开源机器人项目

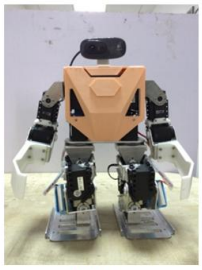
HANDS FREE

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：


斯通/Stone


巴克


基隆/JiLong

无人机系统

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

最全资料请去百度云： http://pan.baidu.com/s/1c201NC

HANDS FREE 网页介绍：

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

http://www.rosclub.cn/post-265.html

http://wiki.exbot.net/HandsFree

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://shop145029875.taobao.com/?spm=a1z10.3-c.0.0.zpwB3d

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 和 ROSCLUB 是友好的合作关系，所以很多使用攻略和问题也会发布在 ROSCLUB 上。

OPENRE 使用手册

第一篇：简述

OpenRE 是一个专门为机器人写的、基于 STM32 系列微处理器的嵌入式开源库。它的前身是 HANDSFREE 的 Embedded 库，Embedded 库主要是为一代平台“基隆”服务的，经过后续的优化，库变得更加的鲁棒和通用，从而独立于平台成为了一个专门为机器人而造的一个嵌入式库，于是重命名为 Open Source Robot Embedded Library (OPENRE)。

获取 OpenRE 源码： https://github.com/HANDS-FREE/OpenRE
获取 Embedded 源码： https://github.com/HANDS-FREE/Embedded

OpenRE 库相对于之前的 Embedded 库一个最大的变化就是把原来的 windows keil 开发环境迁移到 linux 下，使用 makefile + QTCreator + armgcc 来进行开发，相信熟悉 makefile 的都知道他有多方便，至于 windows 系统的开发者，也是可以配置 windows 下的 makefile + QTCreator + armgcc 的环境来开发的，这里也强烈建议开发者学习使用 makefile，在构建大规模程序框架的时候显得特别给力。

对于那些是知道用集成开发环境的开发者，或许认为 keil 更加的方便，但是其实是“会者不难，难者不会”，若是实在不想学习强大的 make，你也可以按照后面的傻瓜式操作来进行开发。不过 HANDSFREE 的宗旨第一要义是学习和科研，追求永无止境，不断创新，所以恕我们很难照顾那些不想进步的人，从 OpenRE 开始就没有 keil 的工程文件啦。

获取 makefile 的学习资料：http://pan.baidu.com/s/1c201NC

OpenRE 库还移植了 PX4 的 bootloader，增加了硬件抽象层以适用于不同的电路板，增加了 Eigen 和 Matrix 矩阵运算库，对框架和一些包都进行了优化，等等。

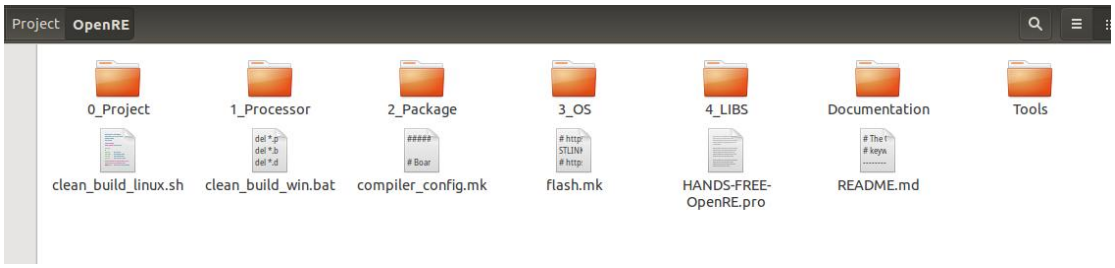
在你正式进行开发之前，你需要准备一些基础知识，会使用 linux 系统(笔者现在使用的是 ubuntu 14.04 + ROS indogo)，会用 git 管理自己的代码（同时注册一个 github 账号），会使用并且安装好了 qtcretor。

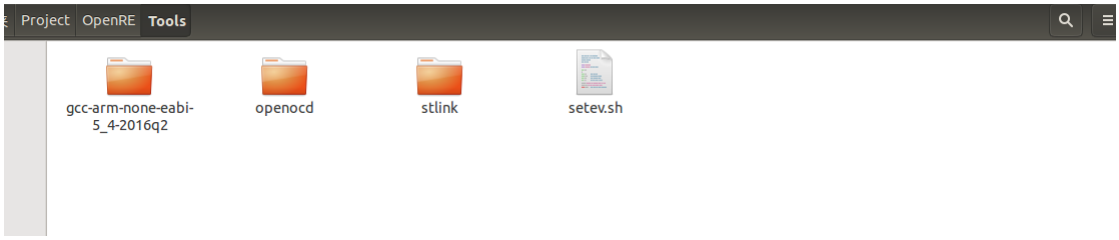
第二篇：建立开发环境

获取 OpenRE 源码后，在 OpenRE 目录下建立 Tools/目录，把笔者下载好的工具链和一件配置脚本放进去。

工具链下载地址：
https://pan.baidu.com/s/1jl6TeJG#path=%252FHANDSFREE%252FOpenRE_Tools
一件配置脚本下载地址：
https://github.com/HANDS-FREE/OpenRE_Tools

放好之后，在 Tools/ 目录下运行 setev.sh 就可以配置好了，配置好后就可以把压缩包删除了。这样目录结构就和下面一样：





配置过程主要就是把压缩包解压，同时进行编译，这个过程可能不会完全顺利，所以确保你的编译没有出现问题，也就是说 openocd 和 stlink 都有生成可执行文件。

Openocd 包是烧写器的驱动，支持 jlink，stlink，swd 等 。 stlink 包只能支持 stlink。默认情况下 OpenRE 是调用 Openocd 进行烧录的，所以确保这个包编译过程是正常的。

即使配置好了，烧写可能能不正常，这时可能需要改 USB 权限，或者用 sudo 命令进行烧写。

第三篇：编译烧写

OpenRE 移植了 PX4 的 bootloader，所以开发者可以使用 bootloader 来烧写，也可以直接用配套的 jlink 进行烧写。

关于什么是 bootloader 可以百度 stm32 bootloader 获取介绍，总之有了 boot 后你就可以直接用 USBTTL 接口进行烧写了。

用 jlink 烧写时，由于不同的电路板外部晶振频率可能不一样，所以把 A 板的固件烧到 B 板就可能导致死机和锁死，然后下一次烧写的时候就烧不进了，当出现这种情况的时候，方法就是按下板子的复位键(按住不动)，运行烧写指令，一两秒后，松开复位键，就能烧写进去了。

为了确保你不会烧写错误，你先确保每个工程的 makefile 文件和你的板子是匹配的。这里先介绍每个工程所独有的 makefile，下一篇会详细介绍整个库的 makefile。

Makefile

```
1 ######################################define project name, board type, and path
2
3 #####ROBOT_MODEL : UGV_JILONG_2WD UGV_JILONG_3WD UGV_STONE_2WD UGV_STONE_2WD_PLUS
4 ROBOT_MODEL      ?= UGV_STONE_2WD
5 #####BOARD_TYPE: CONTROL_UNIT_V1 CONTROL_UNIT_V2
6 BOARD_TYPE       ?= CONTROL_UNIT_V2
7 #####BOOTLOADER  : ENABLE DISABLE
8 BOOTLOADER_MODE ?= DISABLE
9 PROJECT          =  Template_NO_RTOS_$(ROBOT_MODEL)_$(BOARD_TYPE)
10 #####FPU  : ENABLE DISABLE
11 FPU_STATE        ?= ENABLE
12 TOP_PATH         =  ../../../..
13
14 ################################################################################source
15
16 CXX_SRC          += ../src/main.cpp ../src/stm32f4xx_it.cpp
17
18 #Includes
19 INCDIR  += -I. -I../src/
20
21 ##########################################################################package
22 #PAKG: common robot_abstract math imu motor sbus_ppm servo robot_wheel hf_link
23 PAKG    =
24
25 #OS_MODULE: UCOSII UCOSIII GUI FAT
26 OS_MODULE =
27
28 #LIB_MODULE: EIGEN MATRIX  etc
29 LIB_MODULE =
30
31 ################################################################################include rules
32
33 include $(TOP_PATH)/compiler_config.mk
34
```

黑色字体是变量和变量值，注释部分是变量的可选值。接下来解释一下每个变量的含义。

ROBOT_MODEL： 是你机器人的型号，可选值是 HANDSFREE 发布的几款机器人，当然整个变量只有当你烧写机器人程序的时候才会生效，烧写简单的测试程序时不起作用。

BOARD_TYPE： 是你使用的主控制器的型号，目前 HANDSFREE 发布了 CONTROL_UNIT_V1 和 CONTROL_UNIT_V2 两款主控，由于外部晶振频率不一样，所以不能选错，你可以看到你板子后面的丝印确认你板子的版本。

BOOTLOADER_MODE： 是否使用 bootloader 模式烧写，使用和不使用在工程里编译了不同的代码，同时烧写指令也是调用不同的驱动程序的。当然如果要使能该项，请确保你的电路板已经烧好了 bootloader 的了，否则不能用 bootloader 进行烧录。另外对于已经烧好 bootloader 的板子，不能使用 jlink 进行烧写，否则将会把之前的 boot 冲掉。

PROJECT： 工程名字，随便起
FPU_STATE： 是否使能 FPU
TOP_PATH： 顶层目录的路径
PAKG： 依赖的包
OS_MODULE： 依赖的 OS 模块
LIB_MODULE： 依赖库

在正式烧写前，请确保你之前的 makefile 都配置对了，主要是 ROBOT_MODEL，BOARD_TYPE，BOOTLOADER_MODE 这三个变量。

1. 直接使用 jlink 的 SWD 模式进行烧写：

先编译一个简单的工程验证一下你的开发环境已经配置成功：
首先把 makefile 的 BOARD_TYPE 改成你板子的型号,如果你使用的第三方板子，则需要看懂 makefile，看后面的配置是不是符号你的板子。
插上 jlink 同时用 usb 上电。

UCOSIII STM32F4 IOToggle:

```
cd OpenRE/0_Project/STM32F4DEMO/Template_UCOSIII/linux
make
make burn
```

烧写成功的现象是，板子上的 led 一直再闪，蜂鸣器一直在叫。

接下来烧写一个基于 CONTROL_UNIT_V2 的斯通移动机器人的工程：
这个工程的 makefile 配置是：

```
ROBOT_MODEL       ?= UGV_STONE_2WD
BOARD_TYPE        ?= CONTROL_UNIT_V2
BOOTLOADER_MODE   ?= DISABLE
```

Wheel_Robot_Beta:

```
cd OpenRE/0_Project/Application/Wheel_Robot_Beta/linux
make clean
make
make burn
```

因为所有的工程都是共享源码的，所以切换工程进行编译的时候，记得先 make clean。

2. 使用 bootloader 模式进行烧写：

先不做介绍，可参考 PIX 官网
http://dev.px4.io/software_update.html

第四篇：makefile 详解

Makefile

```
1 ######################################define project name, board type, and path
2
3 #####ROBOT_MODEL : UGV_JILONG_2WD UGV_JILONG_3WD UGV_STONE_2WD UGV_STONE_2WD_PLUS
4 ROBOT_MODEL     ?= UGV_STONE_2WD
5 #####BOARD_TYPE: CONTROL_UNIT_V1 CONTROL_UNIT_V2
6 BOARD_TYPE      ?= CONTROL_UNIT_V2
7 #####BOOTLOADER  : ENABLE DISABLE
8 BOOTLOADER_MODE ?= DISABLE
9 PROJECT         =  Template_NO_RTOS_$(ROBOT_MODEL)_$(BOARD_TYPE)
10 #####FPU  : ENABLE DISABLE
11 FPU_STATE       ?= ENABLE
12 TOP_PATH        =  ../../../..
13
14 ################################################################################source
15
16 CXX_SRC         += ../src/main.cpp ../src/stm32f4xx_it.cpp
17
18 #Includes
19 INCDIR  += -I. -I../src/
20
21 ################################################################################package
22 #PAKG: common robot_abstract math imu motor sbus_ppm servo robot_wheel hf_link
23 PAKG    =
24
25 #OS_MODULE: UCOSII UCOSIII GUI FAT
26 OS_MODULE =
27
28 #LIB_MODULE: EIGEN MATRIX  etc
29 LIB_MODULE =
30
31 ################################################################################include rules
32
33 include $(TOP_PATH)/compiler_config.mk
34
```

每个工程都有一个自己的 makefile 文件，并且都 include 了顶层目录下的 compiler_config.mk，而 compiler_config.mk 文件都包含了其它的.mk 文件。这里主要介绍以下几个重要的.mk 文件，看懂这几个文件，自己 DIY 就完全不是问题。

compiler_config.mk:

这个文件除了去包含其它几个.mk 之外，还定义的编译的规则，一句话说，就是定义合适的规则，按照合适的编译流程，编译出.elf 文件并且转化成.hex 和 .bin 的固件，懂 makefile 的应该是秒懂啦。

主要就是介绍以下图片 的内容，

CXX_SRC += $(foreach n,$(PAKG),$(wildcard$(PACKAGE_PATH)/$(n)/*.cpp))
就是对 makefile 的 PAKG 变量进行解析，只要依赖的包，就包含其源文件和路径。
ifeq "$(strip $(ROBOT_MODEL))" "UGV_JILONG_3WD"
DDEFS   += -DHF_ROBOT_ID=1
endif
就是解析他的机器人模型，同时定义全局宏代表机器人的型号，以便工程内部预编译不同的代码。

```
#package
PACKAGE_PATH    = $(TOP_PATH)/2_Package
CXX_SRC         += $(foreach n,$(PAKG),$(wildcard $(PACKAGE_PATH)/$(n)/*.cpp))
C_SRC           += $(foreach n,$(PAKG),$(wildcard $(PACKAGE_PATH)/$(n)/*.c))
INCDIR          += $(foreach n,$(PAKG),-I$(PACKAGE_PATH)/$(n))

ifeq "$(strip $(ROBOT_MODEL))" "UGV_JILONG_3WD"
DDEFS           += -DHF_ROBOT_ID=1
endif
ifeq "$(strip $(ROBOT_MODEL))" "UGV_JILONG_2WD"
DDEFS           += -DHF_ROBOT_ID=2
endif
ifeq "$(strip $(ROBOT_MODEL))" "UGV_STONE_2WD"
DDEFS           += -DHF_ROBOT_ID=3
endif
ifeq "$(strip $(ROBOT_MODEL))" "UGV_STONE_2WD_PLUS"
DDEFS           += -DHF_ROBOT_ID=4
endifyihug
```

flash.mk：
这个文件就是和烧写相关的，一句话，就是调用正确的驱动程序烧写正确程序。主要就是理解这块，就是解析是否使用 BOOTLOADER_MODE

```
ifeq "$(strip $(BOOTLOADER_MODE))" "ENABLE"
BURN_TYPE       = usbttl_bootloader_upload
else
# (stlink/swd/jlink)_(stlink/openocd)_flash  "device"_"driver"_flash
BURN_TYPE       = swd_openocd_flash
endif
```

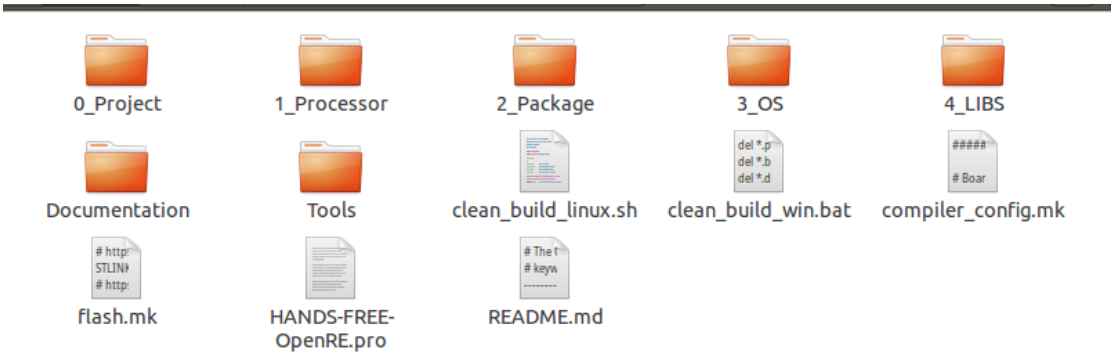
当不用 boot 模式时 BURN_TYPE = swd_openocd_flash 就是代表，使用 jlink 的 SWD 模式，和 openocd 这个驱动程序去烧写。

1_Processor/board.mk:
包含底层源文件和路径，同时解析 BOARD_TYPE，以根据不能的板子型号，做不同的宏定义。

3_OS/os.mk:
包含操作系统层源文件和路径，同时解析 OS_MODULE 变量，以包含不同的文件和路径。

第五篇：OPENRE 架构详解

1.各文件夹介绍



0_Project 文件夹下是工程的入口，里面有该库针对不同平台的士例，以及不同硬件的固件。

1_Processor 是 CPU 有关的固件库和底层接口函数，以及 HANDSFREE 不同电路板的抽象层接口函数的实现。

2_Package 是功能包文件夹，里面有大量的和底层无关或者移植方便的功能包，是整个 Hands Free 的核心

3_OS 文件夹下是操作系统层的支持库，涵盖了实时操作系统(RTOS)，界面(GUI)，文件系统（FATFS），IO 设备(IO,比如网络接口 LWIP,USB 接口)等。

4_LIBS 文件夹下是用于将来放置第三方库，比如 Eigen 库

Documentation 文件夹下是库的说明文档。

Tools 文件夹是用来放编译烧写工具链的。

clean_build_win.bat: windows 下的批处理文件，双击运行，用于清理编译生成的文件，方便保存，压缩，传输。

clean_build_linux.sh: linux 下的批处理文件,同上

HANDS-FREE-OpenRE.pro： QT 工程文件，这里只能使用 Qt Creator 的代码编辑功能，编译是用 make 命令。

另外的.mk 文件就是 makefile 文件。

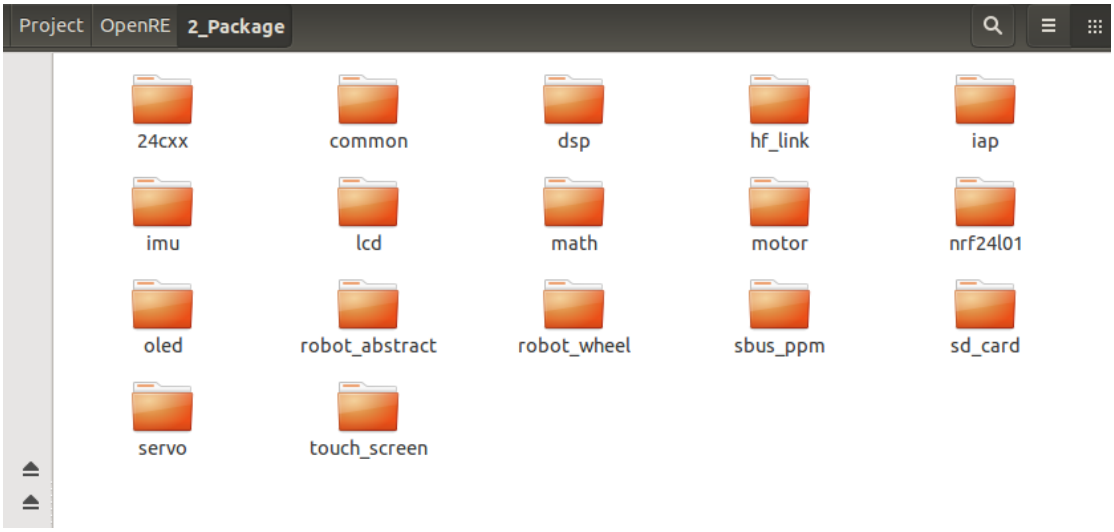
2. OPENRE 的架构介绍

其实也很简单，写库也就是为了使用方便，跨平台，容易移植，具备很多实用的功能，并且整体较为鲁棒，OPENRE 大概也就划为三个部分。

一是硬件抽象，就是不管用什么处理器，用哪个电路板都能为上面提供统一的接口函数，这一块由文件夹 1_Processor 来做。

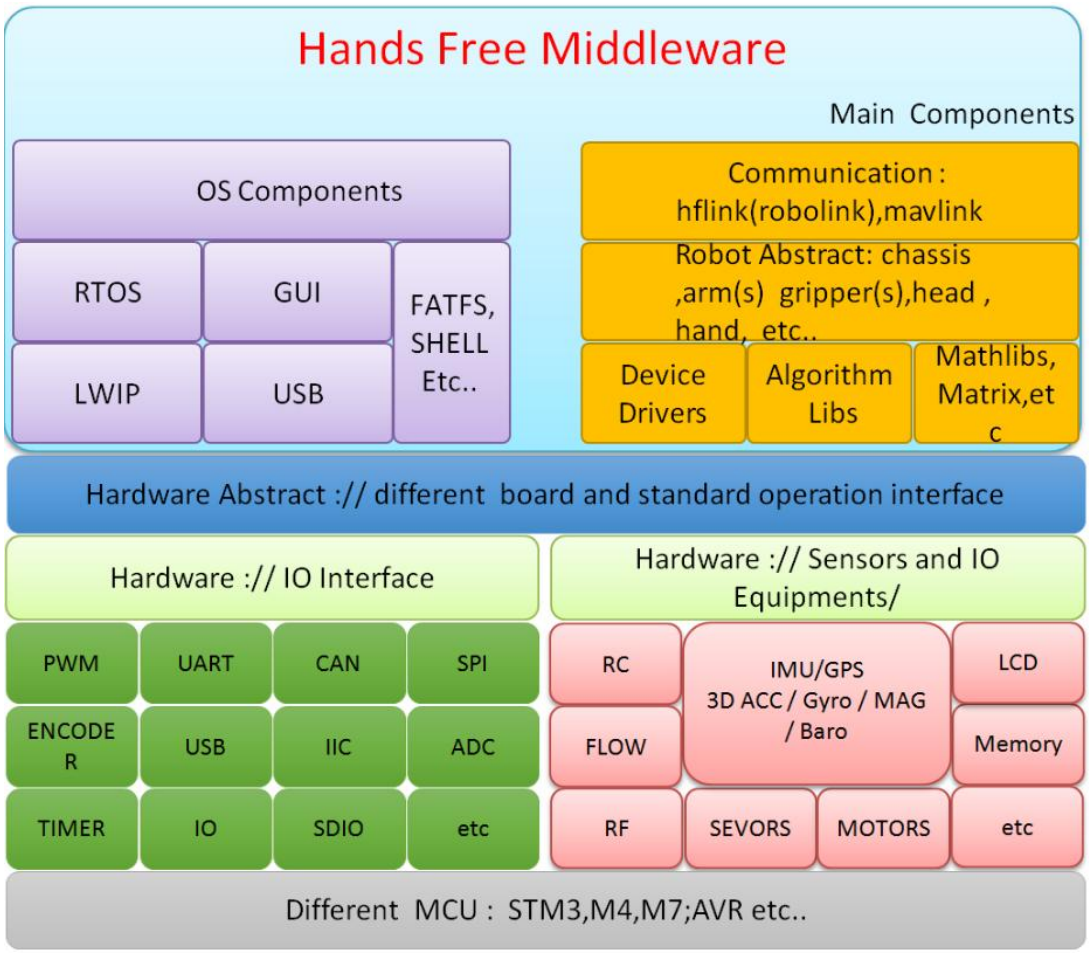
目前封装了 stm32f1 和 f4 的底层配置函数，以及 handsfree 不同电路板的硬件抽象，经过这一层的抽象封装，上面的功能包就可以完全隔离硬件，具备很强的跨平台能力，代码也能变得很简洁。

二是提供很多实用的功能包，这一块由文件夹 2_Package 来做，里面涵盖了伺服设备，传感器，输入输出设备的驱动包，算法包，通信包等各种和机器人有关的功能包。而功能包也是整个库最有用的一部分，对于自己去搭建机器人嵌入式系统的伙伴来说，有这么一个齐全并且被反复验证的库，是非常省事的。



三是提供一个整体的支撑体系，以增加鲁棒性，并扩展各种高级功能，这一块是由 3_OS 文件夹来做，提供了实时系统内核，文件系统，USB，GUI,TCP/IP 等组件的支撑。

OpenRE 目前是以 HANDS FREE 开源机器人项目中的一个组件的形式存在，架构形式如以下框图。



一片 STM32F4 在有的人手上能当小 PC 来用，当然也有人只是用来点个灯，随着微处理器的性能越来越强，像这种单片机上的代码也就能变得越来越复杂。OpenRE 库能帮助我们在短时间内构建复杂控制系统，。

有很多人不熟悉嵌入式的伙伴可能会选择 arduino 来开发，不过个人认为，用来 DIY 简单的巡线小车还是不错的，但是对于移动机器人，飞控这种复杂要求度较高的系统建议还是用 cortex m3 以上处理器，个人觉得 STM32F4 是不错的选择，毕竟 pixhawk，大疆，小米都用这个做他们的飞控，同时使用 OpenRE 库编程可以像 arduino 一样方便。

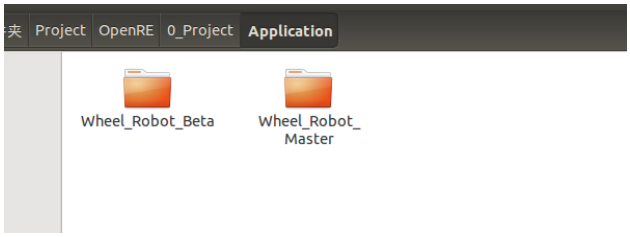
同时 OpenRE 遵循 BSD 3-Clause 开源许可证，使用者可以自由免费使用到任何用途。

工程解说:

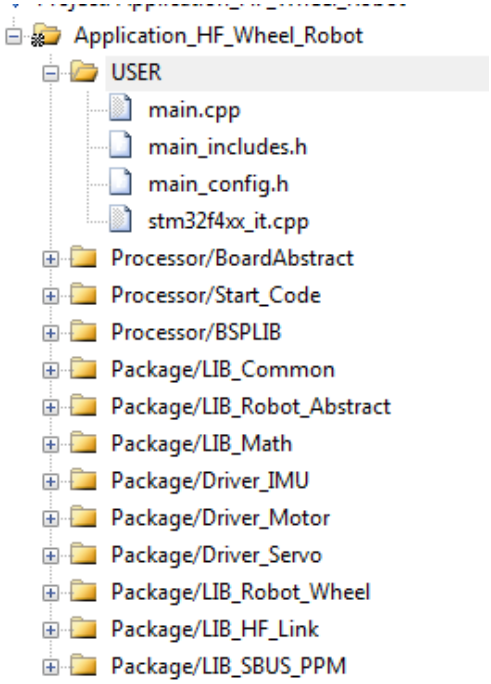
0_Project 下有几个文件夹其中 STM32F1DEMO，STM32F4DEMO 下的是分别在 F1 和 F4 处理器下的几个模板工程。

Bootloader 下是移植的 PX4 的 bootloader。

Application 下的是移动平台的固件工程。Beta 目前是裸奔版的固件工程，Master 是 UCOSIII 操作系统版的固件，功能是一样的，操作系统版可能更加鲁棒。



这里以裸奔版的工程为例来大概解释一下代码。工程包涵以下依赖的包。



Processor/BoardAbstract， Processor/Start_Code ,Processor/BSPLIB 是底层硬件抽象的代码，为功能包提供了统一的接口函数，Package/...则是具有不同功能的包。以下解释这些功能包。

LIB_Robot_Abstract: 这里面只有一个机器人抽象类，主要是把机器人的各种属性，比如轮子半径，速度，头部，机械臂等都以结构体的形式抽象，这样一整个机器人的所有参数都映射着一片内存。 通过对内存的读写操作，来获取机器人各种数据和控制机器人机身。简单的说，这个类代表的就是我们要控制的那个机器人。

LIB_HF_Link: 这样还有一个好处就是方便通信，只需要在上位机（PC）上构建同样的类，实现上下位机内存映射即可，这样操作 PC 上的类，即可实现对机器人的控制，而这一部分是由 LIB_HF_Link 这个包实现的，HFLink 是专门为机器人所定制的一个轻量级通信协议。作用和 mavlink 一样。

Driver_IMU： 这是 IMU 驱动包，里面有加速计，陀螺仪，磁力计，gps 的驱动代码，最终为系统提供一个可靠的姿态和 GPS 坐标。

Driver_Motor： 是电机驱动包，负责所有电机的速度环控制，通过和这个包，你可以指定电机到达一定的速度。

Driver_Servo： 里面有云台的驱动，通过和这个包来控制云台的 pitch 和 yaw。

LIB_Robot_Wheel： 这个包用于管理整个底层系统的控制，通过解析 HFLink 的数据，实现上位机对底层的控制，同时组织管理电机，云台，机械臂等局部的控制。以及解算当前机器人的坐标，获得相关数据参数并不断写进 Robot_Abstract 层里，以便于 HF_Link 把不断更新的数据发送到上位机。

LIB_SBUS_PPM： 是航模遥控器的驱动，可以使用遥控器来控制平台的运动。

有关 HFlink 和 IMU 算法请看代码手册文件夹下的文档。