Q2 和 Q3 截止，电流从电源正极经过 Q1，从坐到右流过电机，然后经过 Q4 回到电源负极。

3.3 电路设计与接线

Signal Name	MCU	AT8236
PWM1_PWM2 OutputA	PF0	AIN1
PWM1_PWM2 OutputB	PF1	AIN2
PWM1_PWM3 OutputA	PF2	BIN1
PWM1_PWM3 OutputB	PF3	BIN2
QEI0 PhA	PD6	E1A
QEI0 PhB	PD7	E1B
QEI1 PhA	PC5	E2A
QEI1 PhB	PC6	E2B

表格 2 接线表

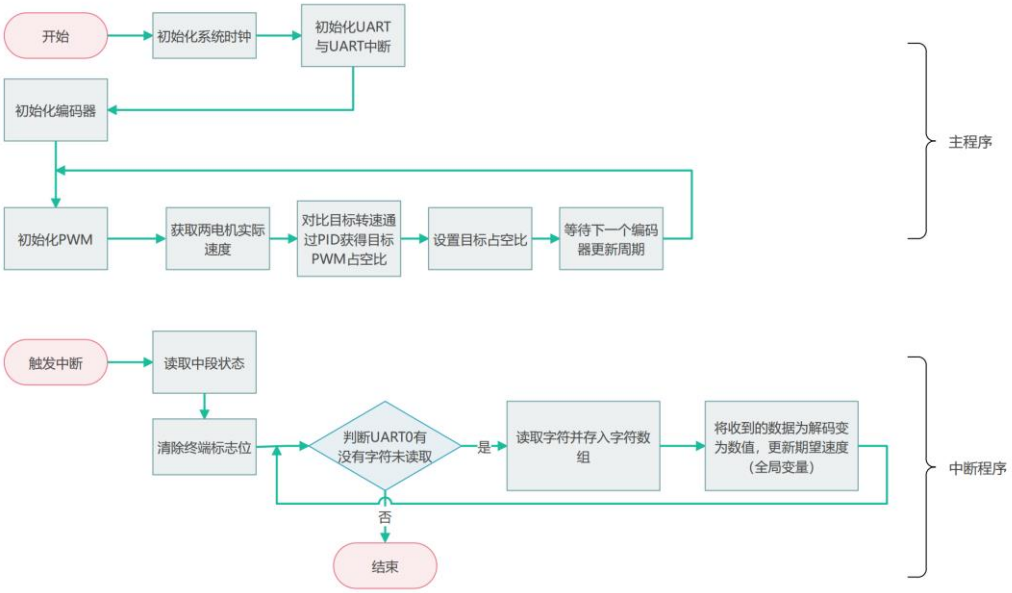


图片 6

如上表 2 和图 6 所示为实际的电路连接示意图

四、软件设计

4.1 MCU 软件设计



图片 7 MCU 部分程序流程图

如图片 7 所示为单片机的程序流程图。

以下为 main.c 中的代码。

```
/**
/**
 * main.c
 */
#include <stdint.h>
#include <stdio.h>
#include "buct_hal.h"
#include "pid.h"
#include <stdbool.h>
#include "uartstdio.h"
volatile int32_t DesierdSpeed1=50, DesierdSpeed2=50;
int32_t ActualSpeed1, ActualSpeed2;
volatile uint32_t Speed1 = 500;
volatile uint32_t Speed2 = 500;
float Velcity_Kp = 0.1, Velcity_Ki = 1.1, Velcity_Kd = 0;
int main()
{
    USART_Config();
    initQEI0();        // Initialize Quadrature Encoder Interface 0
    initQEI1();        // Initialize Quadrature Encoder Interface 1
    initCarPWM();      // Initialize Car PWM signals
    while (1)
    {
        ActualSpeed1 = getMotor1Velocity(); // Get the actual speed from QEI0
module
        ActualSpeed2 = getMotor2Velocity(); // Get the actual speed from QEI1
module
        Speed1 = Velocity_FeedbackControl1(DesierdSpeed1, ActualSpeed1);
        Speed2 = Velocity_FeedbackControl2(-1*DesierdSpeed2, ActualSpeed2);
        UARTprintf("%i,%i,%i\r\n",DesierdSpeed1,ActualSpeed1,-1*ActualSpeed2);
        setMotor1(Speed1); // Set the speed of Motor 1
        setMotor2(Speed2); // Set the speed of Motor 2
        waitQEI0Speed();   // Wait for the next velocity reading from QEI0 to
complete
        waitQEI1Speed();   // Wait for the next velocity reading from QEI1 to
complete
    }
}
```

其中主要以全局变量 **DesierdSpeed1,DesierdSpeed2** 作为全局变量来记录期望速度。其将有可能在中断函数 **void USART_Config(void);** 中被修改。中断函数为：

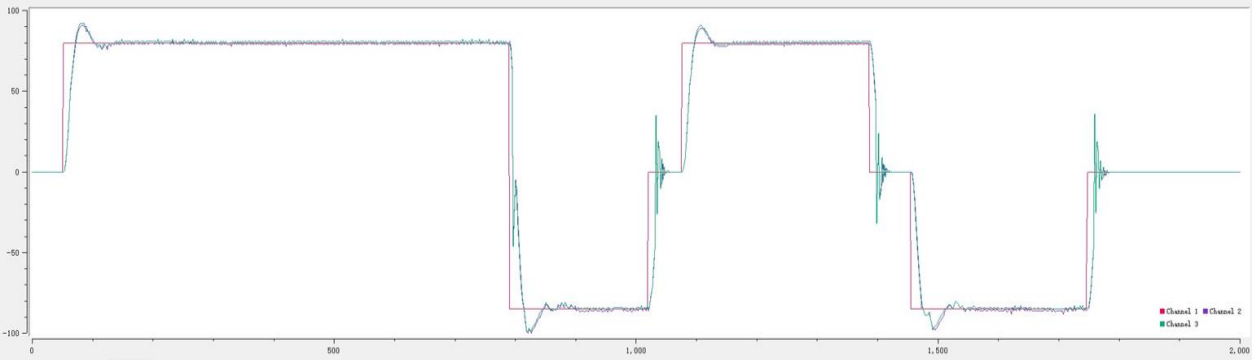
```
/**
 * muart.c
 */
extern int32_t DesierdSpeed1;
extern int32_t DesierdSpeed2;
void USART0_IRQHandler(void)
{
    uint8_t i = 0;
    uint8_t num_Arry[4];
    uint32_t re_buf;
    // 读取中断状态
    uint32_t status = UARTIntStatus(UART0_BASE, true);
    // 清除中断标志位
    UARTIntClear(UART0_BASE, status);
    // 判断UART0有没有字符未读取
    while (UARTCharsAvail(UART0_BASE))
    {
        // 如果有字符为读取就取出，使用UARTCharGetNonBlocking防止等待
        re_buf = UARTCharGetNonBlocking(UART0_BASE);
        // 将读取出的字符存入数组
        num_Arry[i] = (uint8_t)re_buf;
        i++;
    }
    DesierdSpeed1 = ArrayToVariable(num_Arry, 2); /字符数组解码
    DesierdSpeed2 = ArrayToVariable(num_Arry + 2, 2);
   UARTprintf("%i, %i \n", DesierdSpeed1, DesierdSpeed2);
}
```

该函数用于接受处理 UART 发出的目标速度，更行全局变量 DesierdSpeed1，DesierdSpeed2。

以下为 PID 控制函数（此处之展示了左轮 PID，右轮同理）：

```
/**
 * pid.c
 */
#include "pid.h"
// 外部变量 extern说明改变量已在其它文件定义
extern float Velcity_Kp, Velcity_Ki, Velcity_Kd; // 相关速度PID参数
/*************************************************************************
```

```
函数功能：速度闭环PID控制(实际为PI控制)
入口参数：目标速度 当前速度
返回  值：速度控制值
根据增量式离散PID公式
ControlVelocity+=Kp[e(k)- e(k-1)]+Ki*e(k)+Kd[e(k)-2e(k-1)+e(k-2)]
e(k)代表本次偏差
e(k-1)代表上一次的偏差
ControlVelocity代表增量输出
在我们的速度控制闭环系统里面，只使用PI控制
ControlVelocity+=Kp[e(k)- e(k-1)]+Ki*e(k)
**************************************************************************/
int Velocity_FeedbackControl1(int TargetVelocity, int CurrentVelocity)
{
    int Bias;                                       // 定义相关变量
    static int ControlVelocity1 = 500, Last_bias1;
    // 静态变量，函数调用结束后其值依然存在
    Bias = TargetVelocity - CurrentVelocity;        // 求速度偏差
   ControlVelocity1 += Velcity_Kp * (Bias - Last_bias1) + Velcity_Ki * Bias;
   // 增量式PI控制器
   // Velcity_Kp*(Bias-Last_bias) 作用为限制加速度
    // Velcity_Ki*Bias 速度控制值由Bias不断积分得到 偏差越大加速度越大
   Last_bias1 = Bias;
   return ControlVelocity1; // 返回PWM控制值
}
```



图片 8 控制曲线

通过静态变量 **ControlVelocity1**，保存上一次 PWM 占空比，每次对其进行累加。完成代码后需要对参数进行调整，如图 8 所示通过串口绘图软件对目标速度（红线），左轮实际速度（紫线），右轮实际速度（绿线）进行绘制，最终使得控制稳定且调整速度块。

4.2 下位机通信设计

整体上采用 UART 进行通信，其中帧格式采用以下三种方法：

- 数据位采用创新的方法，将在下文介绍；
- 采用偶校验；
- 采用两位停止位。

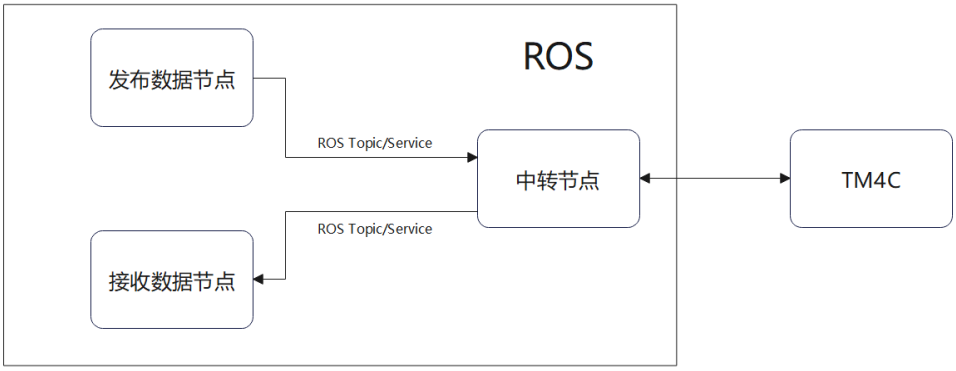
其中数据位的格式如图 9 所示，数据位由 4 个 Byte 组成，第一位和第三位为左右轮方向判断位（0xff 为反转，0x00 为正转），第二位和第四位为左右轮转速的绝对值。如图所示的情况是原地向右快速旋转时的数据位格式。以下提供的代码为速度编码和解码的函数：

图片 9 数据位格式

```
/**
 * muart.c
 */

int32_t ArrayToVariable(uint8_t *Array, uint8_t Length)
{
    int64_t Variable = 0;

    if (Length == 2)
    {
        Variable = (uint16_t)Array[1];
        if ((uint16_t)Array[0] > 0x88)
            Variable *= -1;
    }
    // else if(Length == 4)
    // {
    //     Variable = (((uint32_t)Array[0] << 24) + ((uint32_t)Array[1] << 16)
    //             + ((uint32_t)Array[2] << 8)  + ((uint32_t)Array[3]));
    // }
    return Variable;
}
```

```
void VariableToArray(uint8_t *Array, int16_t a, int16_t b)
{
    if (a >= 0)
    {
        *(Array) = 0x00;
        *(Array + 1) = (uint8_t)(a);
    }
    else
    {
        *(Array) = 0xff;
        a = abs(a);
        *(Array + 1) = (uint8_t)(a&0xff);
    }
    if (b >= 0)
    {
        *(Array + 2) = 0x00;
        *(Array + 3) = (uint8_t)(b);
    }
    else
    {
        *(Array + 2) = 0xff;
        b = abs(b);
        *(Array + 3) = (uint8_t)(b&0xff);
    }

}
```

4.3 上位机通讯设计与 ROS



图片 10

如图 10 所示为 ROS 与 TM4C 进行通讯的结构。其中中转节点通过 termios.h 调用 Linux 的串行端口与 TM4C 进行 UART 通讯，由于基本原理与 MCU 端类似不再展示。

下面是中转节点的代码：

```
// listener.cpp
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/UInt32.h"
#include <stdint.h>
#include <stdio.h>
using namespace std;
#include <iostream>

#ifdef __cplusplus
extern "C"
{
#endif
#include "c_uart.h"
#ifdef __cplusplus
}
#endif

void SpeedCallback(const std_msgs::UInt32::ConstPtr &msg)
{
  ROS_INFO("I heard: [%x]", msg->data);
  uartWrite(msg->data);
}

int main(int argc, char **argv)
```

```
{
  ros::init(argc, argv, "listener");
  if(!uartInit()) return 0;
  ROS_INFO("Start");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("desire_speed", 5, SpeedCallback);

  // ros::Publisher pub = n.advertise<std_msgs::UInt32>("true_speed", 1000);
  ros::Rate loop_rate(10);
  while (ros::ok())
  {
    ros::spinOnce();
    // std_msgs::UInt32 t_speed;
    Speed s = uartRead();
    // ROS_INFO("True Speed is: [%i,%i]", s.Speed1, -1*s.Speed2);
    loop_rate.sleep();
  }
  uartClose();
  return 0;
}
```

其中 uartRead()和 uartWrite()分别为读取和发送 URAT 消息在 c_uart.c 中定义。

下面为键盘节点的部分代码：

```
# teleop_twist_keyboard.py
# 读取按键循环
while not rospy.is_shutdown():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    # 不产生回显效果
    old_settings[3] = old_settings[3] & ~termios.ICANON & ~termios.ECHO
    try:
        tty.setraw(fd)
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

    if ch == 'w':
        v = speed
        turn = 0
        stop = 0
    elif ch == 's':
```

```
        v = -1 * speed
        turn = 0
        stop = 0
    elif ch == 'a':
        v = speed
        turn = 1
        stop=0
    elif ch == 'd':
        v = speed
        turn = -1
        stop=0
    elif ch == 'e':
        speed += 15
    elif ch == 'c':
        speed -= 5
    elif ch == 'z':
        exit()
    elif ch == 'q':
        stop_robot()
        v = 0
        turn = 0
        speed = 20
        stop=1
    else:
        pass
    if speed > 50:
        speed = 50
    if speed < 20:
        speed = 20
    print("speed: %i ",speed)
    print("turn: %i ",turn)
    print("stop: %i ",stop)
    if stop==1:
        speed = 0

    if turn == 0:
        if v >= 0:
            cmd.data = (speed << 16) | (speed)
        else:
            cmd.data = 0xff00ff00 | ((speed << 16) | (speed))
    elif turn == 1:
        if v >= 0:
```

```
            cmd.data =0xff000000| (speed << 16) | (speed)
        else:
            cmd.data = 0x0000ff00 | ((speed << 16) | (speed))
    else:
        if v >= 0:
            cmd.data =0x0000ff00| (speed << 16) | (speed)
        else:
            cmd.data = 0xff000000 | ((speed << 16) | (speed))

        # 发送消息
    pub.publish(cmd)
    rate.sleep()
```

本段代码中读取键盘的按键触发中断，将帧的数据位发送给中转节点。

五、设计结果及小结

总体上采用上位机（树莓派，Raspberry）控制下位机（Tiva™ C Series TM4C123G LaunchPad），再通过下位机控制电机驱动模块（AT8236）和直流电机。最终实现网联、闭环控制小车，使其具备基础但是完整的轮式移动机器人整体架构。最终达到了联网用键盘控制移动机器人的设计目标。

在单片机方面学习了 PWM 模块，UART 中断，编码器的使用和 UART 中断的使用。更全面完整的了解掌握了单片机对直流电机的闭环控制，同时深化了在单片机入门课程和自动控制原理中学到的内容。

另一方面，第一次独立实现上位机 ROS 系统与下位机的通讯，弥补了我三年以来对软硬件结合和上下位机通讯的知识空白。为我以后的机器人学习打下了很好的基础。