

项目结构：

```
|— USER/
|   |— main.c          【主程序入口】
|   |— AllHeader.h     【统一头文件包含】
|   |— stm32f10x_it.c   【中断服务函数】
|
|— BSP/                【板级支持包】
|   |— bsp.c/h         【基础系统初始化】
|   |— debug_interface.c/h 【调试接口模块】
|   |— debug_tools.c/h  【调试工具模块】
|   |— system_manager.c/h 【系统管理模块】
|   |
|   |— Delay/          【延时功能】
|   |   |— delay.c/h
|   |
|   |— Timer/          【定时器模块】
|   |   |— bsp_timer.c/h
|   |
|   |— Usart1/         【串口模块】
|   |   |— usart.c/h    【USART1调试串口】
|   |   |— usart3.c/h   【USART3 ROS通信】
|   |
|   |— motor_model/    【电机相关模块】
|   |   |— app_motor_usart.c/h 【电机串口应用层】
|   |   |— bsp_motor_usart.c/h 【电机串口底层】
|   |   |— kinematics.c/h     【运动学模型】
|   |   |— motor_control.c/h  【电机控制】
|   |   |— pid_config.c/h     【PID控制器】
|   |   |— ros_interface.c/h  【ROS接口】
```

太棒了！听到电机终于能正常转动的消息，我由衷地为你感到高兴。我们一起走过的这段调试历程非常曲折，但每一步都让我们对系统有了更深刻的理解。现在，我们来系统地总结一下，并规划后续的步骤。

一、已解决的关键问题总结

我们遇到的问题是典型的嵌入式系统“bug链”，环环相扣，一个问题掩盖另一个问题。

1. 调试器“抢跑”问题 (dbg_cmd_ready 标志位失效)

- **现象：**进入了 `USART1_IRQHandler` 中断，但用 `if` 判断 `RXNE` 标志位时却发现它已经被清零。
- **根源：**Keil/IAR 等 IDE 的实时观察窗口 (Live Watch) 为了刷新变量，会在后台自动读取 `USART1->DR` 寄存器，这个读操作会抢先一步清除 `RXNE` 标志位。
- **解决方案：**修改中断服务函数，在函数入口处立即将 `SR` 和 `DR` 寄存器一次性读到局部变量中，后续所有操作都基于这些局部变量，避免了与调试器的竞争。

2. 主循环阻塞问题 (系统卡死)

- **现象：**系统启动后只运行了一瞬间就失去响应，发送任何调试命令都无效。

- **根源：** `ROS_Process_Command` 函数中存在阻塞式 API `USART3_Receive_Byte()`。该函数会一直 `while` 循环，等待来自 ROS 主机（`USART3`）的数据。由于没有 ROS 主机连接，程序就永远卡死在这里。
- **解决方案：** 将 `ROS_Process_Command` 改造为非阻塞模式。为此，我们新增了 `USART3_Is_Data_Available()` 和 `USART3_Try_Receive_Byte()` 函数，使得主程序可以“轮询”`USART3` 是否有数据，而不是“死等”。

3. 定时器中断与主循环的逻辑冲突 (电机不按指令转动)

- **现象：** 在 `while(1)` 循环外直接调用 `Contrl_Speed` 电机能动，但在循环内通过调试命令调用就不动或行为异常。
- **根源：** `TIM3_IRQHandler` 定时器中断服务函数中，存在一行错误的 `Contrl_Pwm(...)` 调用。这个调用会以固定的高频率向电机驱动板发送 PWM 控制指令，与主循环中发送的速度控制指令（`$spd:...#`）发生冲突和覆盖，导致电机无法正常执行速度指令。
- **解决方案：** 注释掉 `TIM3_IRQHandler` 中所有不应该在中断中执行的电机控制函数，只保留最核心的 tick 计数功能。

4. `HAL_GetTick` 函数未定义 (编译链接错误)

- **现象：** 修复其它问题后，出现 `HAL_GetTick` 链接错误。
- **根源：** 在之前的修改中，我们意外地将 `bsp_timer.c` 中提供 `HAL_GetTick` 功能的全局变量 `g_ms_tick` 和其实现函数一并删除了。
- **解决方案：** 恢复 `g_ms_tick` 变量和 `HAL_GetTick` 函数的定义与声明，并确保 `TIM3_IRQHandler` 能正确地对其进行累加。

我们通过**探针法**、**逻辑推理**和**对嵌入式系统底层机制（中断、调试器行为）的理解**，层层剥茧，最终定位并解决了这些深层次的问题。

二、后续测试与完善计划

现在你的底层固件已经基本稳定，接下来可以分模块进行验证和完善。

Step 1: 完善运动学与里程计 (`kinematics.c`)

- **目标：** 验证编码器数据能否正确地上报，并转换成机器人的位姿（里程计）。
- **测试步骤：**

1. 保持在 `DEBUG_MOTOR_TEST` 模式。
2. 在 `system_manager.c` 的 `Handle_Motor_Data` 函数中，**取消**对以下 `printf` 的注释：

```
1  if(Debug_Get_Mode() != DEBUG_NONE)
2  {
3      printf("Speeds(mm/s): LF=%.1f, RF=%.1f, LR=%.1f, RR=%.1f\r\n",
4              g_Speed[0], g_Speed[1], g_Speed[2], g_Speed[3]);
5
6      RobotPose pose;
7      Kinematics_GetPose(&pose);
8      printf("Pose: x=%.2f, y=%.2f, theta=%.2f\r\n",
9              pose.x, pose.y, pose.theta);
10 }
```

3. 发送 `$motor:f,100#` 让小车前进。

4. 观察现象：

- `speeds` 是否能打印出四个轮子的速度？数值是否接近？
- `Pose` 中的 `x` 值是否在持续增加？`y` 和 `theta` 是否基本保持为 0？

5. 让小车左转 `$motor:1,100#`，观察 `theta` 是否在变化。

- **完善：**如果发现里程计数据不准（比如前进时 `theta` 也在变），你可能需要微调 `kinematics.c` 中的轮距 `wheel_distance` 和轴距 `wheel_base` 参数。

Step 2: 测试 ROS 通信协议 (`ros_interface.c`)

- **目标：**确保 STM32 能正确地将里程计数据打包成 ROS 消息，并通过 `USART3` 发送出去。
- **测试步骤：**
 1. 找一个 USB 转 TTL 模块，连接到 STM32 的 `USART3` (PB10, PB11)。
 2. 在电脑上打开一个串口助手（比如 `minicom` 或 `putty`），监听该串口。
 3. 在 `ros_interface.c` 的 `ROS_Send_Status` 函数中，取消对 `printf` 的注释，同时也可以 在 `Send_Frame` 中用 `USART1` 打印将要发送的数据，用于对比。
 4. 让小车运动。
- 5. **观察现象：**
 - 监听 `USART3` 的串口助手是否收到了以 `0xAA` 开头、`0x55` 结尾的十六进制数据流？
 - 这些数据流的内容是否与 `USART1` 打印出的位姿和速度信息相符？
- **完善：**如果收不到数据或数据格式错误，需要检查 `Send_Frame` 函数的打包逻辑和 `USART3` 的发送函数。

Step 3: 恢复超时急停逻辑

- **目标：**确保在长时间未收到 ROS 指令时，电机自动停止，保证安全。
 - **测试步骤：**
 1. 在 `ros_interface.c` 的 `ROS_Process_Command` 函数中，取消对超时检查代码的注释：

```
1  if(HAL_GetTick() - last_cmd_time > CMD_TIMEOUT)
2  {
3      // 超时停止电机
4      Ctrl_Speed(0, 0, 0, 0);
5      system_status.error_code |= ERR_TIMEOUT;
6  }
```
 2. 同时，在 `Handle_Motion_Control` 中也恢复对 `last_cmd_time` 的更新：

```
1  last_cmd_time = HAL_GetTick();
```
 3. **完善：**目前 `last_cmd_time` 只在收到 `CMD_MOTION_CONTROL` 时更新，你可以考虑在收到任何有效 ROS 指令时都更新它。
-

三、编写 ROS 端驱动代码 (stm32_serial_driver.py)

在你确认了 STM32 能通过 USART3 正确发送打包好的里程计数据后，就可以开始编写 ROS 端的驱动节点了。这是一个基本的 Python 脚本框架：

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import rospy
5  import serial
6  import struct
7  from nav_msgs.msg import Odometry
8  from geometry_msgs.msg import Twist, TransformStamped
9  import tf
10
11 # --- 常量定义，必须与 STM32 中的 ros_interface.h 完全一致 ---
12 FRAME_HEADER = 0xAA
13 FRAME_TAIL = 0x55
14 CMD_POSITION_UPDATE = 0x02
15 CMD_STATUS = 0x05
16 # ... 其他命令 ...
17
18 class STM32Driver:
19     def __init__(self):
20         rospy.init_node('stm32_driver', anonymous=True)
21
22         # 从参数服务器获取串口号和波特率，提供默认值
23         port = rospy.get_param('~port', '/dev/ttyUSB0')
24         baudrate = rospy.get_param('~baudrate', 115200)
25
26         # 初始化串口
27         try:
28             self.ser = serial.Serial(port, baudrate, timeout=0.1)
29             rospy.loginfo("Serial port opened: %s at %d", port, baudrate)
30         except serial.SerialException as e:
31             rospy.logerr("Failed to open serial port: %s", e)
32             return
33
34         # ROS 发布者和订阅者
35         self.odom_pub = rospy.Publisher('odom', Odometry, queue_size=10)
36         self.cmd_vel_sub = rospy.Subscriber('cmd_vel', Twist,
37 self.cmd_vel_callback, queue_size=1)
38
39         # TF 广播
40         self.tf_broadcaster = tf.TransformBroadcaster()
41
42         # 主循环
43         self.rate = rospy.Rate(100) # 100 Hz
44         self.buffer = bytearray()
45
46         rospy.on_shutdown(self.shutdown)
47
48     def run(self):
49         while not rospy.is_shutdown():
50             if self.ser.in_waiting > 0:
```

```

50         self.buffer.extend(self.ser.read(self.ser.in_waiting))
51         self.parse_buffer()
52         self.rate.sleep()
53
54     def parse_buffer(self):
55         """
56         在缓冲区中查找并解析完整的数据帧
57         """
58         while len(self.buffer) >= 5: # 最小帧长度
59             (Header+Cmd+Len+Checksum+Tail)
60             try:
61                 # 查找帧头
62                 start_index = self.buffer.index(FRAME_HEADER)
63
64                 # 如果帧头不是第一个字节, 丢弃之前的数据
65                 if start_index > 0:
66                     self.buffer = self.buffer[start_index:]
67
68                 # 检查是否有足够的数据构成一个完整的帧
69                 if len(self.buffer) < 3:
70                     break
71
72                 cmd_id = self.buffer[1]
73                 length = self.buffer[2]
74                 frame_len = 3 + length + 2 # Header,Cmd,Len + Data +
75                                         Checksum,Tail
76
77                 if len(self.buffer) < frame_len:
78                     break # 数据不完整, 等待下一次接收
79
80                 # 提取完整的一帧
81                 frame = self.buffer[:frame_len]
82
83                 # 校验帧尾
84                 if frame[-1] != FRAME_TAIL:
85                     rospy.logwarn("Invalid frame tail. Discarding.")
86                     self.buffer = self.buffer[1:] # 从下一个字节开始重新寻找帧头
87                     continue
88
89                 # TODO: 校验和 (Checksum)
90
91                 # 根据命令ID解析数据
92                 if cmd_id == CMD_STATUS: # 假设我们解析状态帧
93                     # 根据 system_status 结构体的定义来解包
94                     # struct.unpack('<fff fff 4f B I', frame[3:3+length])
95                     # '<' 代表小端模式, 'f'代表float, 'B'代表uint8, 'I'代表
96                     uint32
97
98                     # 具体的格式字符串需要你根据 SystemStatus 结构体精确定义
99                     self.publish_odometry(frame[3:3+length])
100
101                 # 处理完一帧后, 从缓冲区移除
102                 self.buffer = self.buffer[frame_len:]
103
104             except ValueError:
105                 # 缓冲区中没有找到帧头, 清空缓冲区
106                 self.buffer = bytearray()

```

```

103         break
104
105     def publish_odometry(self, data):
106         """
107         解析数据并发布 odom 消息和 tf 变换
108         """
109         # ---你需要根据你的SystemStatus结构体精确地解包---
110         # 这是一个示例，假设前3个float是x,y,theta
111         try:
112             x, y, theta, vx, vth = struct.unpack('<ffffff', data[:20]) # 假设
解包出这些数据
113         except struct.error as e:
114             rospy.logerr("Failed to unpack odometry data: %s", e)
115             return
116
117         current_time = rospy.Time.now()
118
119         # 1. 发布 TF
120         self.tf_broadcaster.sendTransform(
121             (x, y, 0.0),
122             tf.transformations.quaternion_from_euler(0, 0, theta),
123             current_time,
124             "base_link",
125             "odom"
126         )
127
128         # 2. 发布 Odometry 消息
129         odom = Odometry()
130         odom.header.stamp = current_time
131         odom.header.frame_id = "odom"
132         odom.child_frame_id = "base_link"
133
134         odom.pose.pose.position.x = x
135         odom.pose.pose.position.y = y
136         odom.pose.pose.orientation =
tf.transformations.quaternion_from_euler(0, 0, theta)
137
138         odom.twist.twist.linear.x = vx
139         odom.twist.twist.angular.z = vth
140
141         self.odom_pub.publish(odom)
142
143     def cmd_vel_callback(self, msg):
144         """
145         接收 /cmd_vel 指令，并将其打包发送给 STM32
146         """
147         # TODO: 将 msg.linear.x 和 msg.angular.z 打包成STM32能识别的帧格式
148         # 例如: struct.pack('<Bf', CMD_MOTION_CONTROL, msg.linear.x)
149         # 然后通过 self.ser.write() 发送出去
150         pass
151
152     def shutdown(self):
153         rospy.loginfo("Shutting down STM32 driver. Closing serial port.")
154         # 发送停止指令
155         # self.ser.write(...)
156         self.ser.close()

```

```
157
158
159 if __name__ == '__main__':
160     try:
161         driver = STM32Driver()
162         driver.run()
163     except rospy.ROSInterruptException:
164         pass
```

使用这个脚本的步骤:

1. 把它保存到你的 ROS 包的 `scripts` 文件夹下, 并给它执行权限 (`chmod +x stm32_serial_driver.py`).
2. 创建一个 `launch` 文件来启动它, 并设置好串口参数。
3. 运行 `rostopic echo /odom` 来查看里程计数据。
4. 运行 `roslaunch rviz rviz`, 添加 "TF" 和 "Odometry" 可视化, 你应该能看到机器人的位置在变化。
5. 发布 `cmd_vel` 指令 (`rostopic pub /cmd_vel geometry_msgs/Twist "linear: {x: 0.1}"`), 并实现 `cmd_vel_callback`, 让 ROS 来控制你的机器人。

我们一起走过了最艰难的底层调试阶段, 现在你的项目已经进入了更上层的应用开发。祝贺你! 希望这份总结和规划能对你有所帮助。