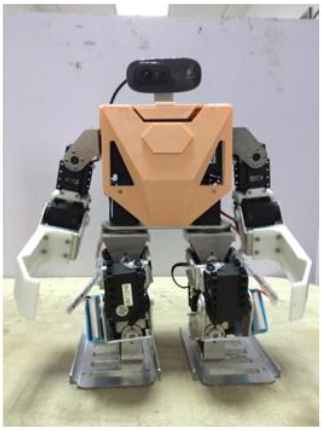
开源机器人项目

HANDS FREE

HANDS FREE 是一个面向机器人研究、开发的开源软硬件系统。她有完备与科学的框架，以优秀的嵌入式系统框架为核心，精良的电路、机械设计为支撑，帮您快速实现多种形态的机器人。本系统包含机器人导航，SLAM，计算机视觉等模块，并拥有自己上层软件和调试系统。她支持国外其他的开源项目，如 ROS, MPRT, PIXHAWK 等，这一切都为您带来了无比的便捷和快乐！

如果你觉得“哎呦不错”的话，就一起加入进来吧！！！

了解我们：



巴克

无人机系统

最新资料和代码请到： https://github.com/HANDS-FREE

HANDS FREE 网页介绍：

http://wiki.exbot.net/HandsFree

http://www.adv-ci.com/

HANDS FREE 交流群：521037187 （Hands Free Community）

小车视频展示：

http://v.youku.com/v_show/id_XMTUyODk4NTUzNg==.htm

核心技术展示：

http://v.youku.com/v_show/id_XMTU0NzgwNzc3Mg==.html?from=y1.7-1.2

购买链接：

https://item.taobao.com/item.htm?spm=a1z10.1-c.w4004-13256568658.2.NQETvg&id=526188987280

ROS 学习社区推荐：

EXBOT : http://blog.exbot.net/

EXBOT 交流群：109434898 (群 1 已满) 426334501（群 2）

ROSCLUB : http://www.rosclub.cn/ ROSCLUB 交流群 ：184903125

EXBOT 已经有很长的历史了，里面有很多 ROS 的使用攻略，以及一些专题的深入讨论 。ROSCLUB 刚起步不久，里面有很多机器人系统方面的文章和 ROS 的使用攻略，HANDS FREE 的很多使用攻略和问题也会发布在 ROSCLUB 上。

RoboLink Manual

只要存在通信就肯定会有通信协议，一个机器人系统肯定是存在多种通信过程，不过这里指的是上位机和底层嵌入式的通信。

RoboLink 也称 HFLink 是该课题下的 HANDS FREE 专门为机器人定制的一个多机通信协议，和小型无人载具 MavLink (Micro Air Vehicle Link)一个概念，但是没那么复杂，方便开发者快速建立的通信结构。这主要是因为国际上还没有比较通用的开源协议来支持大多数机器人的通信，这也说明了机器人还是一个提升空间比较大的方向。

通过上下位机对机器人各项数据的抽象这部分代码的公用，实现封装数据打包与解包过程。现在已经实现了下位机的 USB 串口通讯。数据更新一般以每种指令 10HZ 的速度进行，满足上层规划与感知需求。当然由于我们主控制器性能强劲，通讯数据种类与更新频率都有提升空间。经测试通信包总和频率提升到 1000HZ 也不只是消耗 20%的 CPU。

RoboLink 抽象了多种机器人的运动属性：轮式机器人，平衡车，人形机器人，四旋翼，固定翼，并且可以扩展，支持多机通信。

通信协议：

0XFF 0XFF sender_id receiver_id length_H length_L ****(data) check_sum

对应的状态机如图

```
//communication status
enum Recstate{
    WAITING_FF1,
    WAITING_FF2,
    SENDER_ID,
    RECEIVER_ID,
    RECEIVE_LEN_H,
    RECEIVE_LEN_L,
    RECEIVE_PACKAGE,
    RECEIVE_CHECK
};
```

使用的是和校验

unsigned int check_sum;

check_sum= (0XFF + 0XFF+.....)%255

只有满足以上形式的数据流，才会被 RoboLink 所接受。

多机支持：

所有通信节点都有一个自己的 ID(等价于 MAC 地址或者姓名)，这也是为多机提供基础。多机很直观，就是多个机器人，多个控制终端之间的互相通信，通过 ID 号来一一对应。比如你同时用手机 APP，遥控器，电脑等设备给车发送控制指令，可以通过 ID 号来选择最高优先级的控制端数据，从而保证不混乱。

ID 其实就是一个单字节的数据，不过 RoboLink 有规定，主机和从机的 ID 范围是不一样的，这也是为了统一编排 ID，提升效率。

主机（终端）概念：给实体机械人发送指令的的终端如遥控器，PC ，手机 PP 应用等 ，ID 范围为 0X01 - 0X10 共 15 个 ID。

从机概念：一个实体机器人（被控体）即为一个从机，比如移动机器人，平衡车，四轴 ，ID 范围为 0X11 - 0X40 共 48 个 ID。

广播 ID 0XFF。

网络中主机和从机的数量加起来最大 64 个（用户也可以自己增加），可以理解为一个网络中最多有 64 个人，其中最多有 15 个发号施令的人（监工），最多 48 个执行命令的人（工人），他们通过自己的名字（ID）匹配来交互，接收者可以通过判断 ID 的范围就知道该发送者是主机还是从机，是不是应该解析。

信息交互既可以存在于监工和工人之间，也存在于工人和工人，监工和监工之间。并且 15 个监工的权利（优先级）不是一样大，当多个监工给一个工人同时发送命令的时候，工人先执行优先级高的，比如当自主飞行发生故障时，你可以通过遥控器操控，这是因为遥控器命令的优先级高于自主命令，当然这个优先级你可以自己定义。至于工人和工人之间的交互，比如在机器人多机协同时，几个机器人之间可以通过这种方式来交互信息，0XFF 代表广播信号，即 0XFF 这个名字代表的不是某个工人，而是整个 48 个工人，当监工给这个名字发信息（命令）时，所有工人都执行。

```
#define HF_LINK_NODE_MODEL  0    //1master  0slave
```

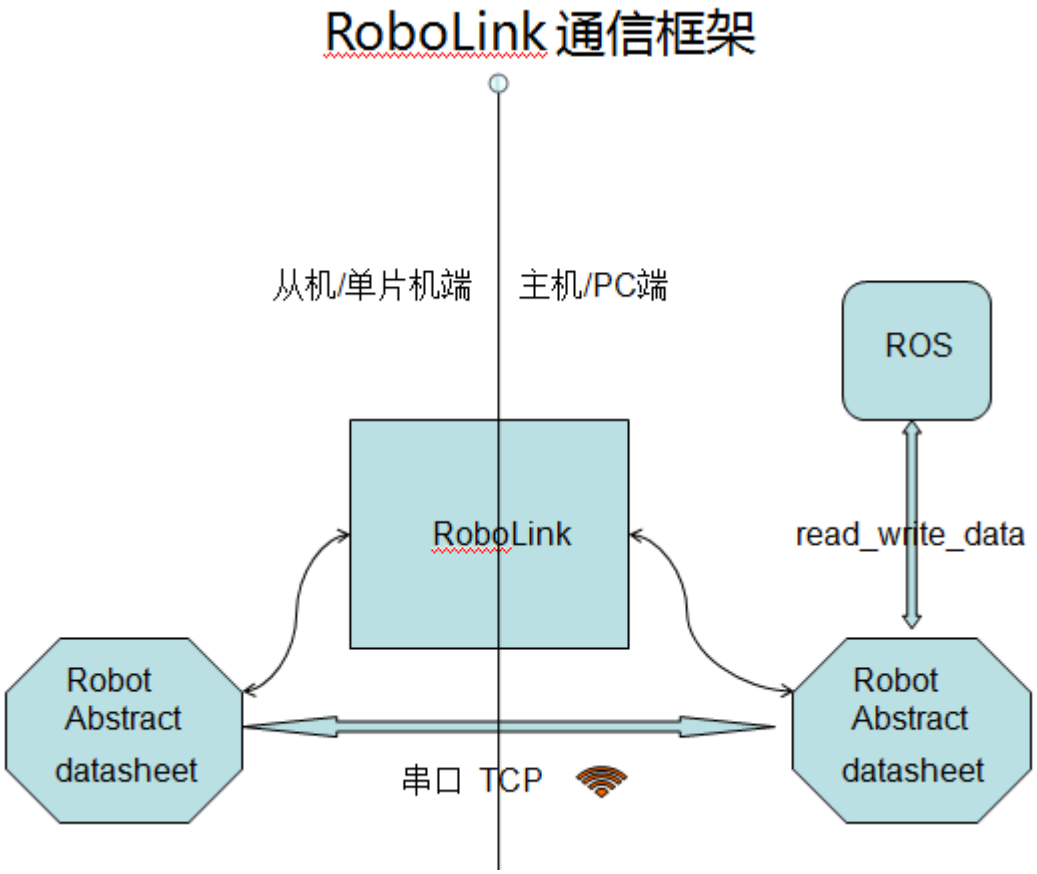
```
class HFLink
{
public:
    HFLink(unsigned char my_id_= 0x11 , unsigned char friend_id_= 0x01 , RobotAbstract* my_robot_=0)
    {
        hf_link_node_model = HF_LINK_NODE_MODEL ;
```

由于主机和从机的代码解析是不一样的，这里需要用到预编译，同时从构造函数可以看出，每构造一个通信节点，都要指明自己的 ID，和通信对方的 ID，当然这还有待改进，理想状况是不需要指定对方的 ID，而是在程序里进行判断解析。

通信框架：

通信即数据交换，RoboLink 专门定义把机器人的数据抽象为一个类，也就是 LIB_Robot_Abstract 这个包，以结构体的形式封装了数据，并且每个结构体是 RoboLink 的最小子单元，也就是说，RoboLink 发送的每个包都是一个结构体。主机从机都定义了同样的抽象类，同时主机专门开一个线程来和从机进行数据交换，最终的效果就等同于把从机的抽象类复制到主机上，当然这都是以结构体为最小单位一个个更新的，RoboLink 以 10HZ 的频率更新这个抽象类。

当实现上面的同步之后，ROS 端就可以直接通过操作主机的抽象类来实现对底层的控制，这是一个异步的过程。 另外 RoboLink 和 ROS 端可能同时对主机抽象类进行读写操作，这就需要在代码上加锁防止重入，同时主机代码还要实现通信的 ACK 检测，丢包处理，多种通信接口的驱动等，不过这些 HANDSFREE 都已经封装好了，对于使用者来说只要知道怎么使用这些数据应该就可以了。另外就是协议本身是一个不完善的东西，对于高级应用的话还是有很多要改进的地方。



头文件代码解析：

```
#ifndef HF_LINK_H
#define HF_LINK_H

#include "robot_abstract.h"

#define HF_LINK_NODE_MODEL  0    //1master  0slave

#if HF_LINK_NODE_MODEL==0
#include "hf_link_port.h"
#endif

static const unsigned short int MESSAGE_BUFER_SIZE = 120;  //limite one mess
/****structure for communications protocol , read Hands Free Link Manua.doc
typedef struct HFMessage{
    unsigned char sender_id;
    unsigned char receiver_id;
    unsigned short int length;
    unsigned char data[MESSAGE_BUFER_SIZE];
}HFMessage;
```

```
//communication status
enum Recstate{
    WAITING_FF1,
    WAITING_FF2,
    SENDER_ID,
    RECEIVER_ID,
    RECEIVE_LEN_H,
    RECEIVE_LEN_L,
    RECEIVE_PACKAGE,
    RECEIVE_CHECK
};
```

```
141     HFMessage rx_message_ , tx_message_;
```

以上宏定义用于选择主机还是从机，同时定义了一个满足协议形式的结构体和一个用于状态机的枚举。定义了两个 message 结构体变量来保存当前接收到的包和即将要发送的包。

定义了一个枚举体表示各种命令，如下图，主机可以给从机发送多种命令，比如控制底盘的速度，控制云台的姿态，又或者是请求从机的坐标，电池信息等，这些命令都被封装在这个枚举体里，从机通过识别这个变量来做出相应的判断。比较要注意的就是“握手指令”，即 SHAKING_HANDS，当首次通信的时候，从机会向主机发送握手指令，主机就会把初始数据发给从机，比如主机记忆的坐标和姿态，通过握手主机从机的初始数据就同步了。

```
//comand type
enum Command{
    SHAKING_HANDS,
    READ_ROBOT_SYSTEM_INFO,
    SET_GLOBAL_SPEED,
    READ_GLOBAL_SPEED,
    SET_ROBOT_SPEED,
    READ_ROBOT_SPEED,
    SET_MOTOR_SPEED,
    READ_MOTOR_SPEED,
    READ_MOTOR_MILEAGE,
    READ_GLOBAL_COORDINATE,
    READ_ROBOT_COORDINATE,
    CLEAR_COORDINATE_DATA,
    SET_ARM_1,
    READ_ARM_1,
    SET_ARM_2,
    READ_ARM_2,
    SET_HEAD_1,
    READ_HEAD_1,
    SET_HEAD_2,
    READ_HEAD_2,
    READ_IMU_DATA,
    SET_ROBOY_PARAMETERS,
    SAVE_ROBOY_PARAMETERS,
    SET_MOTOR_PARAMETERS,
    SAVE_MOTOR_PARAMETERS,
    LAST_COMMAND_FLAG};
```

Set_Global_Speed//给机器人发送 世界坐标系的速度向量 X Y Z (float)
Set_Robot_Speed//给机器人发送 机器人坐标系的速度向量 X Y Z (float)
Read_Global_Coordinate//读取机器人世界坐标系坐标值,返回一个 Package:
Set_Motor_Speed //给机器人发送 电机速度 V1 V2 V3 (float)
Read_Motor_Speed//读取三个电机的角速度,返回一个 Package:
Read_Motor_Mileage//读取三个电机的里程值
Clear_Coordinate_Data //清除底盘历史数据 即设置当前坐标 为坐标原点
Set_Arm_1//给机械臂 1 发送关节夹角数据 R1 R2 R3 R.....
Read_Arm_1/读取机械臂 1 关节夹角数据 R1 R2 R3 R.....
Set_Arm_2//给机械臂 2 发送关节夹角数据 R1 R2 R3 R...
Read_Arm_2//读取机械臂 2 关节夹角数据 R1 R2 R3 R...
Set_Head_1//给头部发送 PITCH ROOL YAW (float)
Read_Head_1//读取 PITCH ROOL YAW (float)

主机从机只要通信双方保证枚举变量一致，即可正常通信，而且这样增加了其可扩展性，使用者可以自定义其指令。很明显，以上指令是为不同系统准备的，正常情况下不可能都用上，即存在冲突，比如 Set_Robot_Speed，Set_Motor_Speed

是不可能同时执行的，但是如果有些二逼要同时给机器人发送这两条指令，Hands Free 会执行更底层的指令，也就是 Set_Motor_Speed ，因为最终 Set_Robot_Speed 会转化成 motor speed 执行的，所以说 Set_Motor_Speed 比 Set_Robot_Speed 更底层。

每条指令都有自己的有效周期，比如给机器人发送一条 Set_Robot_Speed ，他肯定不会一辈子都按照这个指令执行的，你想想，要是主机给机器人发送了一条速度指令后，突然信号断了，为了不让机器人冲出地球，所以每条指令只执行有限的时间，也就是说，你得以一定频率给机器人发送指令，他才会不停的动。

协议举例：

16 进制表示： 黄色为指令枚举代号
发送 ff ff 01 11 00 0d 01 (一号****) (二号****) (三号****) check
给机器人发送 世界坐标系的速度向量 x y z
返回 ff ff 11 01 00 02 01 check

发送 ff ff 01 11 00 0d 02 (一号****) (二号****) (三号****) check
给机器人发送 机器人坐标系的速度向量 x y z
返回 ff ff 11 01 00 02 02 check

发送 ff ff 01 11 00 01 03 check 读取机器人世界坐标系坐标值
返回 ff ff 11 01 00 02 03 (X****) (Y***) (Z****) check 返回世界坐标系坐标值

HANDSFREE 使用机体坐标系即 Set_Robot_Speed 控制指令来控制机器人，而没使用控制电机速度指令即 Set_Motor_Speed 来控制，这样相当于把底层封装成一个整体，但是为了上位机在特殊情况下直接驱动电机，协议保留了直接控制电机速度的指令 Set_Motor_Speed。

主要函数分析：

主机主要是通过 masterSendCommand()函数来通信的，形参 command_state 就是主机想执行的命令，比如给主机的机器人抽象类的 expect_robot_speed 结构体赋值为(1，0，0)。再运行 masterSendCommand(Set_Robot_Speed)函数，就可以发现机器人以 1m/s 的速度在前进。

```
//only for master
//the master can use masterSendCommand function to send data to
//like SET_GLOBAL_SPEED , READ_ROBOT_SYSTEM_INFO, READ_ROBOT_SPE
unsigned char masterSendCommand(const Command command_state);
```

```
public:
    //common
    unsigned char byteAnalysisCall(const unsigned char rx_byte);

    //only for slave
    //command updata flag , the robot need to traverse These flag to decide update his own behavior
    unsigned char receive_package_renew[LAST_COMMAND_FLAG];
```

同时主机从机所接受的数据流都由 unsigned char byteAnalysisCall(const unsigned char rx_byte) 函数来解析，只需要不断从缓冲区取出数据传递到该函数即可实现对机器人抽象类的读写操作。同时每收到一个包 receive_package_renew 数组的对应位置 1，通过查询数组知道收到了什么指令以及同时处理对应的事件，实现对机器人物理状态的更新。