

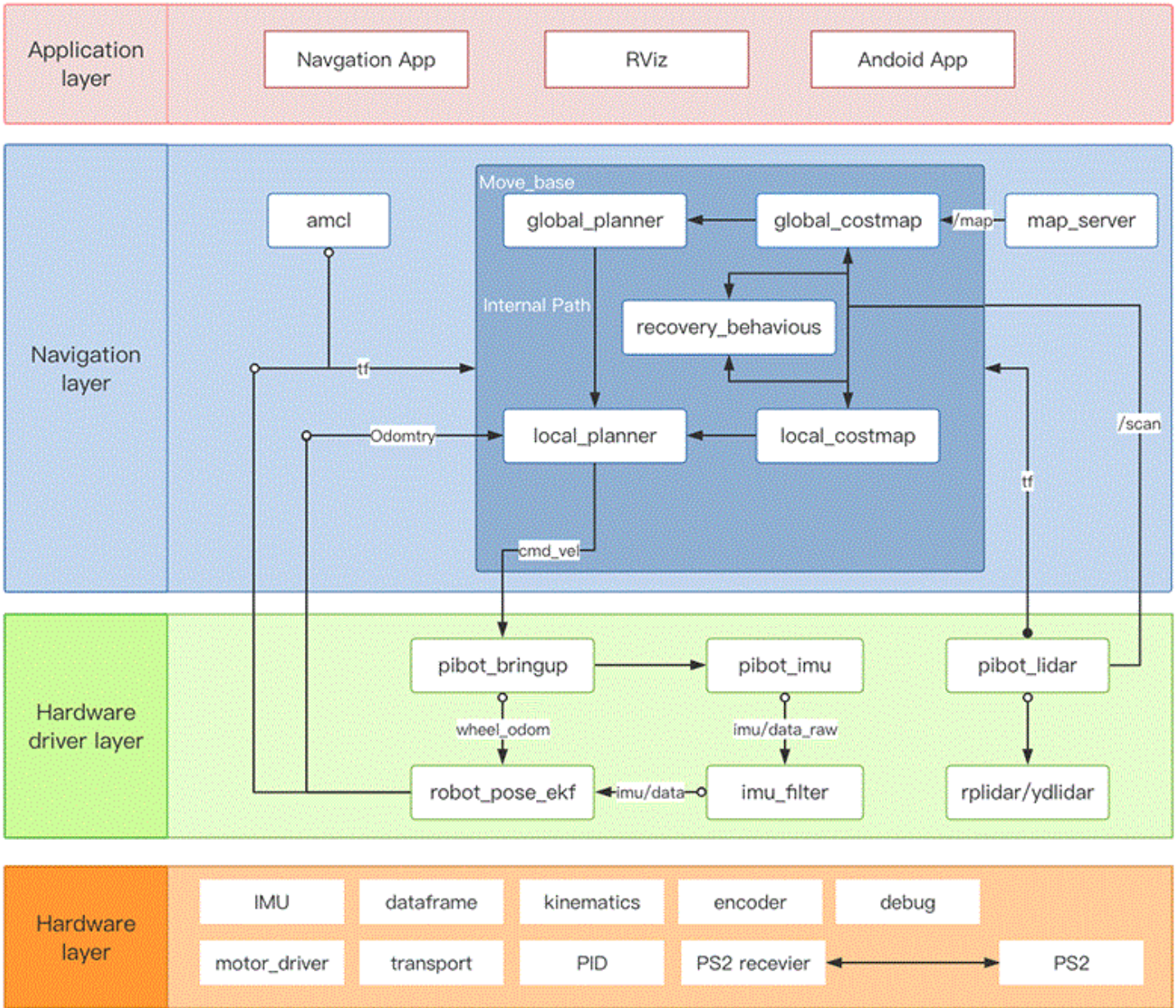
- 1. 概述
- 2. 软件框架
- 3. 下位机开发环境
  - 3.1 环境搭建
    - 3.1.1 Arduino
    - 3.1.2 STM32F1
    - 3.1.3 STM32F4
      - ubuntu下的gcc编译
      - windows的下cubemx(生成keil工程)
- 4. 通讯协议以
- 5. 参数配置
  - 5.1 默认参数
    - 界面配置
  - 5.2 电机顺序与电机方向
    - 5.2.1 电机顺序
      - 两轮差分
      - 四轮差分/四轮麦克纳姆轮
      - 三轮全向
    - 5.2.2 测试电机顺序
    - 5.2.3 电机方向的的确认与调整
    - 5.2.4 电机编码器方向的的确认与调整

# 1. 概述

- PIBOT下位机支持多种主板，Arduino Mega2560、STM32F1及STM32F4等
- PIBOT下位机支持多种运动模型(差分、全向、麦克纳姆轮)，无需重新烧写固件即可修改支持
- PIBOT下位机支持不同的参数的机器人，执行设置相关参数即可

# 2. 软件框架

# 系统框架



橙色部分为下位机的功能模块

## 3. 下位机开发环境

- Arduino Mega 2560 为主控单元，使用 Visual studio code + Platform IO 进行开发，支持 Windows 和 ubuntu 环境
- STM32F1 为主控单元，使用 Keil 进行开发

- STM32F4 为主控单元，Ubuntu 下使用 Visual studio code 进行开发

## 3.1 环境搭建

### 3.1.1 Arduino

具体请参考[Visual Studio Code插件PlatformIO IDE开发Arduino](#)

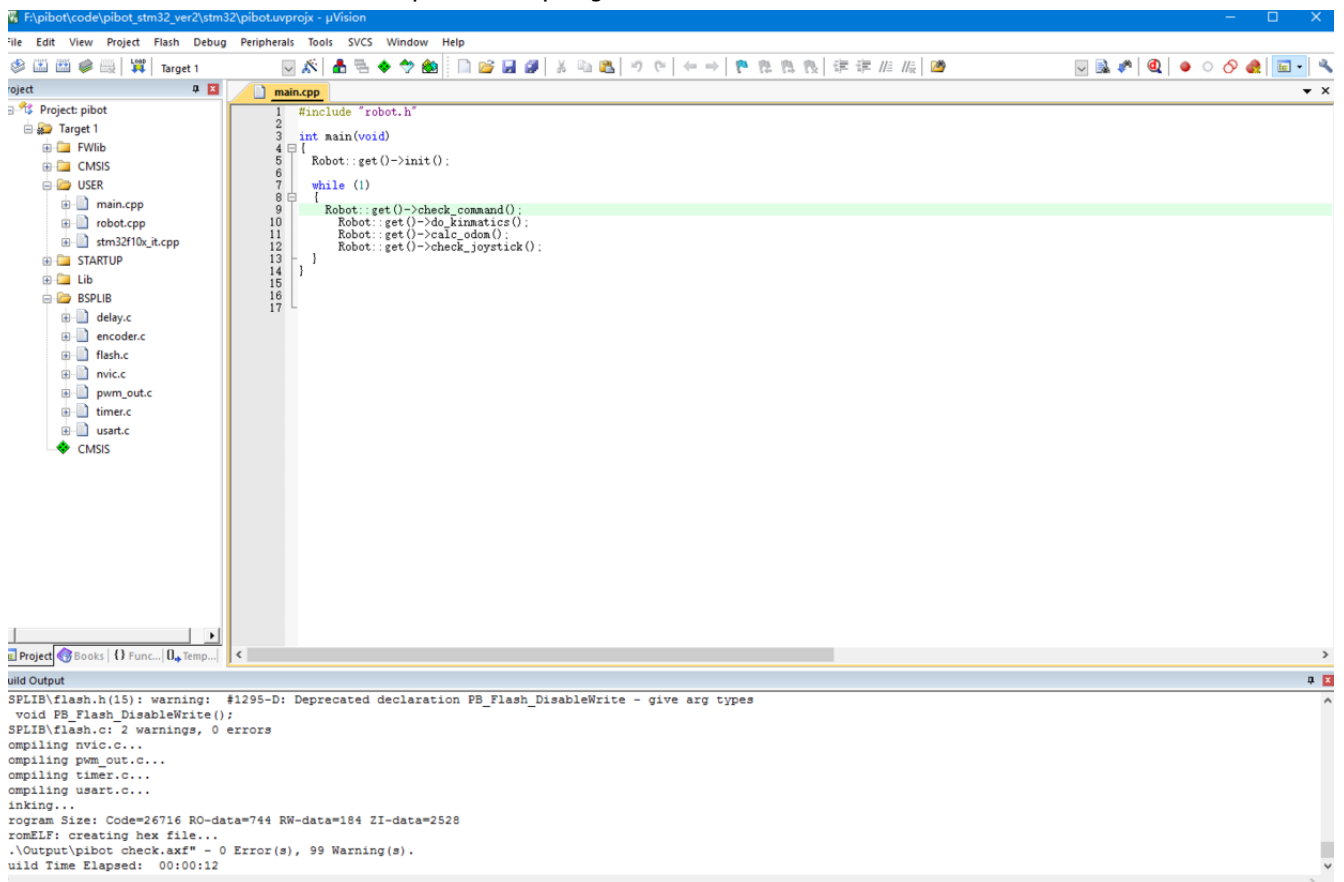
### 3.1.2 STM32F1

- 安装Keil开发环境  
pibot docs/software/MDK520目录提供了Keil5的安装包 MDK520.EXE ，安装过程不再赘述

完成安装后需要继续安装Keil.STM32F1xx\_DFP.2.2.0.pack

- 编译与烧写
  - 编译

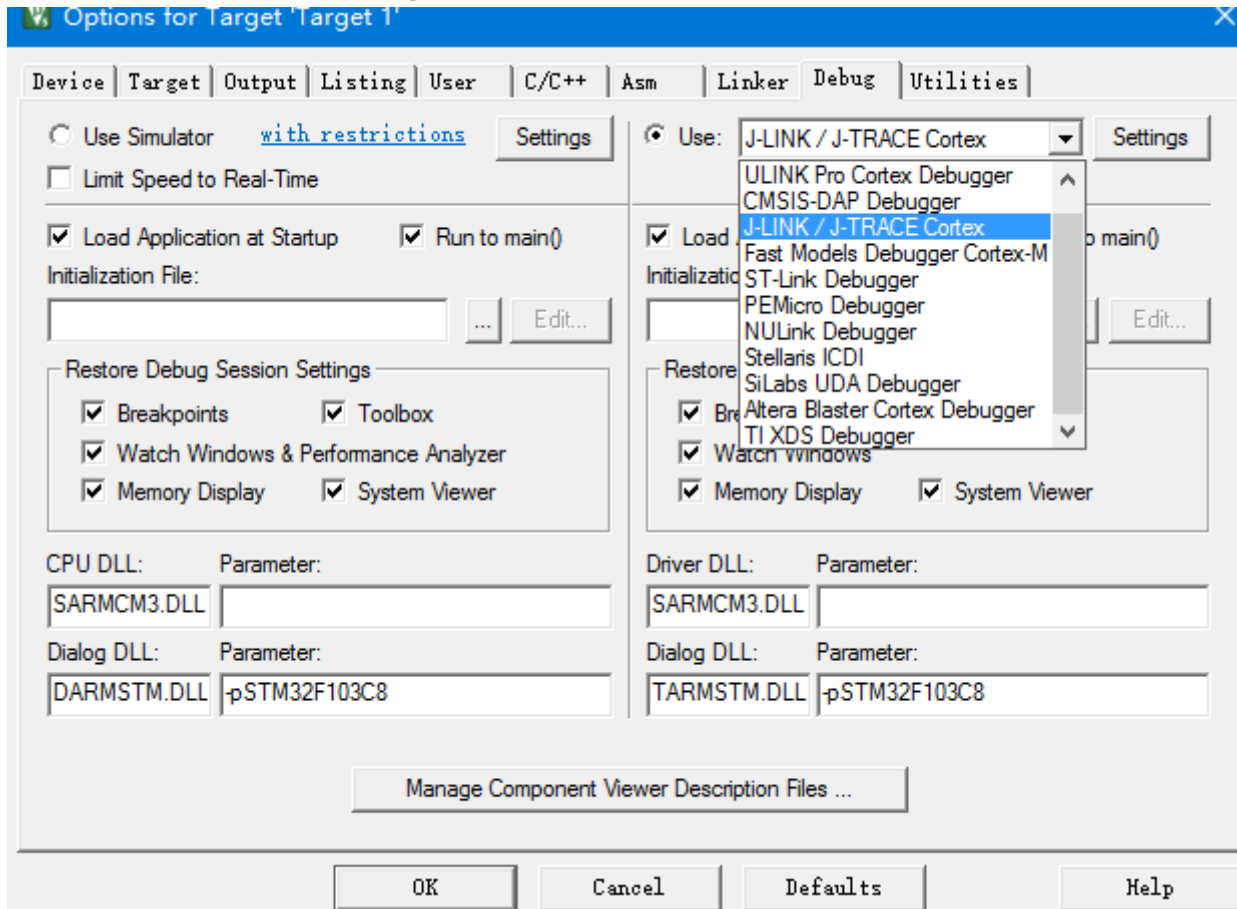
到 PIBOT 源码包中双击打开 pibot.uvprojx 工程,编译



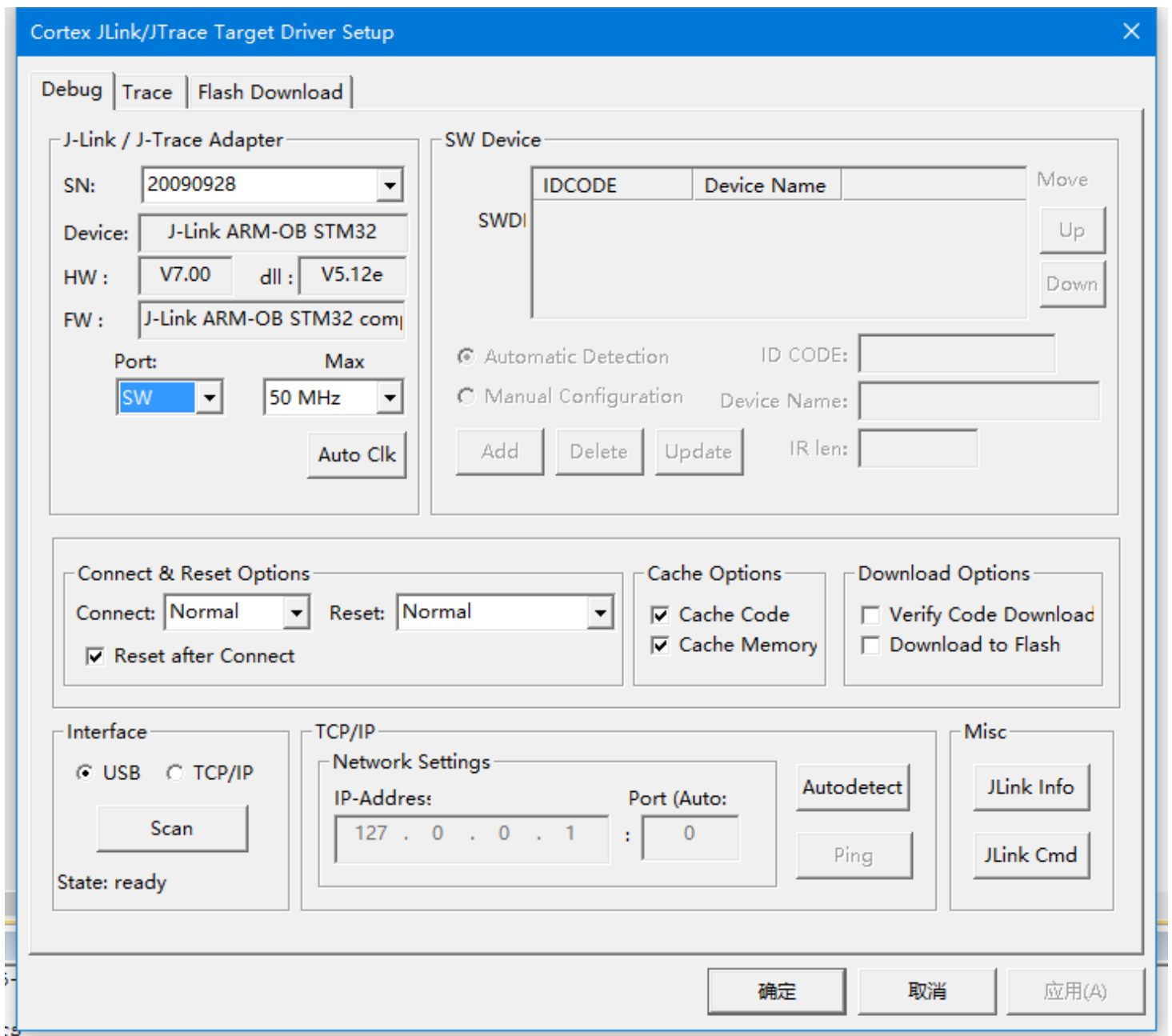
- 烧写程序

这里我们使用 JLink 烧写程序，连接 JLink 至开发板，开发板上电这里的JLink只需要连接GND SWDIO SWCLK三根线

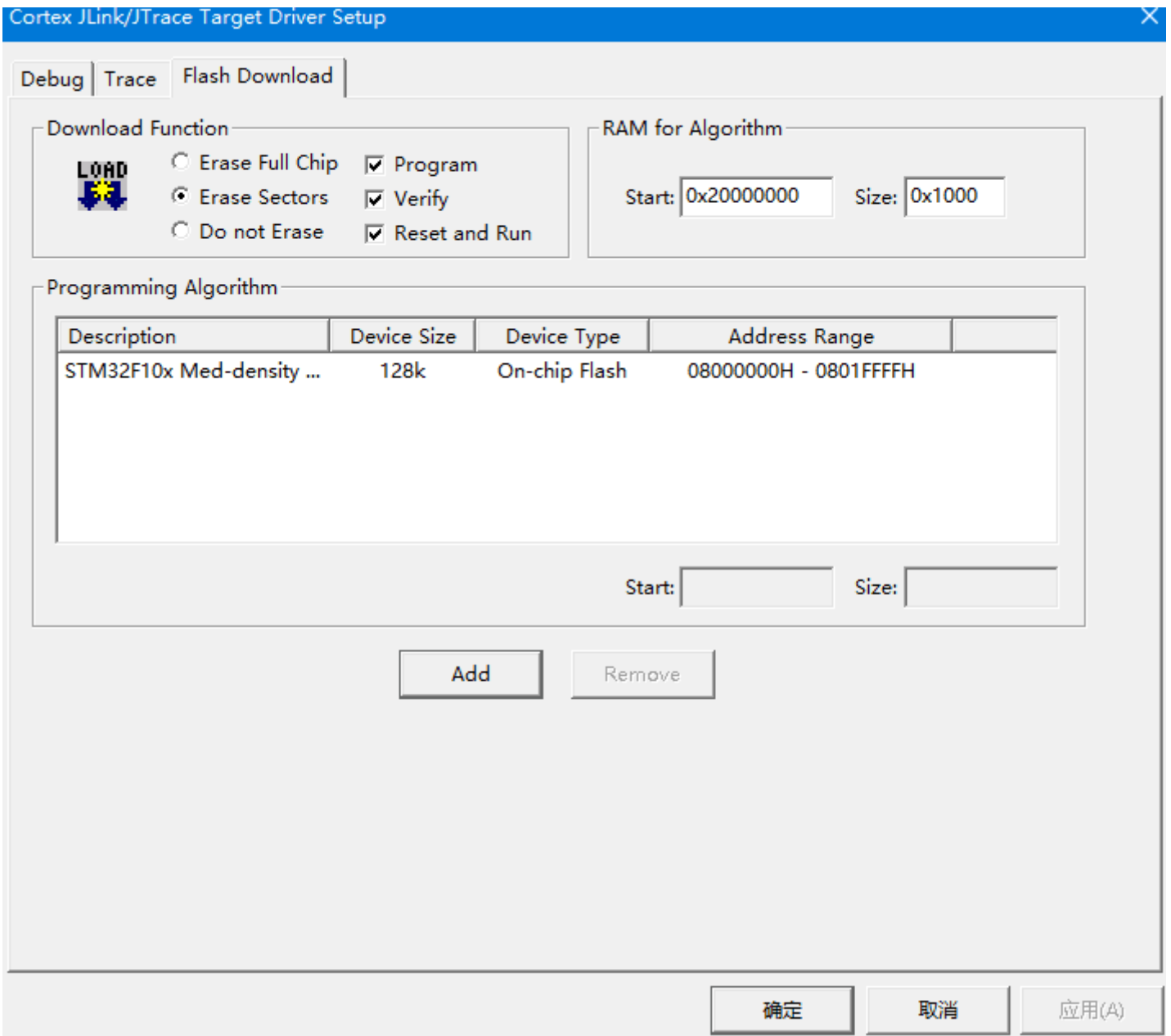
打开工程选项，切换至Debug标签，选择 JLink



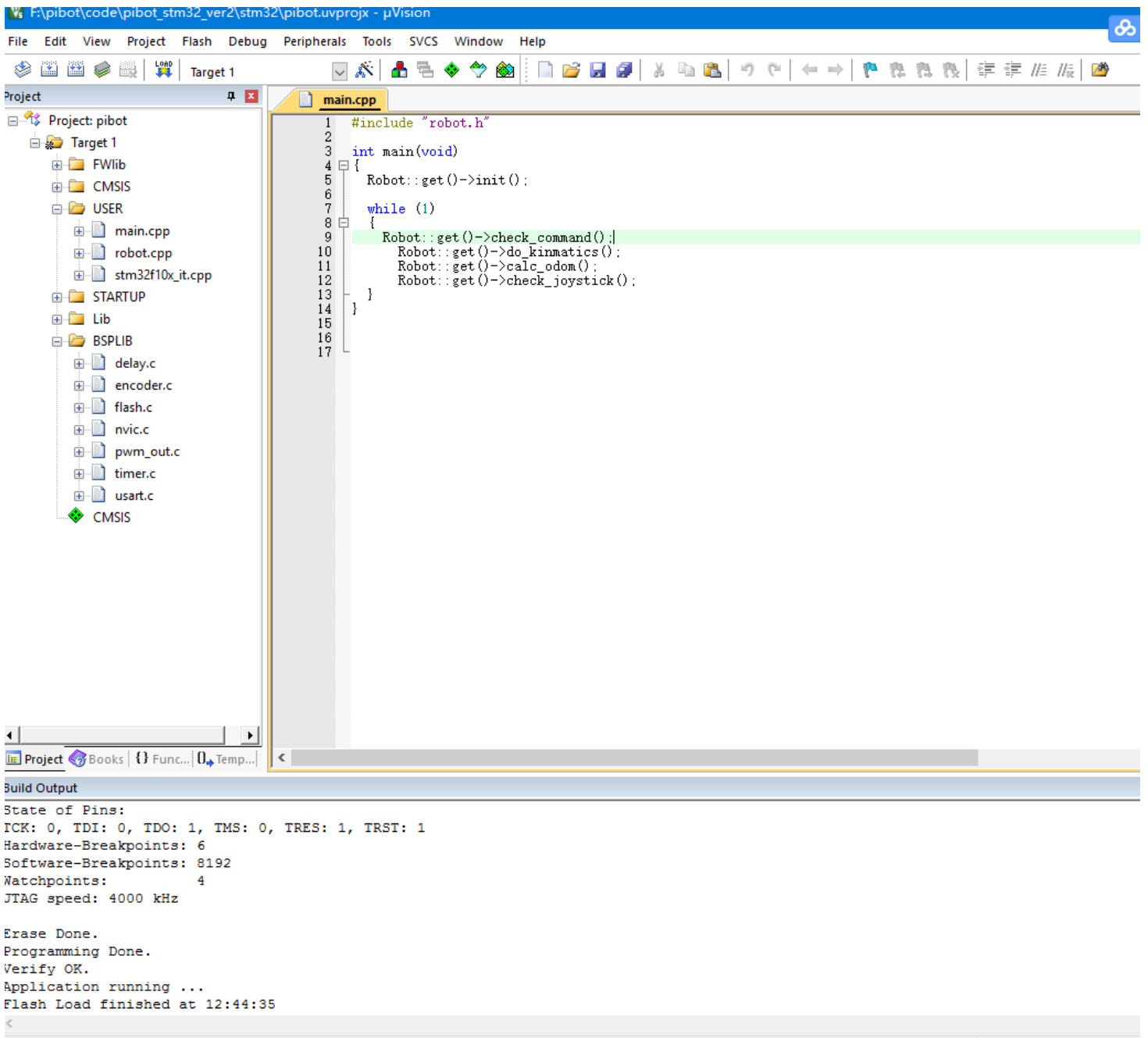
点击 Settings 按钮，Port 选择 SW



切换至Flash Download标签,如下图设置



保存后即可烧写程序



同时可以看到程序运行指示灯在闪烁表示程序在正常运行了

### 3.1.3 STM32F4

#### ubuntu下的gcc编译

- 安装交叉编译器  
sudo apt-get install gcc-arm-none-eabi
- 编译

```
cd stm32
make
```

即可完成编译

- 烧写程序  
配置openocd，使用jlink烧写程序
- 安装openocd  
`sudo apt-get install openocd`
- 烧写  
连接好jlink  
`make burn` 即可完成程序烧写

## windows的下cubemx(生成keil工程)

- 安装Keil开发环境  
pibot/软件工具/MDK520目录 提供了 Keil5 的安装包 MDK520.EXE ， 安装过程不再赘述

完成安装后需要继续安装 Keil.STM32F4xx\_DFP.2.9.0.pack

- 编译  
打开 cubemx\MDK-ARM\ 目录下的工程文件,编译同F1
- 烧写程序  
同F1

# 4. 通讯协议以

通讯协议具体请参见[协议文档](#)

# 5. 参数配置

运动参数出厂时都内置在板子的 EEPROM/FLASH 中， 配置完成驱动板需要重新上电生效

## 5.1 默认参数

执行下面的python脚本即可以设置默认的参数

```
cd ~/pibot_ros/pypiobot/transport
python set_default_params.py
```



执行前使用 `pibot_view_env` 命令查询当前的 `PIBOT_MODEL` 和 `PIBOT_BOARD` 是否配置确认，如果不正确，使用 `pibot_init_env` 脚本配置，具体参见上位机开发

```
pibot@pibot:~$
```

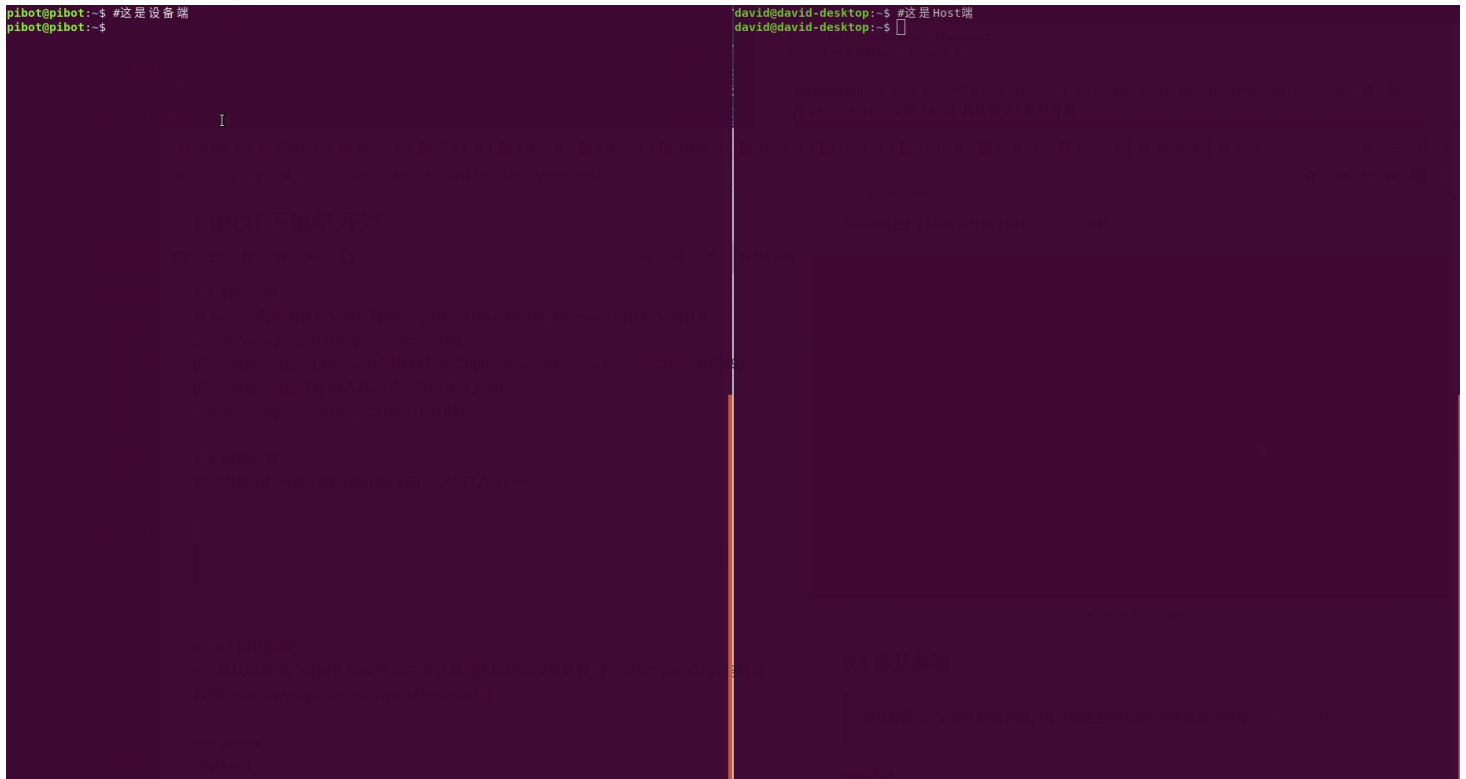


## 界面配置

设备端执行 `pibot_brinup`，Host端执行 `pibot_configure` 即可以看到配置参数的UI界面

```
pibot@pibot:~$ #这是设备端
pibot@pibot:~$

david@david-desktop:~$ #这是Host端
david@david-desktop:~$
```



具体参数定义说明如下

- model\_type 模型参数， 具体为2轮差分/三轮全向/麦克纳姆轮
- wheel\_disiameter 主动轮的直径
- wheel\_track apollo :两个主动轮的轮间距 zeus :三个轮子所在圆直径 hades :四个轮子矩形长宽之和 hera 左右轮距\*系数
- encoder\_resolution 轮子旋转一周编码器变化值的绝对值(一般为4\*编码器分辨率， 如4 \* 11 固件程序做了4倍频)
- motor\_ratio 电机的减速比
- do\_pi\_interval 计算pid的间隔时间， 固定值10
- kp ki kd
- ko 为一个系数， 实际 P I D 参数为 kp/ko ki/ko kd/ko
- cmd\_last\_time 命令激励的超时时间， 即超过该时间没有新的命令会机器人会停止
- max\_v\_liner\_x max\_v\_liner\_y max\_angular\_z 底层速度限制， 遥控器键盘或者导航层下发的速度会被该值限制
- imu\_type 支持的 imu 类型
- motorX\_exchange\_flag 电机方向调整参数， 具体见下面说明
- encoderX\_exchange\_flag 编码器方向调整参数， 具体见下面说明

## 5.2 电机顺序与电机方向

由于电机批次或者电机型号不一致， 首次接入PIBOT需要做电机顺序的矫正

### 5.2.1 电机顺序

#### 两轮差分

```

/*
          x
          ^
          |
          |
          |
y<-----
-----
-
-
-
-
-
-
1-----2
*/

```

## 四轮差分/四轮麦克纳姆轮

```
/*
                x
                ^
                |
                |
y<-----
3-----2
-
-
-
-
-
4-----1
*/
```

## 三轮全向

```
/*
                x
                ^
                |
                |
y<-----
      1
      - -
      - -
      - -
      - -
      - -
2-----3
*/
```

## 5.2.2 测试电机顺序

- 确认小车模型

查看 `pibot_bringup` 打印输出，确认当前配置模型是否跟使用的一致，如不一致则host端打开 `pibot_configure` 调整

```
pi@pi:~$
```



```
1
```

- 测试电机

退出之前的程序， 架空小车（便于观察）

```
cd ~/pibot_ros/pypiobot/transport
# 测试电机1
python test_motors.py 1000 # 命令输完, 电机1会转动
# 根据配置的车上对照电机顺序图, 观察是否是电机1转动

# 测试电机2
python test_motors.py 0 1000 # 命令输完, 电机2会转动
# 根据配置的车上对照电机顺序图, 观察是否是电机2转动

# 测试电机3 (如果有)
python test_motors.py 0 0 1000 # 命令输完, 电机3会转动
# 根据配置的车上对照电机顺序图, 观察是否是电机3转动

# 测试电机4 (如果有)
python test_motors.py 0 0 1000 # 命令输完, 电机4会转动
# 根据配置的车上对照电机顺序图, 观察是否是电机4转动
```



## 5.2.3 电机方向的确认与调整

确保电机顺序以及方向正常,再测试该项

重做测试电机的测试, 根据输入的参数以及实际转的方向确认电机接线方向是否正确

```
cd ~/pibot_ros/pypibot/transport
# 测试电机1
python test_motors.py 1000 # 命令输完, 电机1会转动
# 观察是否是电机1是否顺时针转动(从电机输出轴/外侧观察)
python test_motors.py -1000 0 # 命令输完, 电机1会转动
# 观察是否是电机1是否逆时针转动(从电机输出轴/外侧观察)
```

如果不一致按照之前的在设备端执行 pibot\_brinup , Host端执行 pibot\_configure 修改 motorX\_exchange\_flag \*\*参数

调整完成主板需要重新上电生效

## 5.2.4 电机编码器方向的确认与调整

确保电机顺序以及方向正常,再测试该项

重做测试电机的测试, 根据输入的参数以及编码器的反馈确认编码器

```
cd ~/pibot_ros/pypibot/transport
# 测试电机1
python test_motors.py 1000 0 # 命令输完, 电机1会转动
# 观察输出的日志, 第一列的值是否在变大

python test_motors.py -1000 0 # 命令输完, 电机1会转动
# 观察输出的日志, 第一列的值是否在变小
```

如下结果

- 在控制电机1顺时针转动时候, 编码器1(第一列)逐渐增大则为正常

```
david@david-MS-7808:~/pibot_ros/pypibot/transport$ python test_motors.py 1000 0
[I] 2021-04-23 23:34:02.941 (0xFBE7):set pwm success
[I] 2021-04-23 23:34:02.941 (0xFBE7):*****get encoder count*****
[I] 2021-04-23 23:34:02.948 (0xFBE7):encoder count: 0 0 0
[I] 2021-04-23 23:34:03.964 (0xFBE7):encoder count: 636.0 0.0
[I] 2021-04-23 23:34:04.472 (0xFBE7):encoder count: 1292.0 0.0
[I] 2021-04-23 23:34:04.980 (0xFBE7):encoder count: 1949.0 0.0
[I] 2021-04-23 23:34:05.488 (0xFBE7):encoder count: 2609.0 0.0
[I] 2021-04-23 23:34:05.996 (0xFBE7):encoder count: 3263.0 0.0
[I] 2021-04-23 23:34:06.504 (0xFBE7):encoder count: 3922.0 0.0
[I] 2021-04-23 23:34:07.012 (0xFBE7):encoder count: 4584.0 0.0
```

- 在控制电机1逆时针转动时候, 编码器1(第一列)逐渐减少则为正常

```
david@david-MS-7808:~/pibot_ros/pypiobot/transport$ python test_motors.py -1000 0
[I] 2021-04-23 23:35:51.575 (0xD657):encoder count: 5402.0          0.0          0.0
[I] 2021-04-23 23:35:52.083 (0xD657):encoder count: 4772.0          0.0          0.0
[I] 2021-04-23 23:35:52.591 (0xD657):encoder count: 4122.0          0.0          0.0
[I] 2021-04-23 23:35:53.099 (0xD657):encoder count: 3473.0          0.0          0.0
[I] 2021-04-23 23:35:53.607 (0xD657):encoder count: 2822.0          0.0          0.0
```

依次控制各个电机，如果符合结果则无需调整，如果某一个或几个不符合则输入下面命令打开配置页面调整对应编码器参数 `encoderX_exchange_flag`

调整完成主板需要重新上电生效