



微型机控制技术

PJ1报告

机器人通信协议设计

马逸君 17300180070

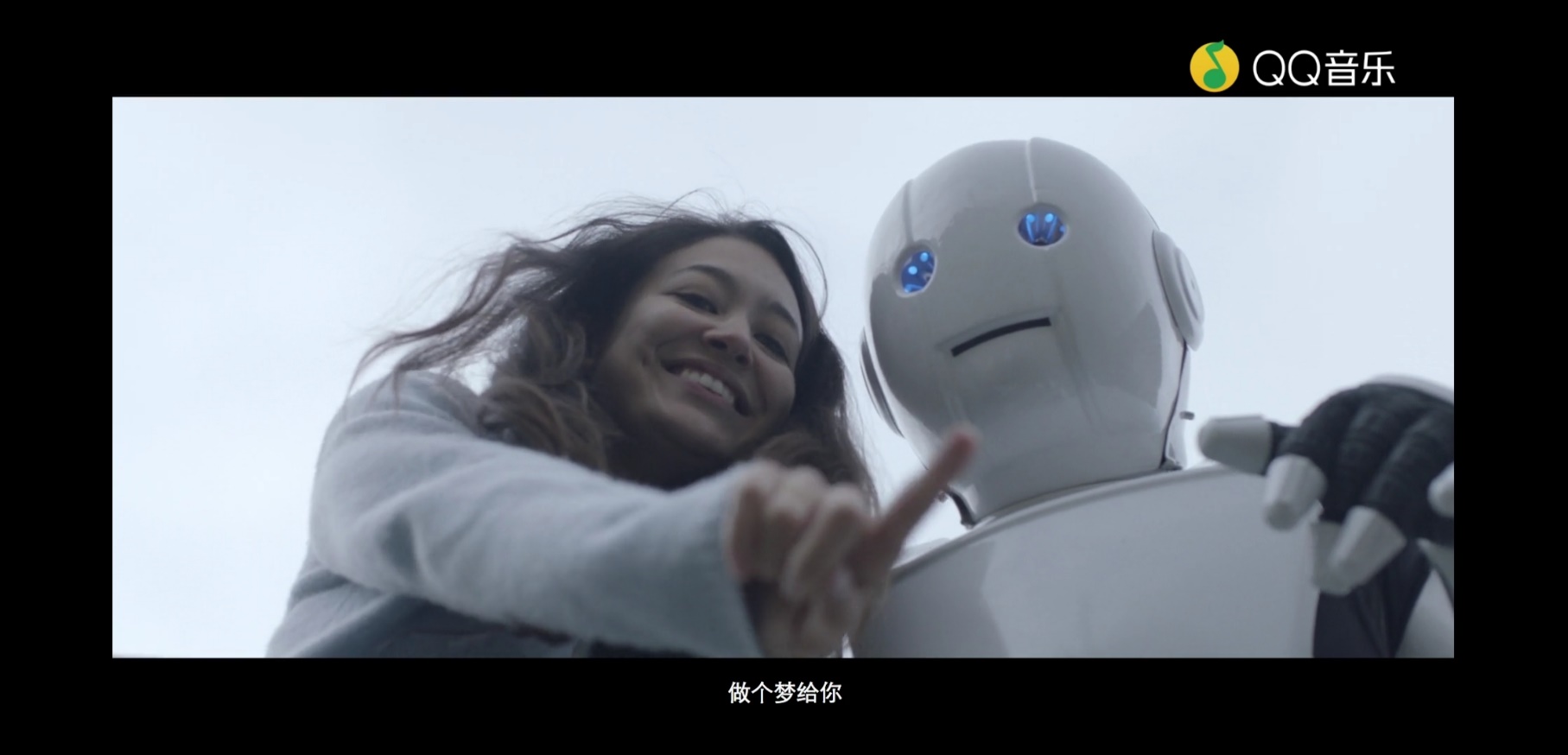
胡永祥 17307130026

2020年3月

背景

机器人（robot）是一种能够半自主或全自主工作的智能机器。历史上最早的机器人见于隋炀帝命工匠所营造的木偶机器人，施有机关，有坐、起、拜、伏等能力。机器人具有感知、决策、执行等基本特征，可以辅助甚至替代人类完成危险、繁重、复杂的工作，提高工作效率与质量,服务人类生活,扩大或延伸人的活动及能力范围。

**机器人正在走进我们的生活**，各种扫地机器人、机器人管家、服务机器人、智能玩具越来越多，甚至一些前沿机器人已经开始朝着独立意识的智能化发展。人们甚至开始讨论机器人是否能/会替代人类的问题，如最近一首当红歌曲《水星记》的MV就讲述了这样一个关于机器人的故事：在一场意外中失去爱人的母亲深陷思念之痛，女儿为了帮助妈妈走出困境，找到一台拥有爸爸的性格、且能展现爸爸的相貌的机器人来陪伴妈妈；朝夕相处中，妈妈渐渐摆脱痛苦，接纳了这个“丈夫”，身体状况也越来越好，一家三口其乐融融的场景再次出现，生活恢复正常；然而虚拟还是难以完全替代真实，妈妈关于那场意外事故的记忆偶然被唤起，痛苦万分，当然，这是后话。

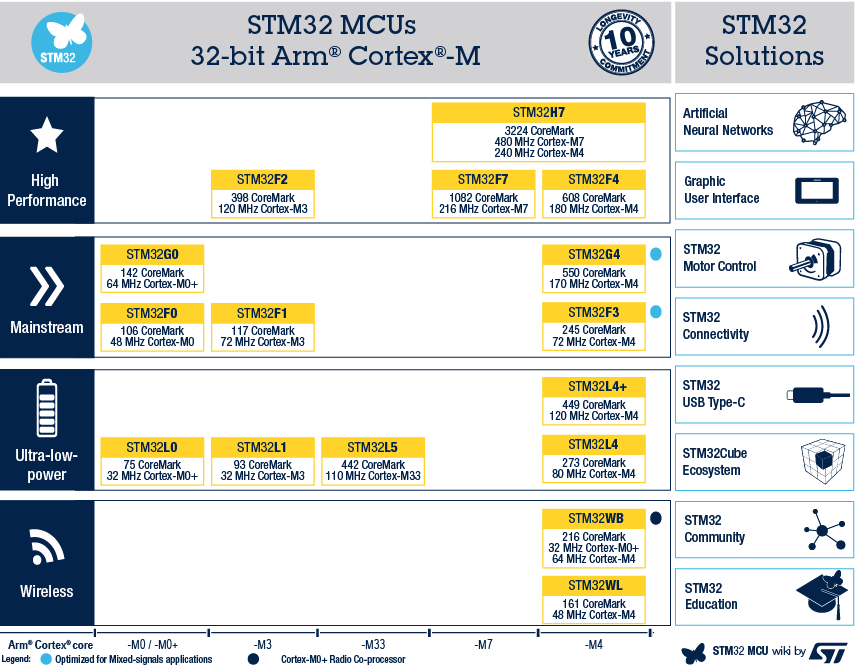
图片来源：水星记MV

此次我们的主角就是一台服务机器人，它拥有智能视觉系统和动作控制，提供端茶送水、取物等功能。智能系统都需要通过学习来赋能，学习的对象就是人类；所以，**我们需要设计一个通信系统，将采集设备传入的数据帧转换成机器人规定的格式并发送给机器人，同时按需与上位机通信。**这就是本次实验的内容。

平台介绍

1. 芯片功能和资源

基于Arm®Cortex®-M处理器的**STM32系列32位微控制器**旨在为MCU用户提供更高的自由度。它提供的产品结合了超高性能、实时性、数字信号处理、低功耗/低压操作和互联功能，同时集成了完整功能、保持了易于开发的特性。

基于行业标准内核的STM32微控制器无与伦比，提供了广泛的工具和软件选择来支持项目开发，这使得该系列产品非常适合小型项目和端到端平台。

（图片来源：<https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>）

本次我们采用**基于STM32F103芯片的硬石YS-F1Pro开发板**为我们的硬件平台，其中STM32F103是STM32系列的一款主流(mainstream)级产品，频率为72 MHz，高达1 MB的闪存，除一般单片机均有的GPIO、定时器、ADC、UART、SPI、I2C等通信接口外，还拥有电机控制、USB和CAN等资源。硬石YS-F1Pro开发板与同类产品相比，板载硬件主打电机控制功能，适于我们的机器人项目。

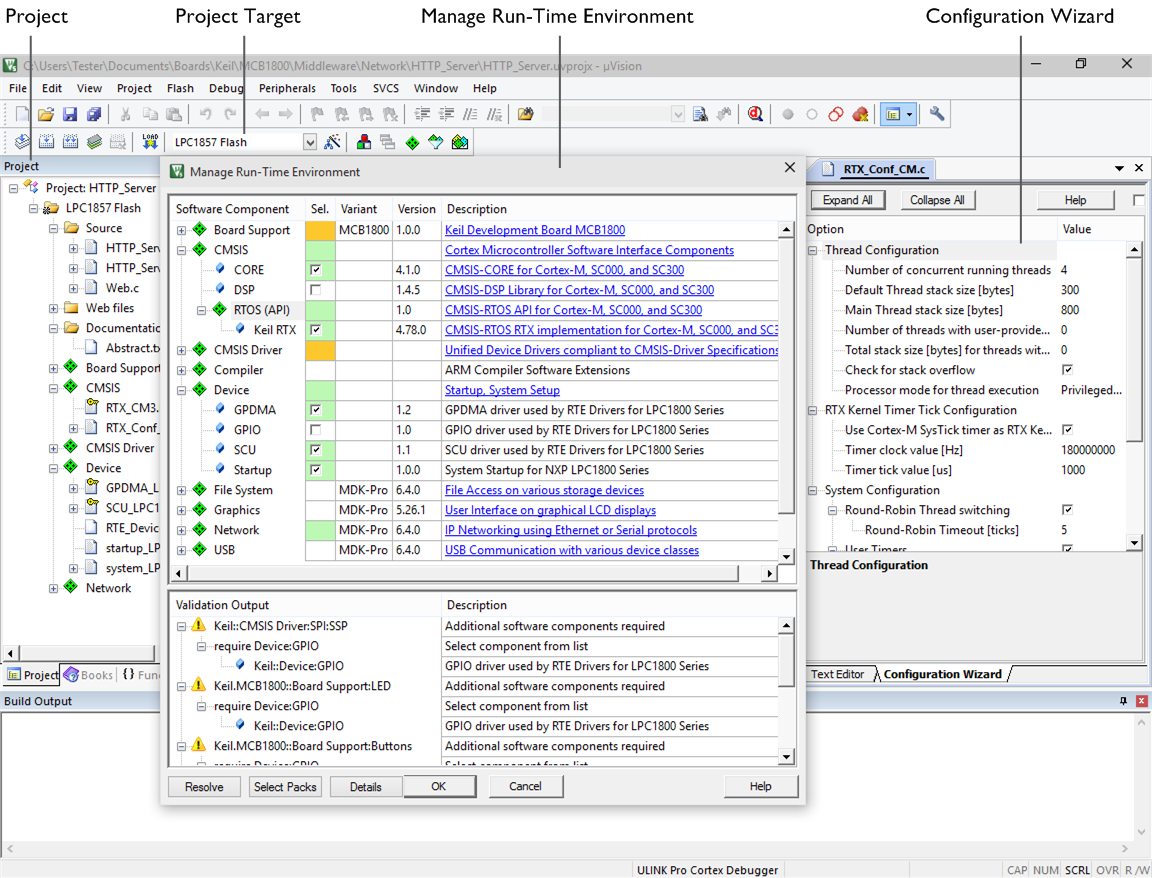
1. 软件框架

Keil MDK单片机开发套件适用于众多基于Arm Cortex-M处理器的微控制器设备的完整软件开发环境。MDK包括µVision IDE和调试器，Arm C / C ++编译器以及基本的中间件组件。它为所有芯片供应商提供了7000多种设备，并且易于学习和使用。

**µVision IDE**是最受欢迎的单片机开发环境之一。它在一个强大的环境中结合了项目管理、运行时环境、构建工具、源代码编辑和程序调试。µVision易于使用，可加快嵌入式软件的开发速度。µVision支持多个屏幕，并允许您在可视表面上的任何位置创建单独的窗口布局。

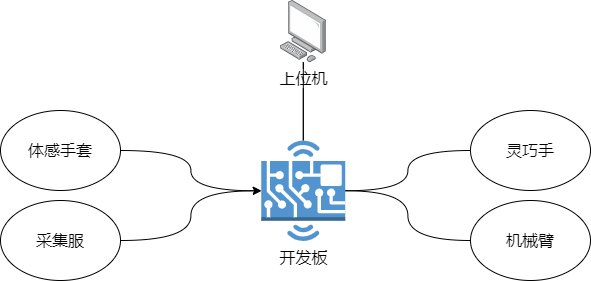
μVision调试器提供了在同一环境中测试、验证和优化应用程序代码的能力。该调试器包括传统功能（例如简单和复杂的断点）、监视窗口和执行控制，并提供对设备外围设备的完全可见性。

本次实验我们选用µVision IDE进行嵌入式开发。



目标及原理介绍

我们的服务机器人的控制主要分为两个设备：灵巧手（机器人的手部，支持抓握等动作）、机械臂（机器人的臂部，可屈曲）。为让机器人模拟人类动作，增设两个采集设备：体感手套、采集服（体感衣），它们传入的数据（手指角度、抓握力度、臂部角度等）可（经转换后）直接用于控制机器人的动作，使灵巧手、机械臂做出与使用者相同的动作。在模拟的基础上，再增设一台计算设备（上位机），其对采集设备采集到的数据进行学习后，就可以为机器人提供决策，实时发送指令控制机器人的动作。

我们需要设计一个通信系统，将采集设备传入的数据帧转换成机器人规定的格式并发送给机器人，并且满足采集设备到上位机、上位机到机器人的通信需求。

第一部分：采集设备-机器人通信

这部分涉及到四个部件的通信，其中体感服、体感手套、灵巧手都是通过串口通信的，机械臂则是采用CAN通信。

实验需要做的主要内容就是将体感服、体感手套收到的位置、速度信号翻译成指令，再将指令发送给机械臂和灵巧手。YS-F1Pro有3个串口，本次我们全部使用了。串口1用于接收体感手套的输入，串口2用于接收体感服的输入，串口3用于与灵巧手之间通信。

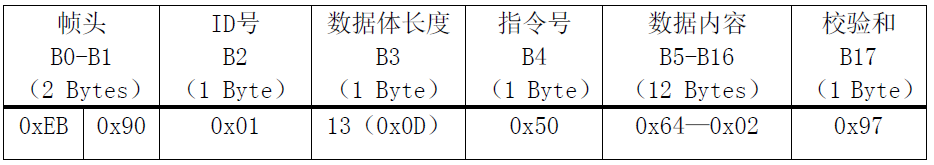
1. HAL\_UART\_Receive\_IT(&husart2, armBuffer, 52); // 启用监听体感服
2. HAL\_UART\_Receive\_IT(&husart1, handBuffer, 18); // 启用监听体感手套
3. HAL\_UART\_Receive\_IT(&husart3, &handResponse[contHandResponse], 1); // 启用监听灵巧手（指令帧的回传）
4. 灵巧手 & 体感手套

首先讨论**指令帧**。灵巧手与体感手套能直接交流的主要指令是**设置目标位置**：主控单元设置灵巧手中 6 个直线驱动器的目标位置，使灵巧手完成相应的手势动作。灵巧手中的 6 个直线伺服驱动器的 ID 号为1-6，其中小拇指的 ID 为1、无名指的 ID 为 2、中指的 ID 为 3、食指的 ID 为 4、大拇指弯曲指关节 ID 为 5、大拇指旋转指关节 ID 为 6。

​ 指令帧长度：18Bytes

​ 指令号：0x50（CMD\_MC\_SET\_DRVALL\_SEEKPOS）

​ 数据内容：6 个驱动器的目标位置，每个位置为2Bytes（小端模式低字节先发送），共12Bytes，目标位置有效值为0~2000。若为0xFFFF，则表示不需要设置该驱动器的目标位置，因此可单独设置某个驱动器的目标位置。

​例如：主控单元设置ID 为1 的灵巧手的目标位置（依次为100、200、300、400、500、600），需要发送的指令帧如下表所示：

​注意到这其中有个**校验和**：是指令帧除帧头外其余字节的累加和的低字节。对于上图的指令帧就是 ((B2+B3+…+B16) & 0xFF)

​因为灵巧手与体感手套有着相同的数据帧格式，所以只需要将体感手套发出的数据帧再传给灵巧手即可控制灵巧手的位置：

1. **if**(HandCheckSum(handBuffer,handBuffer[3]+5) == 0){//如果收到的数据有误则丢弃
2. printf("Data receiving Error from somatosensory glove\n");
3. **return** ;
4. }
6. // Send to robot hand
7. HAL\_UART\_Transmit(&husart3,handBuffer,18,0); //直接将收到的位置信息发给灵巧手
9. // Continue listening on somatosemsory glove
10. HAL\_UART\_Receive\_IT(&husart1, handBuffer, 18);

指令帧是比较长的（18个字符），所以指令帧自带一种设计了一种校验机制：指令帧的最后一个字节是校验和（Check\_Sum）其定义为除了帧头之外其余所有数据累加和的低字节，那么为了确保传输的数据正确，我在收到数据之后也需要先校验指令帧的检验和，如果指令有误则丢弃：

1. /\*\*
2. \* 函数功能: 检查接收数据的校验和
3. \* 输入参数: 数组指针、数组长度
4. \* 返 回 值: 1 for correct
5. \* 说    明：适用于灵巧手系列
6. \*/
7. uint8\_t HandCheckSum(uint8\_t\* a, **int** begin){
8. **int** lenth = a[(begin + 3) % handResponseSize];
9. uint16\_t sum = 0;
10. uint8\_t check = 0;
11. **for**(**int** i = 2; i < lenth - 1; i++){
12. sum += a[(begin + i) % handResponseSize];
13. }
14. check = sum & 0xFF;
15. **if**(check == a[(begin + lenth - 1) % handResponseSize])
16. **return** 1;
17. **else**
18. **return** 0;
19. }

体感手套收到指令帧之后会发出一段**应答帧**作为回应，一般而言，应答帧的主要功能就是说明一个指令被正确执行了，如果一条指令向灵巧手请求内容的话会返回请求的数据。在这里我们并不需要应答帧做什么实际的事情，就是解读一下其含义并告知用户即可。

1. /\*\*
2. \* 函数功能: 解释灵巧手收到的数据
3. \* 输入参数: handResponse
4. \* 返 回 值: 无
5. \* 说    明：在验证校验和之后使用
6. \*/
7. **void** HandResponseDescribe(**int** begin){
8. **switch** (handResponse[(begin + 4) % handResponseSize]){
9. **case** 0x50:{
10. printf("设置目标位置\n");
11. **if**(handResponse[(begin + 5) % handResponseSize] == 0x01){
12. printf("OK\n");
13. }
14. **else** **if**(handResponse[(begin + 5) % handResponseSize] == 0x55){
15. printf("Unable to set hand's position\n");
16. }
17. **break**;
18. }
19. **case** 0x51:{
20. printf("设置速度\n");
21. **if**(handResponse[(begin + 5) % handResponseSize] == 0x01){
22. printf("OK\n");
23. }
24. **else** **if**(handResponse[(begin + 5) % handResponseSize] == 0x55){
25. printf("Unable to set hand's velocity\n");
26. }
27. **break**;
28. }
29. ...  // 类似情况
30. **default**:{
31. printf("Unknown Response\n");
32. **break**;
33. }
34. }
35. }
36. /\*\*
37. \* 函数功能: 串口接收完成回调函数
38. \* 输入参数: 无
39. \* 返 回 值: 无
40. \* 说    明：在此仅给出了与灵巧手有关的代码
41. \*/
42. **void** HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*UartHandle) {
43. **if**(UartHandle == &husart3) {
44. **if**(handResponse[contHandResponse] == 0x16 && handResponse[(contHandResponse + handResponseSize - 1) % handResponseSize] == 0xEE){
45. **int** i = 7;
46. **int** tem1 = (contHandResponse - i + handResponseSize)  
    % handResponseSize;
47. **int** tem2 = (contHandResponse - i + handResponseSize - 1)  
    % handResponseSize;
48. **if**(handResponse[tem1] == 0x16 && handResponse[tem2] == 0xEE){
49. **if**(HandCheckSum(handResponse, tem2) == 0){
50. printf("Date receiving Error from smart hand\n");
51. }
52. HandResponseDescribe(tem2);
53. }
55. i = 16;
56. tem1 = (contHandResponse - i + handResponseSize) % handResponseSize;
57. tem2 = (contHandResponse -i + handResponseSize - 1)  
     % handResponseSize;
58. **if**(handResponse[tem1] == 0x16 && handResponse[tem2] == 0xEE){
59. **if**(HandCheckSum(handResponse, tem2) == 0){
60. printf("Date receiving Error from smart hand\n");
61. }
62. HandResponseDescribe(tem2);
63. }
65. i = 18;
66. tem1 = (contHandResponse - i + handResponseSize) % handResponseSize;
67. tem2 = (contHandResponse -i + handResponseSize - 1)  
     % handResponseSize;
68. **if**(handResponse[tem1] == 0x16 && handResponse[tem2] == 0xEE){
69. **if**(HandCheckSum(handResponse, tem2) == 0){
70. printf("Date receiving Error from smart hand\n");
71. }
72. HandResponseDescribe(tem2);
73. }
74. }
76. contHandResponse++;
77. contHandResponse %= handResponseSize;
78. HAL\_UART\_Receive\_IT(&husart3, &handResponse[contHandResponse], 1);
79. }
80. }

这里有一点需要单独说明一下：应答帧的总长度是不确定的，这就导致不知道什么时候应答帧已经被完整的收到了。但是应答帧有共同的头部，所以我采用的策略是不断存贮和检验收到的应答帧，如果检测到收到了一个应答帧的头部，就去处理该头部的前一个应答帧，所以在效果上应答帧的处理会慢一拍，但是由于指令的发送的频率比较高，所以不会出现太大问题。而体感手套发送的指令帧结构单一，所以没有这样的问题。

1. 采集服 & 机械臂

体感服与机械臂的关系和前述灵巧手部分非常类似，所以在此就不过多的描述了，但是其中有几点需要注意。

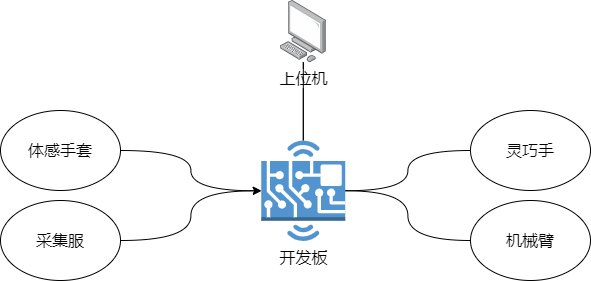
体感采集服与机械臂的数据有一些细微的差异，差异主要是由于体感采集服的输出信号中每一组数据都是低八位在前，高八位在后，而机械臂需要的指令是高八位在前，第八位在后。需要手动处理一下，也就是将收到的体感服数据的高八位与低八位调换一下位置。

另一个需要注意的点是体感采集服每一帧包含52B数据，是体感服当前状态的一个完整描述（左右肩、左右臂、左右手腕、头部）。而机械臂指令帧每帧包含8个字节，在这8B中只描述一个信息（一种角度），所以需要将收到的体感服数据拆分成块，再分块传送给机械臂。

1. /\*\*
2. \* 函数功能: 串口接收完成回调函数
3. \* 输入参数: 无
4. \* 返 回 值: 无
5. \* 说    明：仅选取体感服部分
6. \*/
7. **void** HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*UartHandle)
8. {
9. **if**(UartHandle == &husart2)  // Somatosensory clothes
10. {
11. // Process data
12. **const** uint8\_t fingers[3][5] = {
13. {0, 0, 0, 0, 0},
14. {14, 16, 18, 20, 22},  // 7 (right)
15. {36, 38, 40, 42, 44}   // 14 (left)
16. };
17. **const** uint8\_t byte\_2\_3[15][2] = {
18. {0, 0}, {3, 2}, {5, 4}, {7, 6}, {9, 8}, {11, 10}, {13, 12},
19. {0, 0}, {25, 24}, {27, 26}, {29, 28}, {31, 30}, {33, 32}, {35, 34},
20. {0, 0}
21. };
22. // Convert from somatosensory clothes data to robot arm data
23. // right arm
24. **for**(**int** i = 2; i < 13; i += 2){
25. uint32\_t tem1 = armBuffer[i+1]; // 高8位
26. uint32\_t tem2 = armBuffer[i] + tem1 \* 256; // 全体数据
27. armBuffer[i+1] = tem2 & 0xFF; // 低8位
28. armBuffer[i] = (tem2 / 0xFF) & 0xFF; // 高8位
29. }
30. // left arm
31. ...
32. // head angle
33. ...
35. // Send to robot arm
36. **for** (uint8\_t i = 1; i <= 14; i++)
37. {
38. CAN\_SetTxMsg(i); HAL\_CAN\_Transmit(&hCAN, 8);
39. HAL\_CAN\_Receive\_IT(&hCAN, CAN\_FIFO0);  // Response frame of   
    robot arm; will go to HAL\_CAN\_RxCpltCallback()
40. }
42. // Continue listening on somatosensory clothes
43. HAL\_UART\_Receive\_IT(&husart2, armBuffer, 52);
44. }
45. }

第二部分：采集设备-上位机-机器人通信

为了智能化机器人的行为，仅仅用采集设备直接控制机器人是不够的，需要连接一个功能更强的计算设备（**上位机**），对采集设备传入的数据进行学习后，再智能地控制机器人。而采集设备和机器人与上位机的通信就是需要我们设计的部分。受疫情影响，此部分实验只进行了理论上通信协议的设计。

需求中一共有4条数据通路：

* 采集服/体感手套→开发板→机械臂/灵巧手（模拟）
* 采集服/体感手套→开发板→上位机（分析）
* 上位机→开发板→机械臂/灵巧手（指令）
* 机械臂/灵巧手→开发板→上位机（应答）

首先确定大致结构。把两种数据帧拼合到一起是不可行的：根据文档，两个采集设备传入的频率不相等，不同步。因此我们把上位机对灵巧手的和对机械臂的控制逻辑分开，上位机分别发对两者的指令，开发板根据数据格式判定后分发到灵巧手或者机械臂，应答帧则是收到即发给上位机（无论从哪一设备发来）。

接下来设计细节。上位机数据帧格式未定，最简单的方法自然是用体感衣的格式作为上位机传入通信格式，但到机械臂需经开发板转换。考虑到无线通信的特点是延迟较大、不稳定（帧可能失序到达），所以倾向于采用较大的帧作为通信格式，类似采集服的格式。这种格式相当于一次赋值给全部参数，因为不是所有参数都会改动（可能与之前相等），所以再增加一个标记位来指示本数据帧改动了哪些参数。

考虑以上各种因素及结合四个设备的通信格式（见文档），设计出上位机通信协议如下：