程序使用说明

- **为方便说明，以下文档中将 `Visual Studio Code` 简称为 `VSCode` 。**
- **可能由于网络原因下载或安装失败，若失败请尝试科学上网**

目录

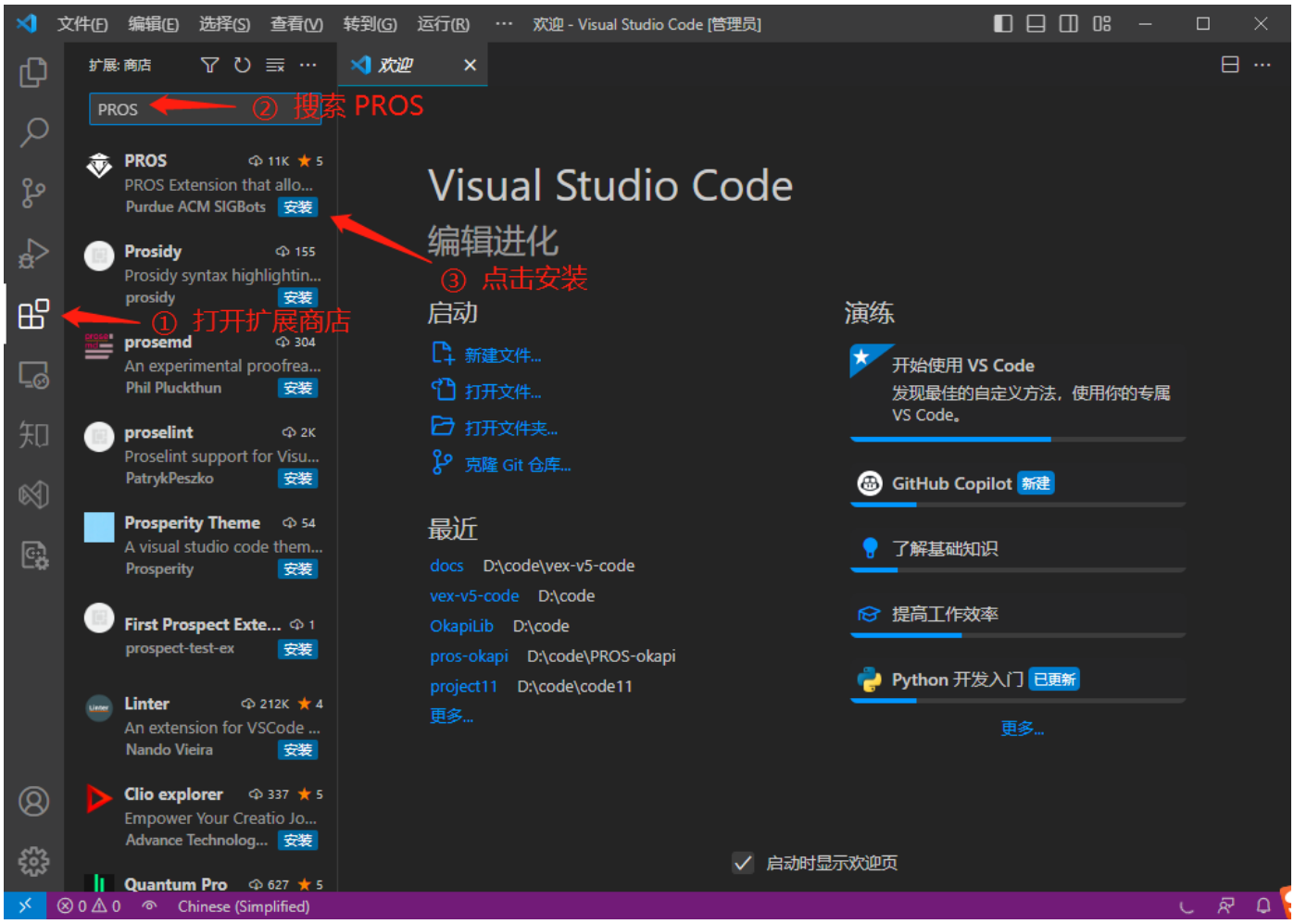
简介

- 本程序依赖于PROS，使用 `C/C++` 编写代码，使用的 `C++` 版本为 `C++17` ，PROS的底层采用FreeRTOS，能保证系统的实时性与可靠性，由于采用PROS作为项目依赖，代码编写自由度较高，可拓展性较强。
- 本程序实现的主要功能有：
  1.使用pid+陀螺仪进行底盘控制，能实现精准的底盘控制。
  2. 采用程序选择器选择自动阶段运行的程序，便于场上调整。
  3. 已实现上层机构的功能,能灵活控制各机构，完成场上功能。

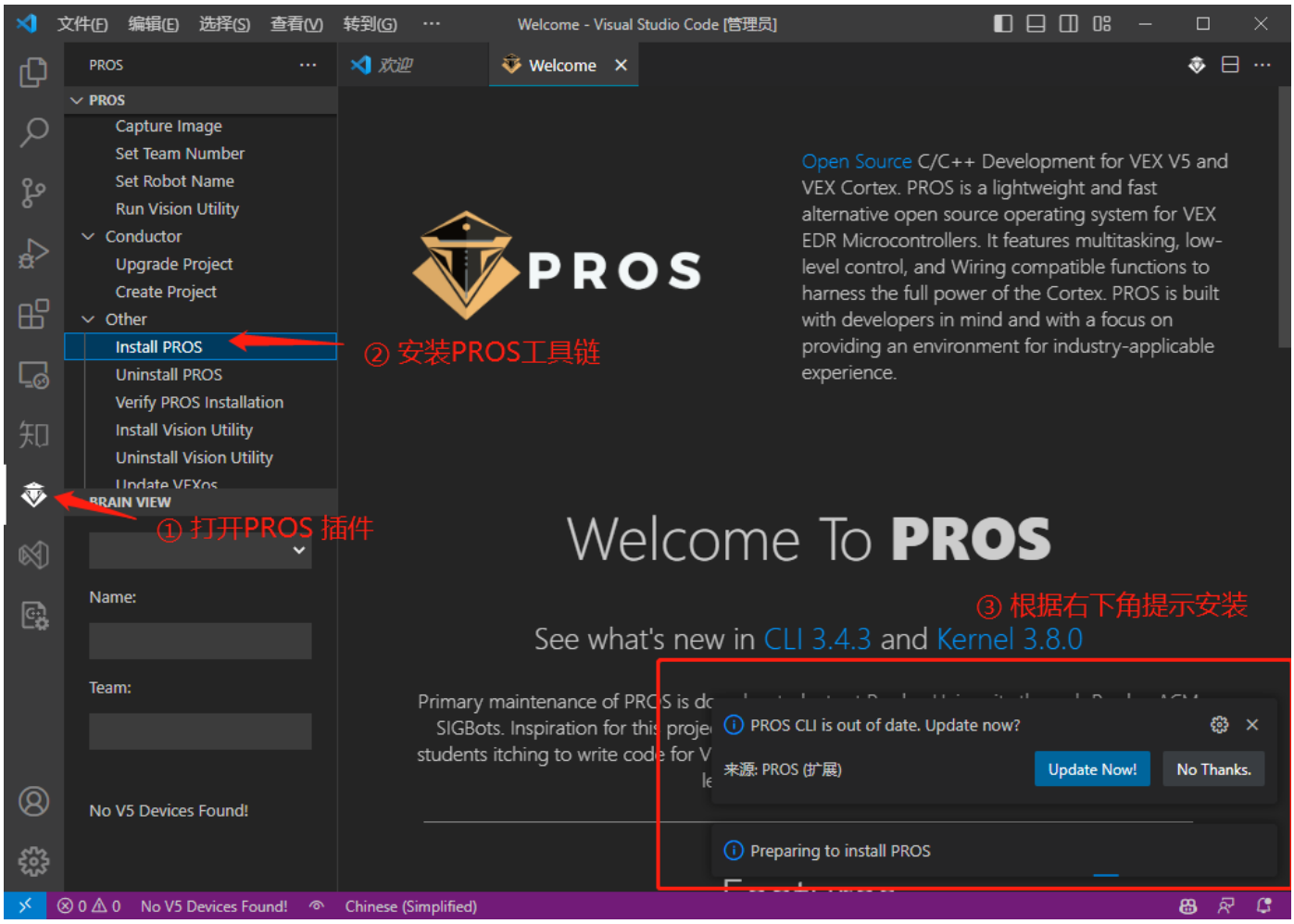
编程环境的搭建和使用方法

PROS的下载与安装

1. 打开VSCode官网(点击这里)，下载并安装VSCode。
2. 打开VSCode，如图所示，下载PROS插件。



3. 打开PROS插件，安装PROS工具链。



4. 等待安装完成。

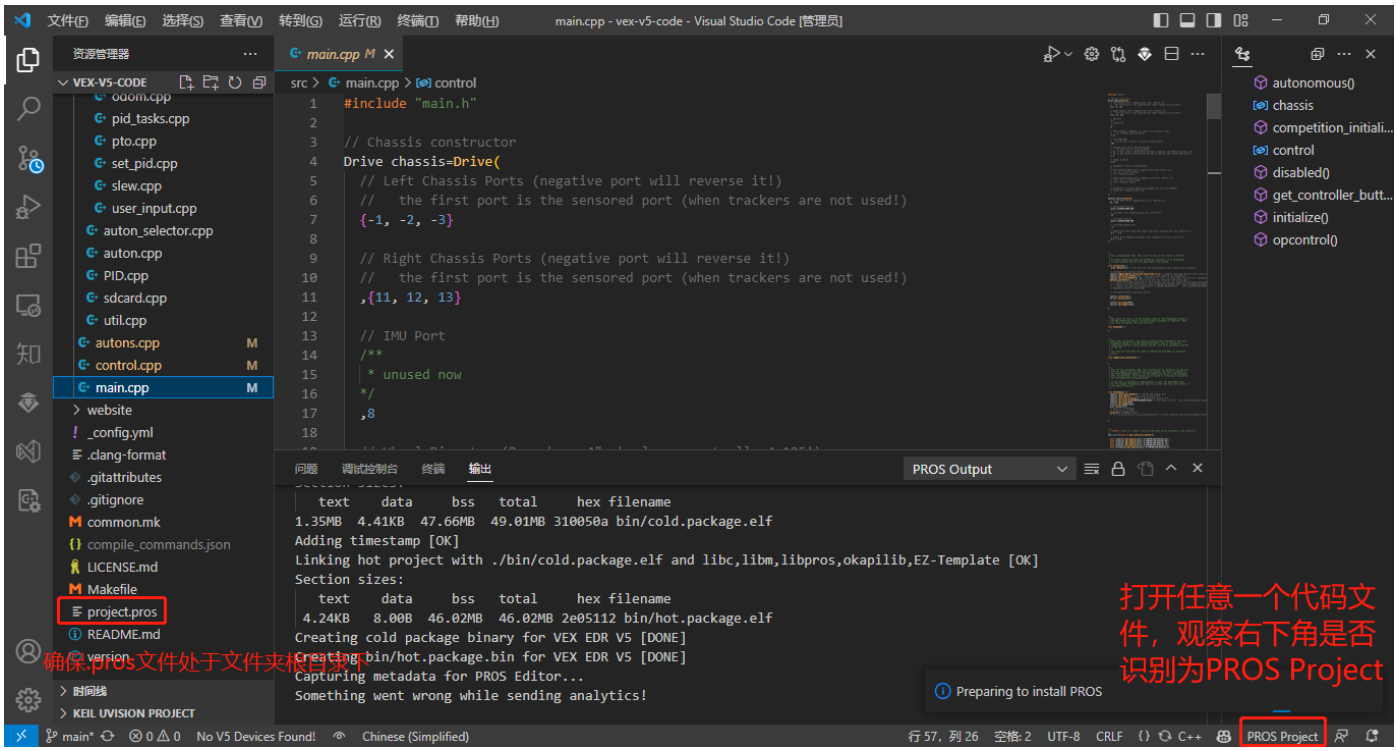
PROS插件的使用方法

特别注意

PROS编译代码时并不会自动保存代码，这意味着**每次更改**之后需**手动保存**代码！

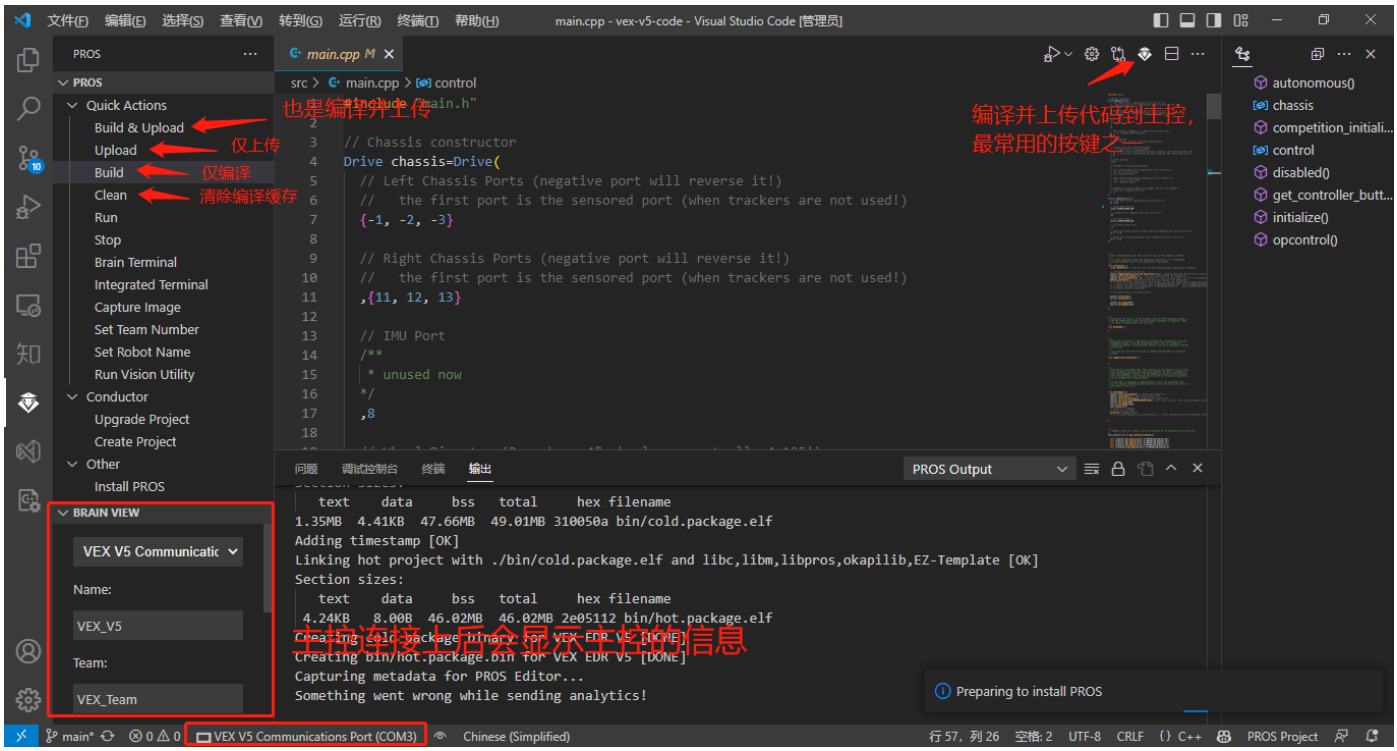
打开项目

- 打开VSCode
- 左上角选择文件-打开文件夹
- 打开PROS项目所属文件夹
- **确认是否已正确打开PROS项目，如下图所示**



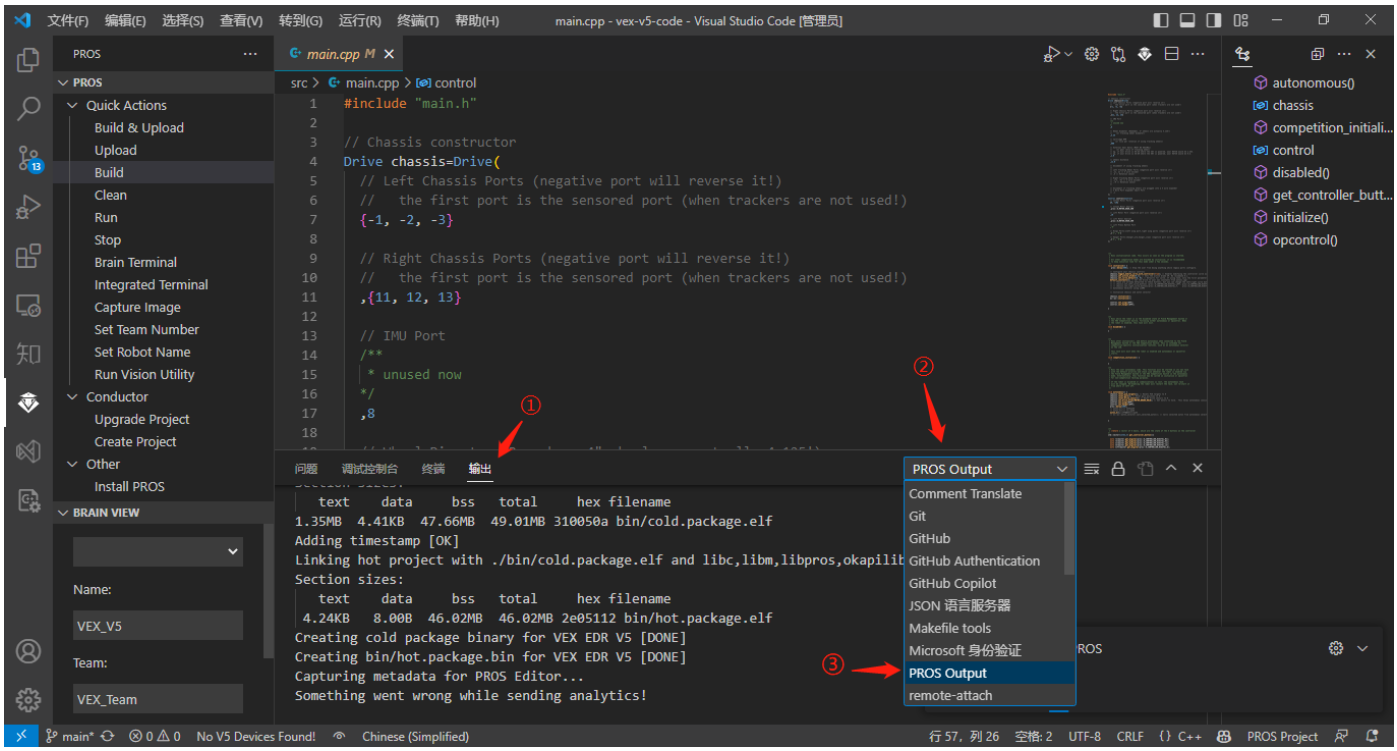
PROS插件常用功能

- 如图所示



查看PROS输出信息（如编译信息）

- 在输出窗口选择PROS OUTPUT，可查看PROS终端输出的信息，如编译进度，上传进度等提示



- PROS在右下角弹出的输出信息并不会自动消失，**请手动关闭消息**，防止因消息过多导致无法获取有效信息。

PROS API 大全

PROS API HOME点击这里

常见问题

1. Q：右下角一直显示 Preparing to install PROS

   A：**插件的小BUG，只要PROS插件各功能正常，如编译、上传等功能便无需理会**
2. Q: 程序打开后出现乱码的现象

   A：

   ① 打开 文件–首选项–设置

   ② 用户–常用设置–文本编辑器–文件–Encoding

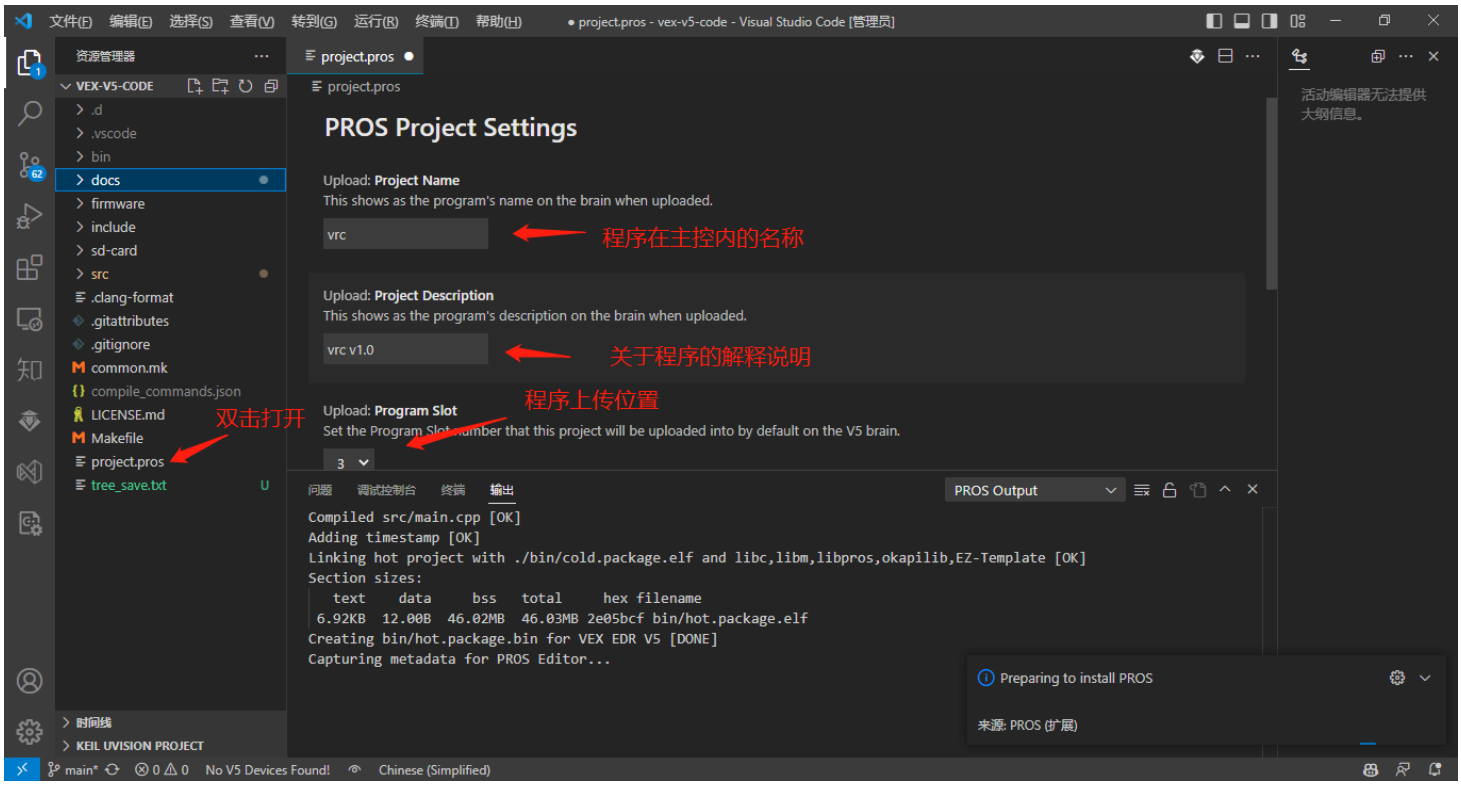
   ③ 将Enconding改为 UTF-8
3. Q:无法上传程序到主控

   A：

   重新打开VSCode，更换usb口，若无效请安装 VEXcode Pro V5 ，该软件的安装程序将自动安装上传程序所需驱动。

PROS 项目的配置

双击project.pros文件即可打开PROS项目的配置界面，可配置项目在主控上的名字、注释、上传位置、图标等



无线传输

- 主控与遥控器的连接方法可以参考这里
- 在将遥控器连接上电脑后，VSCode左下角状态栏会显示 `VEX V5 Controller Port` 设备，则表示遥控器已连接上，此时即可将代码通过遥控器上传至主控。
  - **需要注意的是，在某些情况下使用遥控器传输会较慢，此时VSCode右下角会有所提示，请根据实际需求选择无线或有线传输**
  - 想要更进一步的了解PROS无线传输有关知识，可参考这里

给项目添加新代码文件(cpp/hpp)

若想给项目添加一些新的代码文件，可按照以下步骤进行。

1. 请在 `/include` 目录添加相应的 `.hpp` 文件，在 `/src` 目录添加相应的 `.cpp` 文件。
2. 在 `/include/main.h` 中，在

```
// More includes here...
```

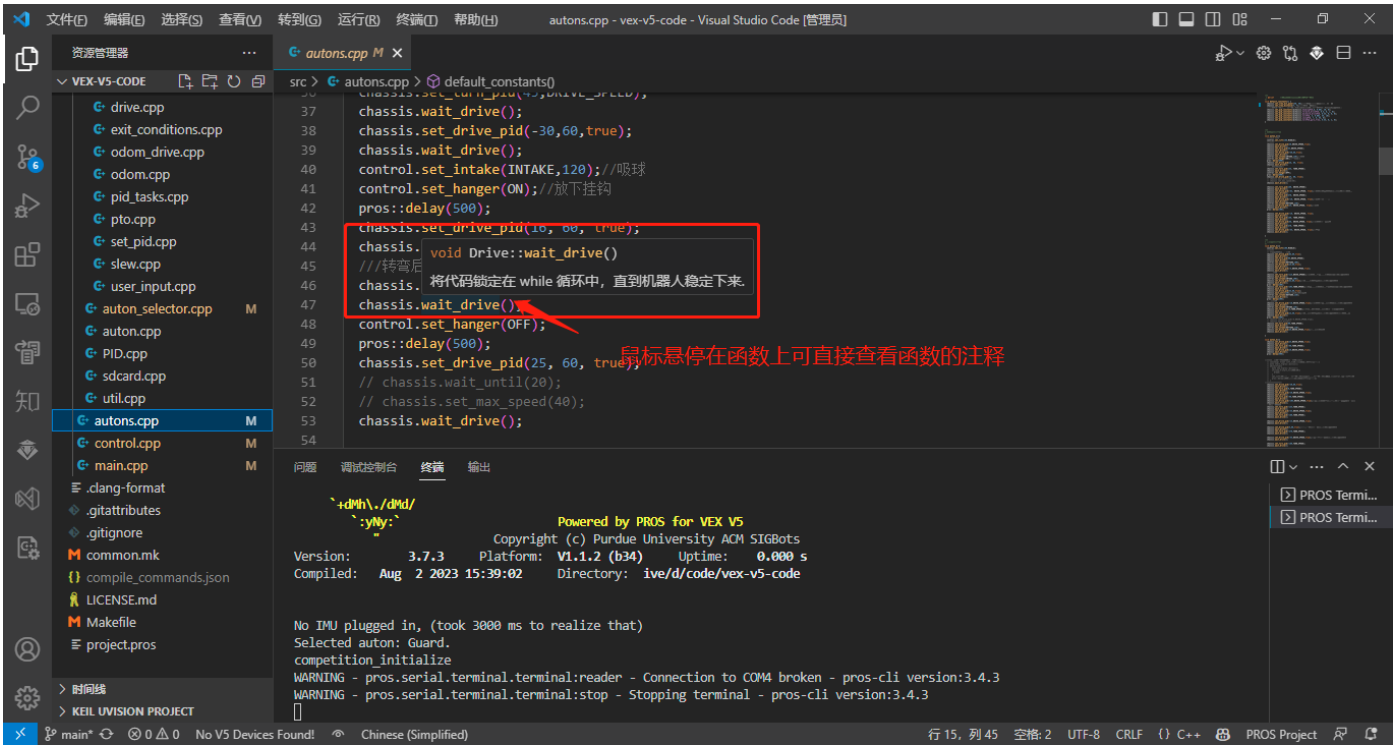
后include相应的头文件。

3. 在 `.hpp` 文件中的第一行，**添加宏定义**

```
#pragma once
```

安装插件：C/C++

- C/C++是由微软推出的插件，为VSCode添加了对C/C++的语言支持，包括编辑（IntelliSense）等功能。安装此插件后可提高代码阅读效率及开发效率。
- 安装方法：同PROS插件安装方法，在拓展商店搜索C/C++后点击安装即可。
- 鼠标悬停在函数上即可查看函数的注释。



- 鼠标左键点击函数+`ctrl` 可直接跳转到函数的声明

代码的整体结构说明

本代码中，所有类、函数的声明均在 `/include` 中，所有的实现均在 `/src` 中，代码的整体架构如下：

```
│  .clang-format
│  .gitattributes
│  .gitignore
│  common.mk                编译配置文件
│  compile_commands.json
│  LICENSE.md
│  Makefile                 编译配置文件，无需改动
│  project.pros             PROS项目配置文件
│
├─.d                        编译生成的文件目录
├─.vscode                   VSCode配置文件目录
├─bin                       编译生成的文件目录
├─docs                      代码相关文档
├─firmware                  编译所依赖的静态库
├─include                   头文件目录
│  │  api.h                 PROS api
│  │  autons.hpp            自动阶段函数声明
│  │  control.hpp           上层机构Control类声明
│  │  main.h                包含整个项目中所使用的通用定义和头文件
│  │
│  ├─display                UI(LVGL)头文件
│  │
│  ├─EZ-Template            模板头文件目录
│  │  │  api.hpp            模板 api
│  │  │  auton.hpp          程序选择器声明
│  │  │  auton_selector.hpp 程序选择器声明
│  │  │  PID.hpp            PID类声明
│  │  │  sdcard.hpp         sdcard相关声明
│  │  │  util.hpp           一些编程时会用到的工具函数声明
│  │  │  PID_Logger.hpp     用于记录pid数据到sd卡
│  │  │
│  │  └─drive               底盘运动相关头文件目录
│  │          drive.hpp     底盘运动Drive类声明
│  │          odom.hpp      Odometry定位声明
│  │
│  │
│  └─pros                   PROS库函数
│
│
└─src                       代码实现目录
   │  autons.cpp            自动阶段函数实现
   │  control.cpp           上层结构控制实现
   │  main.cpp              程序的开始，main.cpp
   │
   └─EZ-Template            模版的实现
       │  auton.cpp         程序选择器
       │  auton_selector.cpp 程序选择器
       │  PID.cpp           PID实现
       │  sdcard.cpp        sdcard使用
       │  util.cpp          工具函数实现
```

```
|  PID_Logger.cpp      用于记录pid数据到sd卡
└drive                 底盘运动实现
        drive.cpp      底盘控制函数
        exit_conditions.cpp    pid退出条件
        odom.cpp               Odometry实现
        odom_drive.cpp         Odometry实现
        pid_tasks.cpp          pid线程的实现
        pto.cpp                pto底盘
        set_pid.cpp            pid设置
        slew.cpp               pid缓加速
        user_input.cpp         控制器相关
```

在程序内配置端口与参数

配置底盘参数及底盘电机端口

1. 使用VSCode打开VEX-VRC-Code-v1.0文件夹，并检查是否已正确打开为PROS项目。
2. 打开src/main.cpp文件。
3. 找到 `Drive chassis` 变量，根据注释配置相关参数。

配置上层结构端口

1. 打开src/main.cpp文件
2. 找到 `Control control` 变量，根据注释配置相关参数。

使用竞赛模板编写代码

- 打开src/main.cpp文件，文件内有以下函数

```
void autonomous(void);
void initialize(void);
void disabled(void);
void competition_initialize(void);
void opcontrol(void);
```

- `void autonomous(void)` 函数将会在自动阶段开始时自动运行。
- `void opcontrol(void)` 函数将会在手动阶段开始时自动运行。
- 无论在何种情况下， `void initialize(void)` 函数总是在程序刚启动的时候就执行。
- `void disabled(void)` 函数将会在竞赛场控**切换到** `DISABLE` 模式时调用。

- `void competition_initialize(void)` 函数只有在连接竞赛场控时才会运行，且将在竞赛开始后自动退出。
- **注意**：在竞赛开始前无法控制电机，即在连接上场控模块后，除

  ```
  void autonomous(void)
  void opcontrol(void)
  ```

  函数外，所有试图控制电机的操作都是无效的。

底盘控制

注：

1. **必须**安装陀螺仪才能启用底盘控制!
2. 陀螺仪度数在顺时针方向增加。

底盘控制的参数整定(PID)

- PID参数在 `/src/autons.cpp` 中的 `void default_constants()` 函数中配置
- 本项目已给出预置的PID参数，建议根据以下方法对该参数进行微调，也可以自行整定参数。

Step 1 - kP（比例）

- PID参数的整定通常从kP开始。将kP设置为某一数字，然后让机器人运动。机器人要么低于目标值（kP 太低），要么机器人围绕目标值周围振荡（kP 太高）。
- 我们一般希望 kP 稍微振荡，通常为一两次反弹（不是反复振荡）。

Step 2 - kD（微分）

找到合适的kP值后，我们可以调整 kD。增加 kD 直到振荡消失。这时机器人的动作应比仅使用P控制更为敏捷。我们可以将kD参数理解为“阻尼”,作用是抑制过冲现象。

Step 3 - 重复Step1与Step2

重复Step 1 和Step 2，直到 kD 无法消除运动中的振荡。那么上一次实验的最佳值就是我们想要的参数。

Step 4 - kI（积分）

- 有时我们可能需要一些额外的动力才能让机器人达到目标值（即减小稳态误差），这时我们可以给使用PID控制。对KI参数的调整应相当谨慎，因其影响通常成指数级增长。第四个参数是启用积分控制器的最小误差值，只有在误差大于该参数时才会启用积分控制器。对于转弯来说，通常将第四个参数设置为15。
- 增加 kI 直到产生任何轻微干扰。可能需要在调整 kI 的同时调整 kD。

前进/后退的参数整定

Step 1 - 调整前进/后退的kP、kD

- 使用上述整定方法，修改 kP、kD。如果选择使用积分控制器，请修改 kI、Start_i。（不建议在前进/后退时使用）
- 由于机器人的重心可能靠前（或靠后），在使用相同的参数时前进和后退的效果也可能不同，这时可以为前进/后退分别设置不同的参数。

```
chassis.set_pid_constants(&chassis.forward_drivePID, 0.45, 0, 5, 0);
chassis.set_pid_constants(&chassis.backward_drivePID, 0.45, 0, 5, 0);
```

Step 2 - 方向修正（Heading Correction）

- 方向修正会在机器人向前/向后行驶时保持面向同一方向（角度）。 此常数通常高于其他 PID 控制器，因为只校正了几个角度的误差。
- 之前所写的整定方法也可用于调整此常量。 增加kP直到有一点振荡，增加kD直到它消失，并重复这些步骤。

```
chassis.set_pid_constants(&chassis.headingPID, 11, 0, 20, 0);
```

Step 3 - 缓慢启动（slew）

- `slew_min_power()` 是机器人在使用缓慢启动时的启动速度。 `slew_distance()` 是机器人从最小速度增加到我们设定的最大速度的距离。
- `slew_min_power()` 应在保证机器人车轮不打滑的情况下尽可能高。
- `slew_distance()` 应在保证机器人车轮不打滑的情况下尽可能小。

```
chassis.set_slew_min_power(80, 80);
chassis.set_slew_distance(7, 7);
```

转向的参数整定

Step 1 - 调整kP与kD

- 使用与上述相同的步骤，将kP设置为某个数字并对其进行修改，直到出现轻微振荡，一次或两次反弹。
- 增加kD，直到振荡消失。
- 重复以上步骤直到kD无法修正震荡

```
chassis.set_pid_constants(&chassis.turnPID, 5, 0.003, 35, 15);
chassis.set_pid_constants(&chassis.swingPID, 7, 0, 45, 0);
```

Step 2 - 调整kI

在转向时，PD控制可能无法满足我们的需求，这时我们就需要用到PID控制。
调整方法如下：

- 有时我们可能需要一些额外的动力才能让机器人达到目标值（即减小稳态误差），这时我们可以给使用PID控制。对KI参数的调整应相当谨慎，因其影响通常成指数级增长。第四个参数是启用积分控制器的最小误差值，只有在误差大于该参数时才会启用积分控制器。对于转弯来说，通常将第四个参数设置为15。
- 增加 kI 直到对系统产生任何扰动（振荡等）。每次增加的kI值应尽可能小，可能需要在调整 kI 的同时调整 kD。

```
chassis.set_pid_constants(&chassis.turnPID, 5, 0.003, 35, 15);
```

如何让底盘运动

控制底盘的函数主要有以下几个：

```
 /**
 * 使用PID让机器人前进/后退指定距离
 *
 * \param target
 *        目标值（英寸）,取负值将后退
 * \param speed
 *        0 to 127, 机器人运动时的最大速度
 * \param slew_on
 *        是否启用缓加速，默认关闭
 * \param toggle_heading
 *        是否使用陀螺仪修正前进方向，默认开启
 */
void Drive::set_drive_pid(double target, int speed, bool slew_on = false, bool toggle_heading


 /**
 * 使用PID让机器人原地转向到指定角度
 *
 * \param target
 *        目标角度（角度制）,0度为机器人加载程序时的朝向。
 * \param speed
 *        0 to 127, 机器人运动时的最大速度
 */
void Drive::set_turn_pid(double target, int speed);


 /**
 * 仅使用左侧或右侧转动。
 *
 * \param type
 *        LEFT_SWING 或 RIGHT_SWING
 * \param target
 *        目标角度（角度制）
 * \param speed
 *        0 to 127, 运动时的最大速度
 */
void Drive::set_swing_pid(e_swing type, double target, int speed);


/**
 * 将代码锁定在 while 循环中，直到机器人走（转）到指定位置（角度）稳定下来。
 */
void Drive::wait_drive();


 /**
 * 将代码锁定在 while 循环中，直到走过指定路程。
 *
 * \param target
 *        当直线行驶时，单位是英寸。当转弯时，单位是角度（角度制）
```

```
 */
void Drive::wait_until(double target);
```

关于以上函数如何使用，可以参考autons.cpp中的使用方法。

上层机构的控制

- 上层机构的控制，主要在 `/src/control.cpp` 中实现，各函数的声明和注释均在 `/include/control.hpp` 中。如何使用这些函数可以参考 `/src/main.cpp` 中的 `void opcontrol()` 函数。
- `Control` 模块将作为一个单独的任务（线程）执行，不会阻塞调用 `Control set/get` 方法的任务（线程）。

程序选择器

- 所有通过

  ```
  ez::as::auton_selector.add_autons();
  ```

  注册过的函数（自动程序）都可通过程序选择器选择，并在比赛开始后执行所选择的函数（自动程序）。本项目在 `src/main.cpp  void initialize()` 函数中注册自动程序。
- 若主控上**已插入sd卡**，则会记住每次选择的程序，如我在上次运行程序时**最后**选择了②号程序，则在下次启动程序时也将**默认**从②号程序开始选择。
- 界面简介

快速开始

1. 在 /src/main.cpp 中配置机器人的端口，如底盘电机、陀螺仪、电磁阀等设备的端口。
2. ◦ 在 /src/autons.cpp 中配置自动任务的底盘速度

```
/**
* 设置正常运行时的底盘速度
*/
#define DRIVE_SPEED  100
#define TURN_SPEED   80
#define SWING_SPEED  80
```

◦ 在 /src/control.cpp 中配置投球机的角度

```
/**
 * @brief 投球机位于顶部和中间位置时的编码器
 * 值，编码器0点在底部
 */
 #define CATAPULT_UP_POS      100.0
 #define CATAPULT_MIDDLE_POS  1600.0
 #define CATAPULT_DOWN_POS    0.0
```

3. 测试PID参数是否需要调整，具体做法为使用预置好的自动任务 void test_pid() 函数测试机器人的运行是否平稳。

    - PID参数在 `/src/autons.cpp` 中的 `void default_constants()` 函数中配置
4. 在 `/src/main.cpp` 中的 `void opcontrol()` 函数中根据需求更改控制器按钮所对应的功能
5. 根据需求更改 `/src/control.cpp` 中的函数
6. 在 `src/autons.cpp` 和 `/include/autons.hpp` 中添加或修改自动任务所对应的函数。
7. 在 `src/main.cpp  void initialize()` 函数里的

```
ez::as::auton_selector.add_autons({
  Auton("Guard.", auton_1),
  Auton("Attack.", auton_2),
  Auton("1min. ", auton_3),
});
```

上添加自动任务函数（若有）。
例如我要添加 `void test_pid()` 函数，要在屏幕上显示的文字是 `test pid` ,方法如下：

```
ez::as::auton_selector.add_autons({
  Auton("Guard.", auton_1),
  Auton("Attack.", auton_2),
  Auton("1min. ", auton_3),
  Auton("test pid",test_pid),
});
```

附录：常用API

底盘控制常用API

```
 /**
 * 使用PID让机器人前进/后退指定距离
 *
 * \param target
 *        目标值（英寸）,取负值将后退
 * \param speed
 *        0 to 127, 机器人运动时的最大速度
 * \param slew_on
 *        是否启用缓加速，默认关闭
 * \param toggle_heading
 *        是否使用陀螺仪修正前进方向，默认开启
 */
void Drive::set_drive_pid(double target, int speed, bool slew_on = false, bool toggle_heading


 /**
 * 使用PID让机器人原地转向到指定角度
 *
 * \param target
 *        目标角度（角度制）,0度为机器人加载程序时的朝向。
 * \param speed
 *        0 to 127, 机器人运动时的最大速度
 */
void Drive::set_turn_pid(double target, int speed);


 /**
 * 仅使用左侧或右侧转动。
 *
 * \param type
 *        LEFT_SWING 或 RIGHT_SWING
 * \param target
 *        目标角度（角度制）
 * \param speed
 *        0 to 127, 运动时的最大速度
 */
void Drive::set_swing_pid(e_swing type, double target, int speed);


/**
 * 将代码锁定在 while 循环中，直到机器人走（转）到指定位置（角度）稳定下来。
 */
void Drive::wait_drive();
```

```
 /**
 * 将代码锁定在 while 循环中，直到走过指定路程。
 *
 * \param target
 *        当直线行驶时，单位是英寸。当转弯时，单位是角度（角度制）
 */
void Drive::wait_until(double target);
```

- 假设我想让机器人以80的速度前进20英寸，则：

```
chassis.set_drive_pid(20,80);
chassis.wait_drive();
```

上层机构常用API

```
inline void set_intake_state(Control_State state){
    intake_state=state;
    drive_intake=true;
}
/**
 * \param state 设置wings的模式
 * - ON: 放出
 * - OFF: 收起
*/
inline void set_wings_state(Control_State state){
    wings_state=state;
    drive_wings=true;
}
/**
 * \param state 设置catapult的模式
 * - UP: 升起
 * - MIDDLE: 中间
 * - DOWN: 放下
*/
inline void set_catapult_state(Catapult_State state){
    catapult_state=state;
    drive_catapult=true;
}

/**
 * \param state 设置hanger的模式
 * - ON: 打开
 * - OFF: 关闭
*/
inline void set_hanger_state(Control_State state){
    hanger_state=state;
    drive_hanger=true;
}
```

- 可参考autons.cpp中的使用方法使用
- 假设我想打开挡板，则：

```
    control.set_wings_state(ON);
```