

00A0 2203∃ 2200∀ 2286⊆ 2713x 27FA⇔ 221A√ 221B³/ 2295⊕ 2297⊗ UTF8gbsn

SunFounder PiArm 机器人

SunFounder

2022 年 07 月 01 日

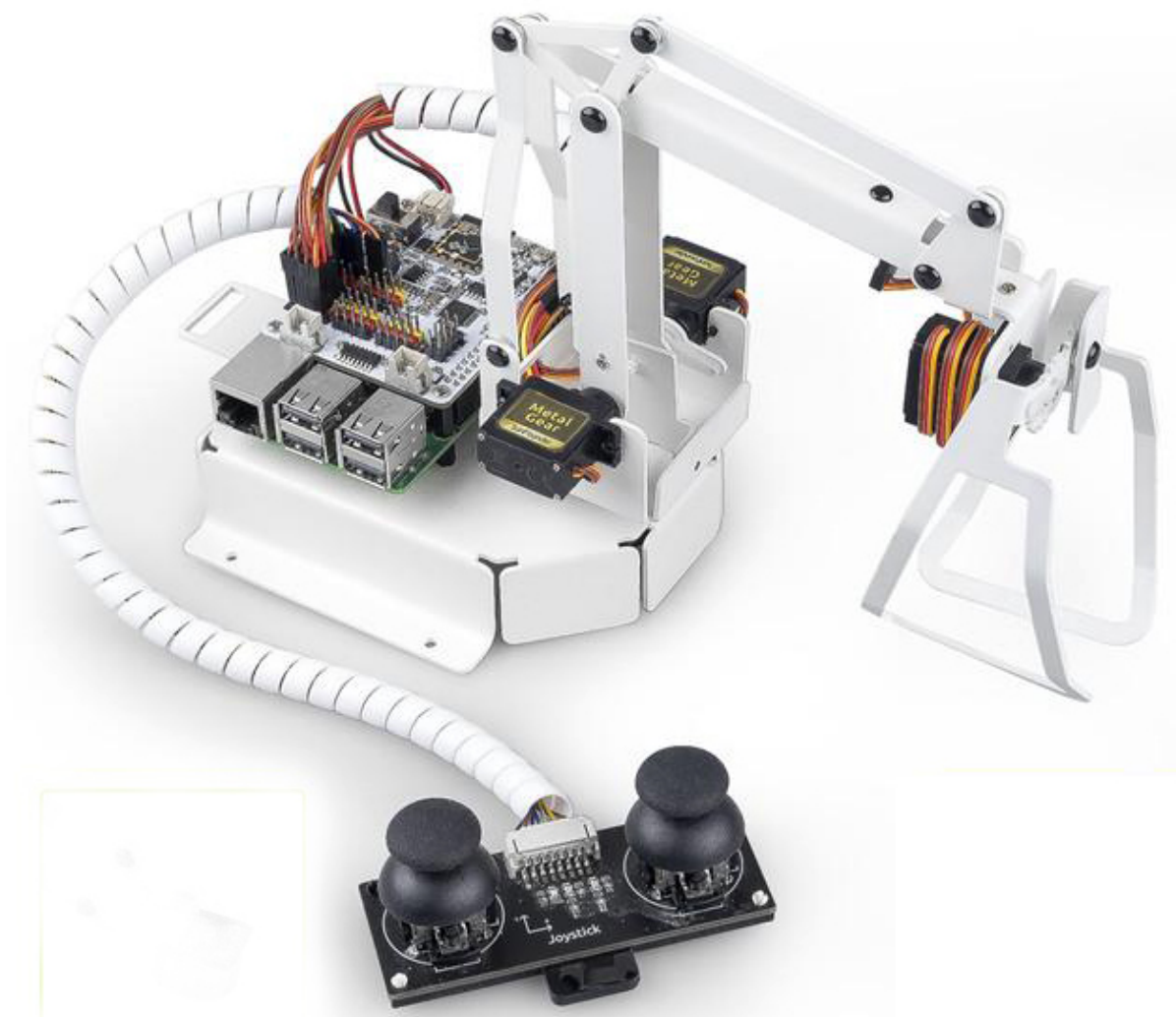
1	元件清单和组装说明	3
2	硬件介绍	5
2.1	手臂	5
2.2	铲斗	8
2.3	竖直夹	12
2.4	电磁铁	16
2.5	双摇杆模块	18
2.6	Robot HAT	21
3	玩转 EzBlock	23
3.1	EzBlock 快速指南	23
3.2	安装并测试三个臂端工具	37
3.3	声音效果	44
3.4	双摇杆控制	49
3.5	远程遥控	60
3.6	坐标模式	76
3.7	记录功能	80
3.8	游戏 - 抓娃娃	83
3.9	游戏 - 收集铁片	92
4	玩转 Python	99
4.1	Python 快速指南	99
4.2	组装和测试 3 个臂端工具	118
4.3	音效	121
4.4	双摇杆模块	123
4.5	键盘控制	129
4.6	坐标模式	138
4.7	记忆功能	140
4.8	游戏 - 抓娃娃	143
4.9	游戏 - 收集铁片	146
5	附录	151
5.1	网页版 EzBlock	151
5.2	apt 和 pip 更换国内源	155
5.3	Filezilla 软件	157
5.4	I2C 配置	159

5.5	远程桌面	161
5.6	关于电池	170
6	疑难解答	173
6.1	安装 EzBlock 操作系统后, 舵机不能转到 0°?	173
6.2	如何在 EzBlock 重新校准机器人	174
6.3	EzBlock 无法连接蓝牙?	175
6.4	APP 搜索到蓝牙, 但无法连接	175
6.5	配置 WIFI 后 APP 无法连接	176
6.6	为什么伺服机有时会无缘无故地返回到中间位置?	176
7	版权声明	177

感谢你选择我们的 PiArm。

PiArm 是用于树莓派的三自由度机械臂。它有 3 个可互换的工具 - 铲斗、竖直夹和电磁铁 - 可帮助你执行不同的任务。

此外，PiArm 提供远程控制和内置双摇杆模块控制。



本教程包括几个部分：器件列表、装配指南、编程和附录。编程部分分为两章：在玩转 EzBlock 和玩转 Python，每一章都允许你让 PiArm 以你想要的方式工作。

- [玩转 EzBlock](#)

如果你是编程新手，请查看本章介绍 EzBlock Studio，这是一种基于块的可视化编程软件，可让你通过简单的拖放块来使 PiArm 移动并实现一些有趣的项目。

- [玩转 Python](#)

如果你更喜欢用更流行的编程语言——python 编程，你可以参考这一节。本章内容从烧录树莓派操作系统开始，到配置树莓派，最后运行代码看效果，即使没有 Python 基础也能快速上手 PiArm。

元件清单和组装说明

你需要先根据清单检查是否有缺少或损坏的组件。如果有任何问题，请联系我们，我们会尽快解决。
请按照 PDF 上的步骤进行组装。

注解：

- 组装前需要购买 2 节 18650 电池并充满电，参见[关于电池](#)。
- Robot HAT 不能给电池充电，需要额外购买充电器。

-
- [PiArm 组装教程 \(.pdf\)](#)

警告： 如果你收到的套件包含了透明的 Robot HAT 保护壳，请不要把它安装上去，以免影响 PiArm 的左右转动范围。

2.1 手臂

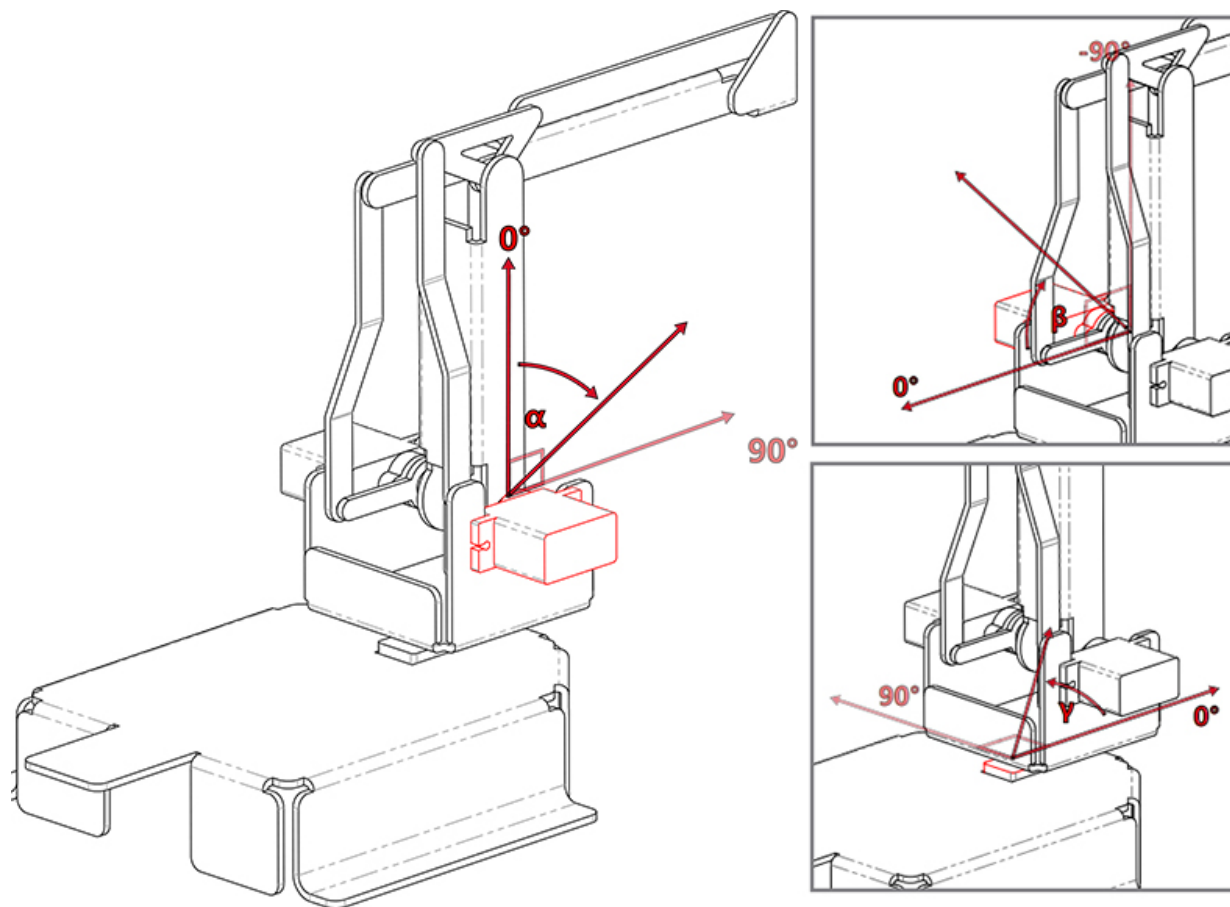
PiArm 的手臂有两种方法可以控制: 角度模式和坐标模式。

- 角度模式: 向手臂上的三个舵机写入一定的角度, 从而使手臂转动到特定的位置。
- 坐标模式: 为手臂建立空间直角坐标系并设置控制点, 写入控制点的空间坐标, 从而让手臂达到特定的位置。

2.1.1 角度模式

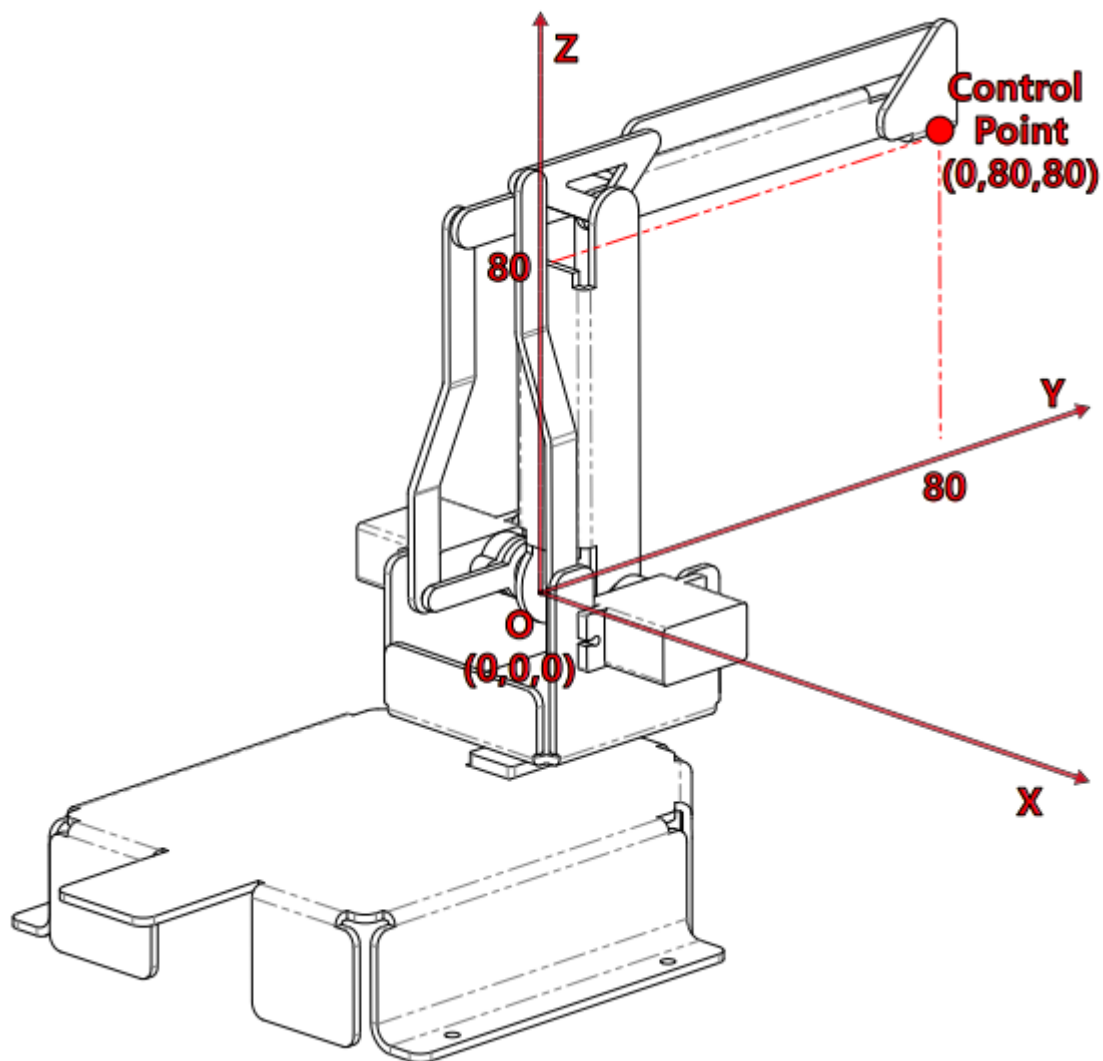
手臂上有三个舵机控制它的上下、左右、前后, 我们用 α 、 β 和 γ 来表示它们的旋转角度, 如下图所示。

- α (alpha): 代表手臂的前后旋转角度, 由于结构的限制, 建议旋转范围为: $-30 \sim 60$ 。
- β (beta): 代表手臂的上下旋转角度, 由于结构的限制, 建议的旋转范围是: $-60 \sim 30$ 。
- γ (gamma): 代表手臂的左右旋转角度, 范围是: $-90 \sim 90$ 。



2.1.2 坐标模式

PiArm 有一个空间矩形坐标系，其原点位于两侧舵机输出轴的中心点。控制点 (Control Point) 位于手臂的顶部，刻度单位为毫米。在初始状态下，控制点 (Control Point) 的坐标为 $(0, 80, 80)$ 。



需要注意的是，PiArm 的臂长是有限的，如果坐标值的设置超过了其机械运动的极限，PiArm 将旋转到一个不可预测的位置。

换句话说，PiArm 的总臂长是 160 毫米，这意味着沿 Y 轴移动的控制点的极限值应该在 (0,0,0) 到 (0,160,0) 之间。但是，由于结构本身的限制，活动范围应该比这个范围小得多。

- X 坐标的推荐范围是 -80 ~ 80。
- Y 坐标的推荐范围是 30 ~ 130。
- Z 坐标的建议范围是 0 ~ 80。

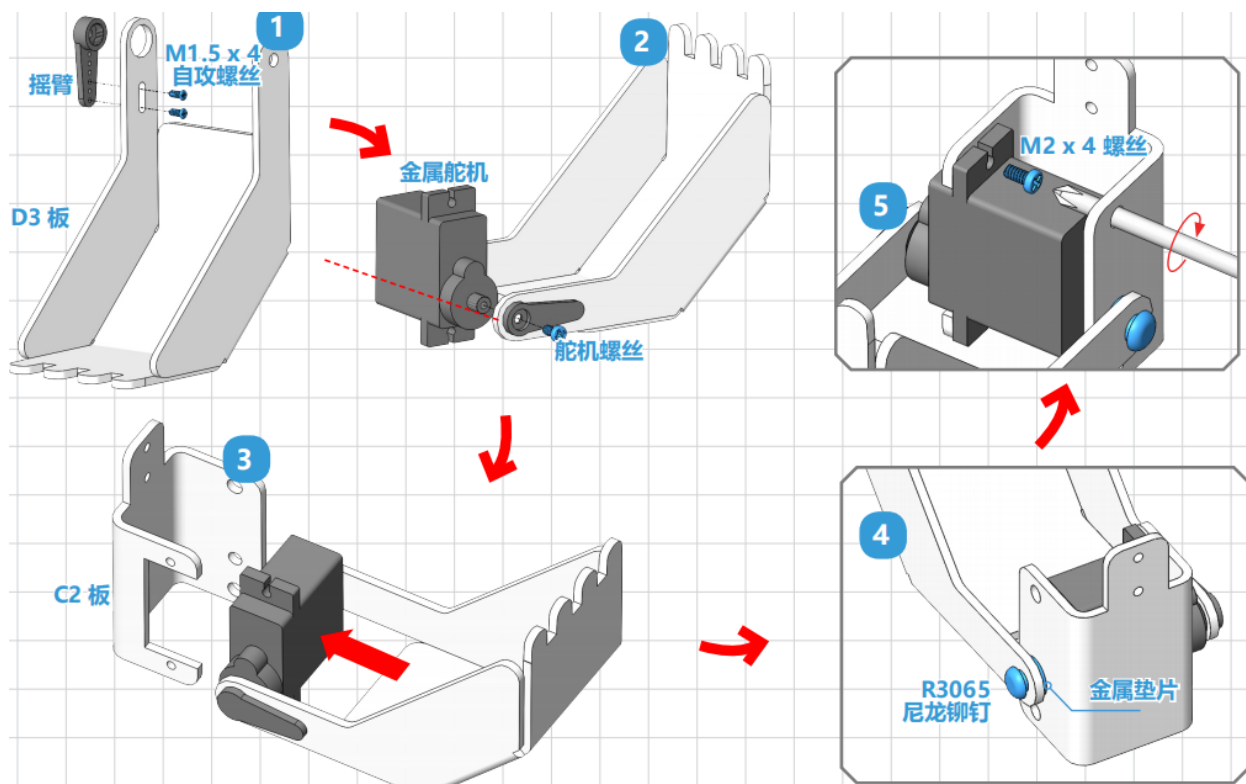
2.2 铲斗



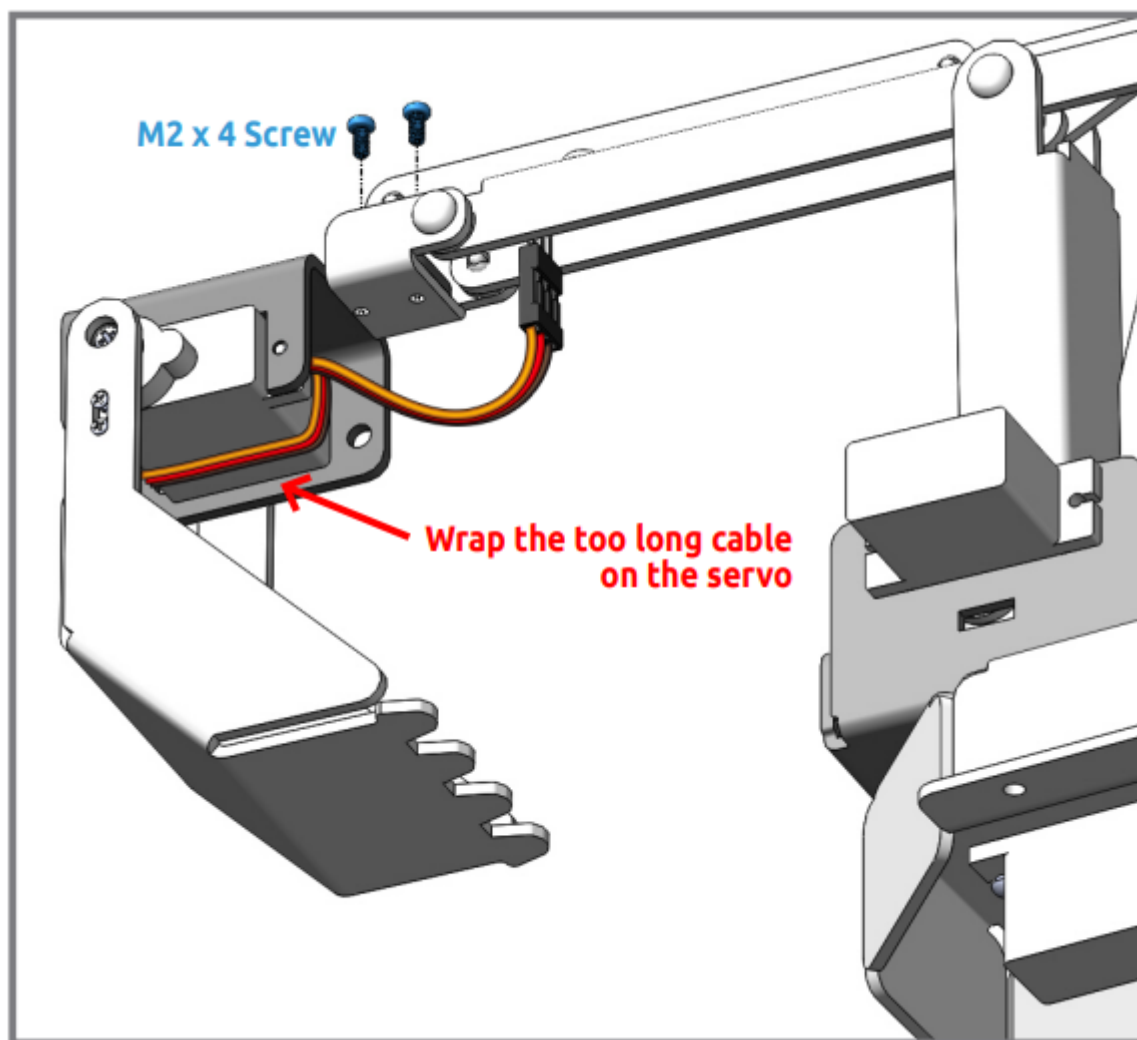
组装铲斗

按照下图组装铲斗。

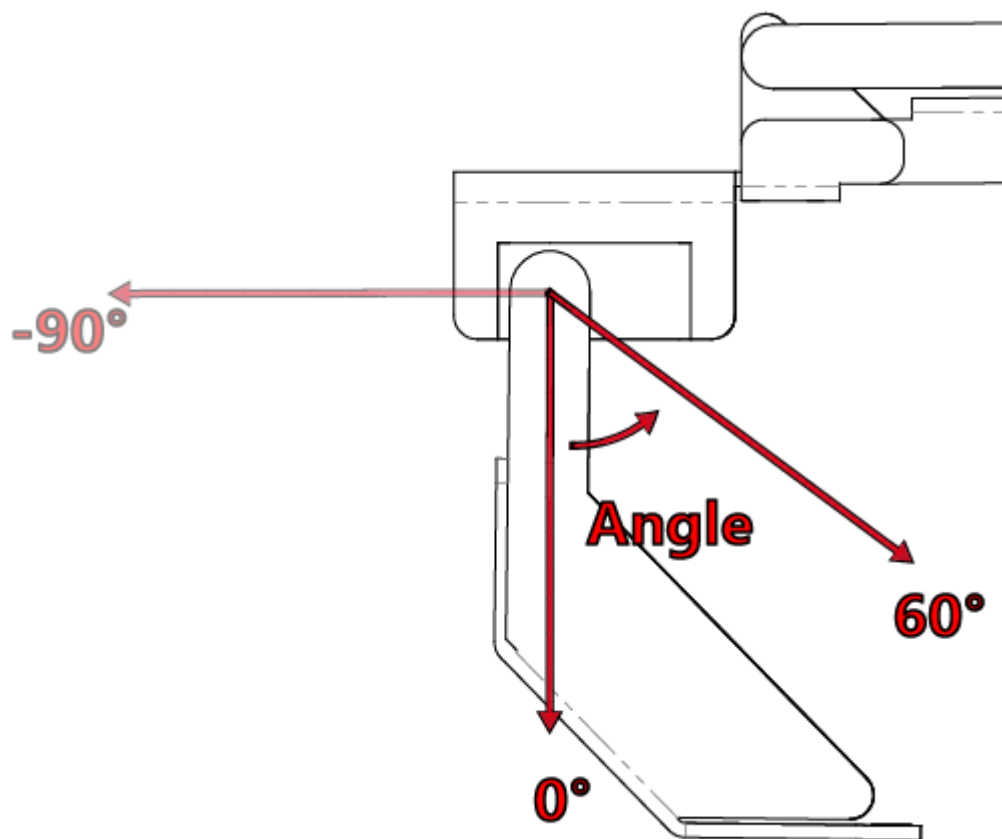
注解： 注意在第 2 步时需要先将舵机插入到 P11 进行调零后，再将 D3 板以垂直方向插入到舵机轴。



用 M2x4 螺丝将铲斗组装到 PiArm 的末端。



铲斗的旋转范围为-90 ~ 60。



使用范围

不能挖水，可以用来挖沙土，挖石子。

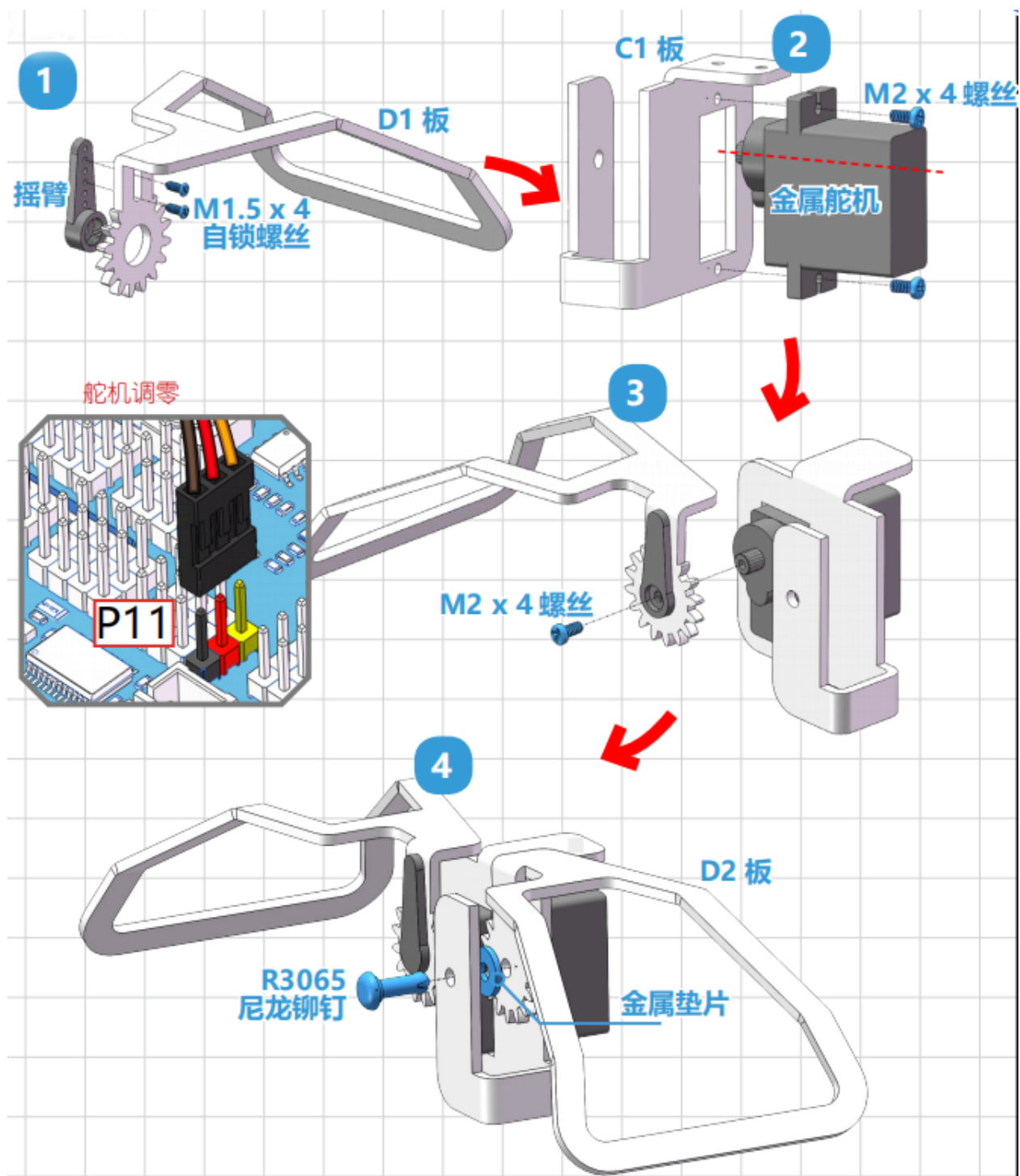
2.3 竖直夹



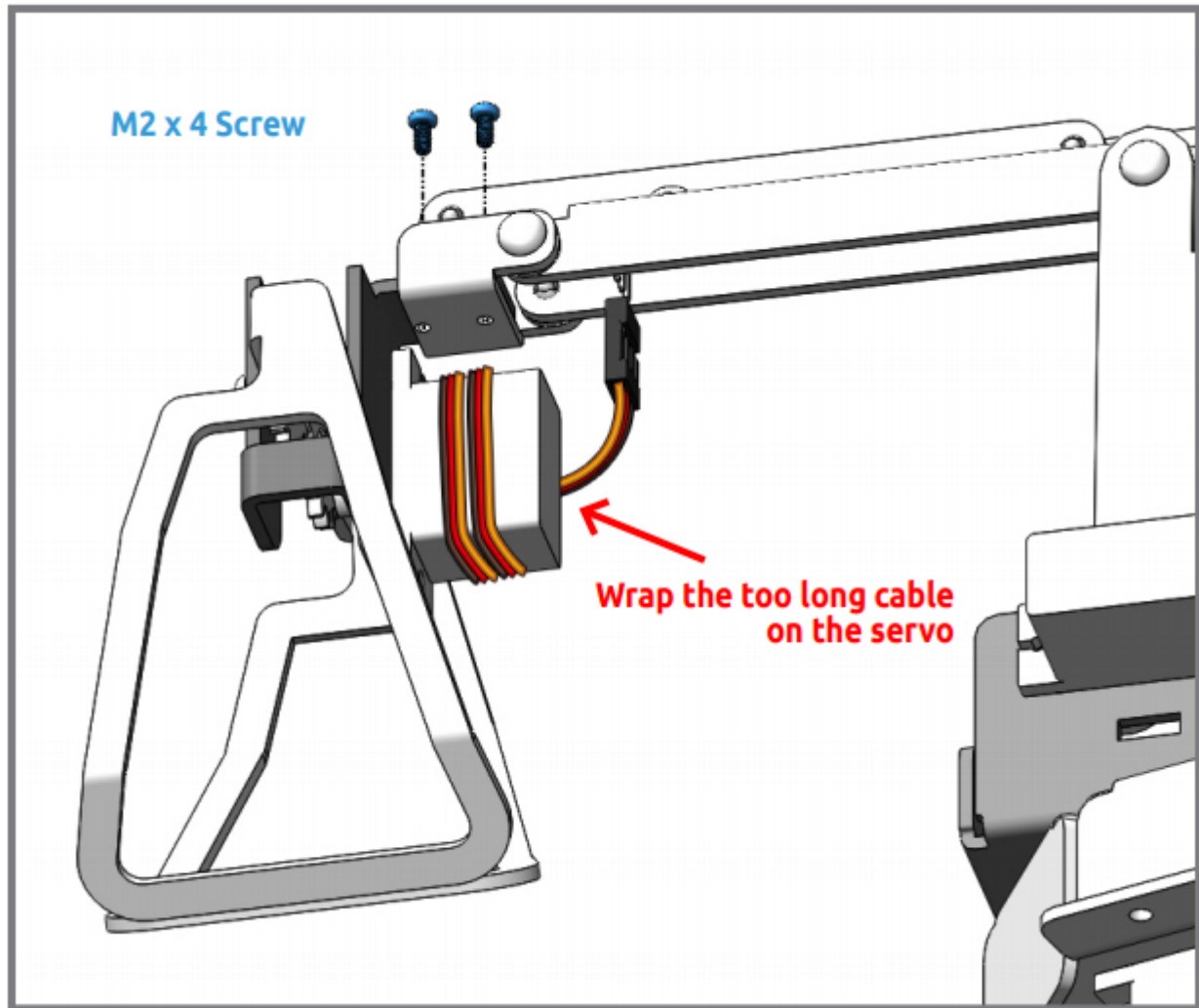
组装

按照下图组装竖直夹。

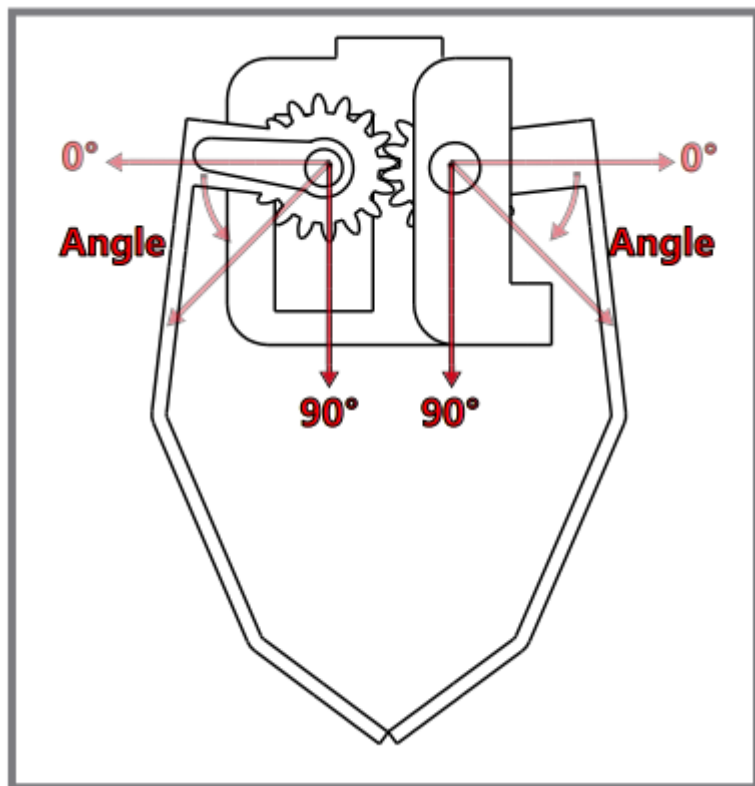
注解： 注意在第 3 步时需要先将舵机插入到 P11 进行调零后，再将 D1 板以垂直方向插入到舵机轴。



将竖直夹用 M2x4 的螺丝固定到 PiArm 的末端。



竖直夹的角度范围是 0-90°。



使用范围

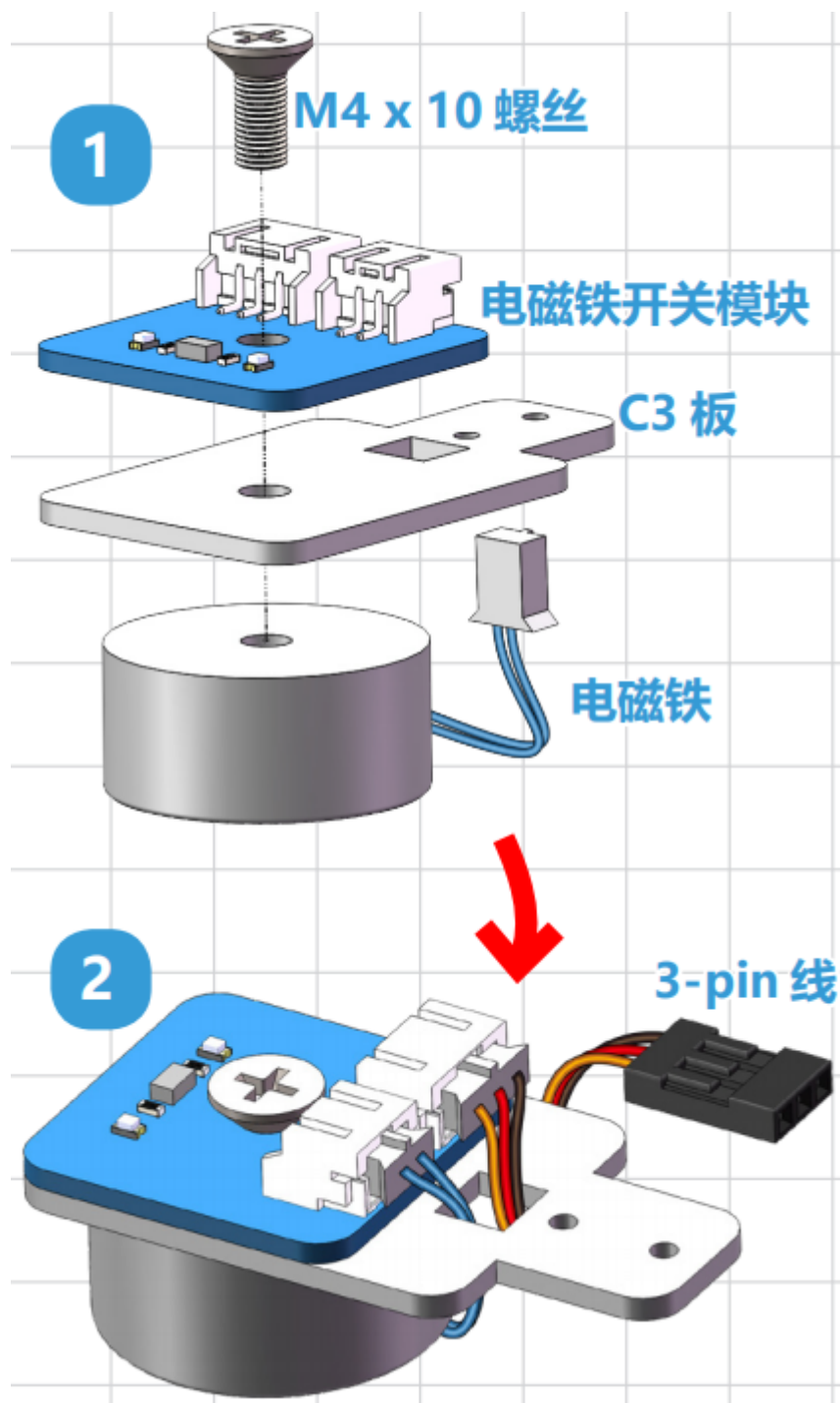
- 夹的物体重量需低于 150g。
- 建议物体的可夹高度需低于 4cm，宽度低于 8.5cm。
- 细长的物体需要找对角度才能夹起。

2.4 电磁铁

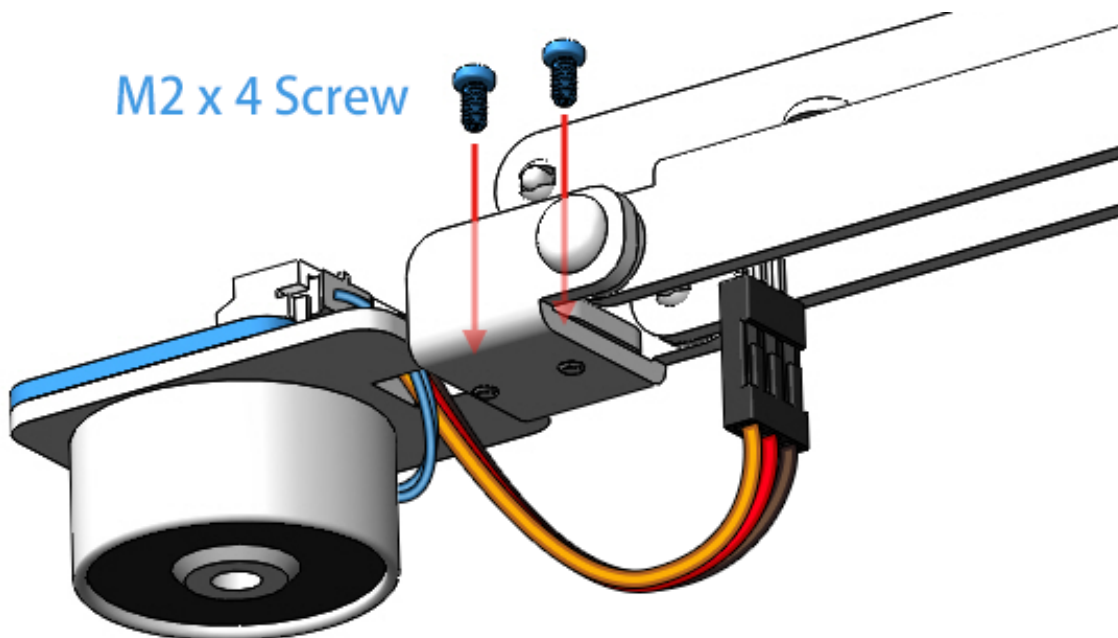


组装

按照下图将电磁铁模块组装起来。



然后用 M2x4 的螺丝将电磁铁固定到 PiArm 的末端。

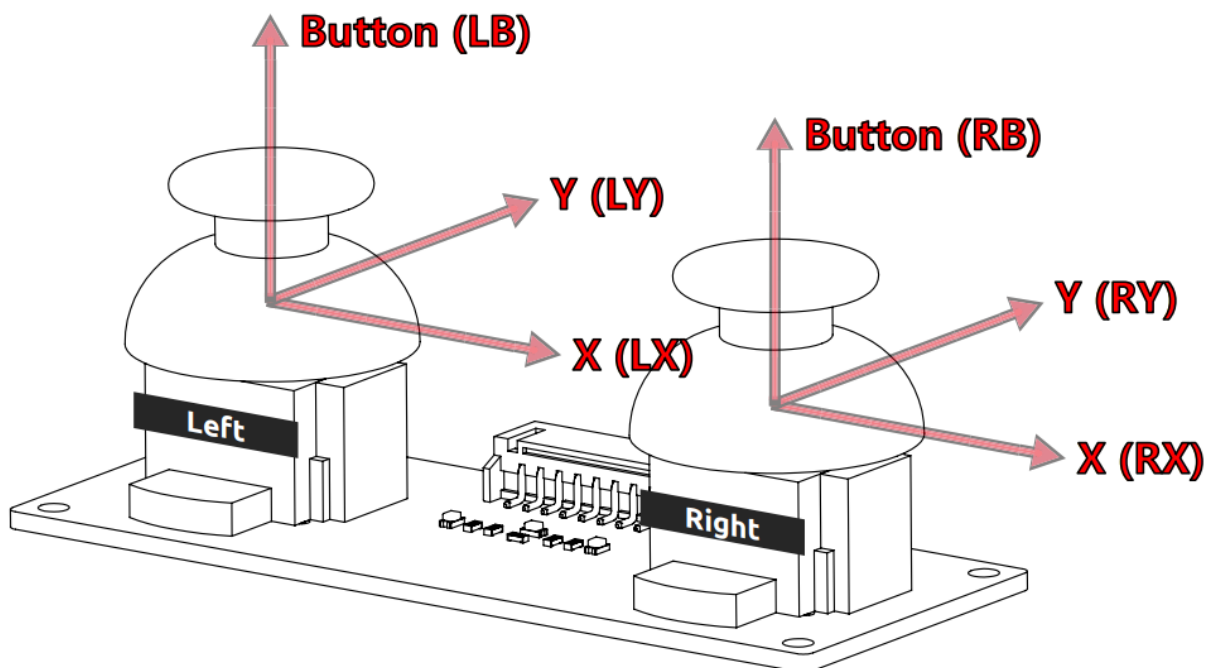


使用范围

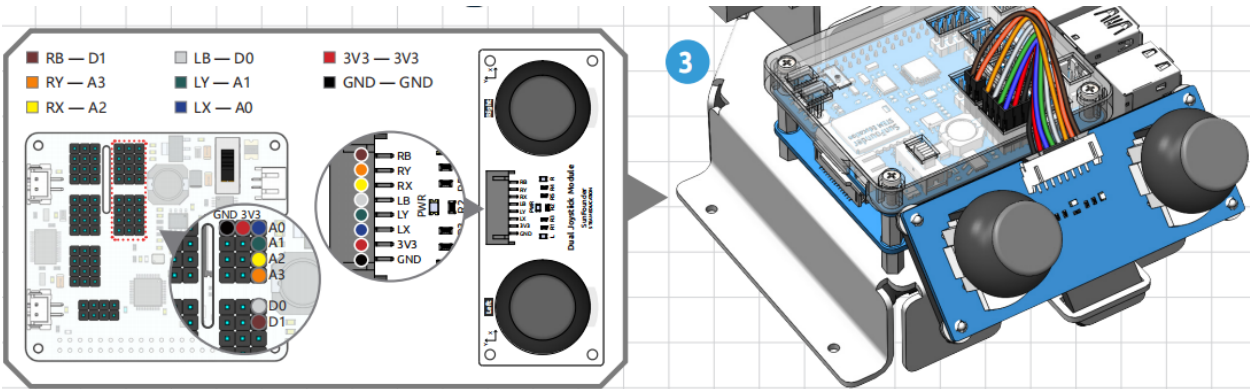
- 只能用来吸铁制品。
- 铁制品的表面积越大，吸附性越强。
- 建议铁制品的重量低于 150g。

2.5 双摇杆模块

双摇杆模块，顾名思义，由两个摇杆组成，每个摇杆可以在 X、Y 和 Z 方向输出电信号。

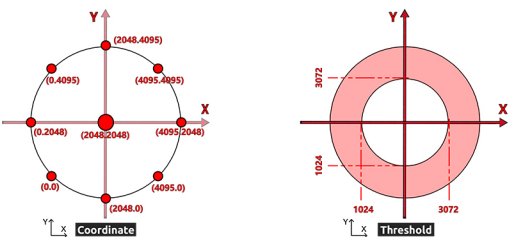


在使用双摇杆模块之前，你需要把它的 8 根线连接到 Robot HAT 的相应引脚上，如下图所示。



摇杆读数范围在 0~4095 的平面坐标系中，原点 (0,0) 在左下角。

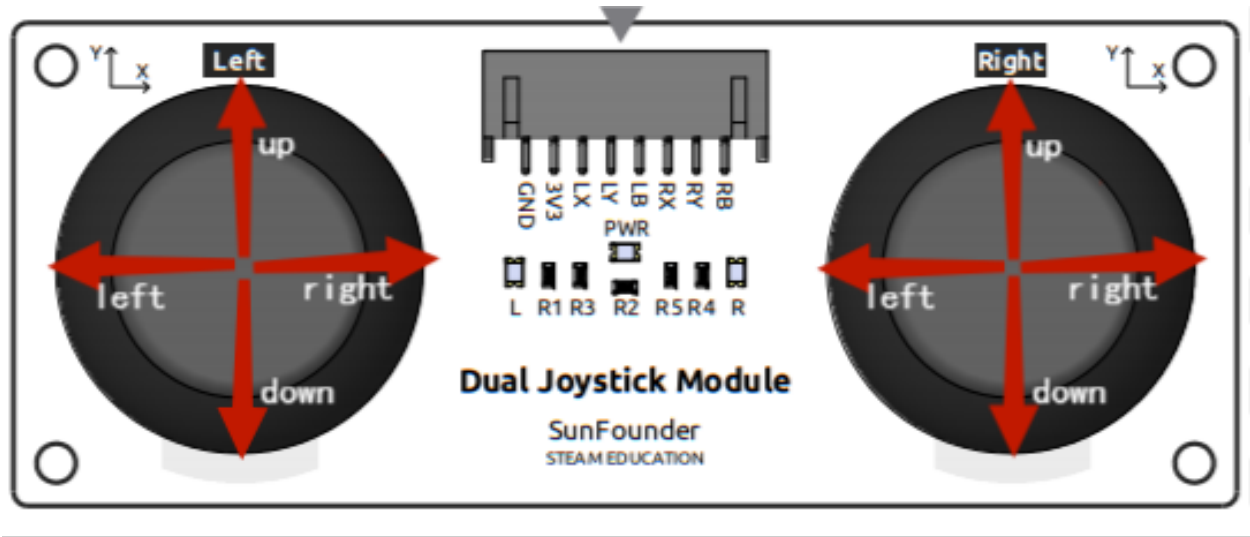
举个例子，不推动摇杆时的坐标值为 (2048,2048)。如果将摇杆向左推动，坐标为 (0,2048)。当向下推动摇杆时，坐标为 (2048,0)，如下所示。



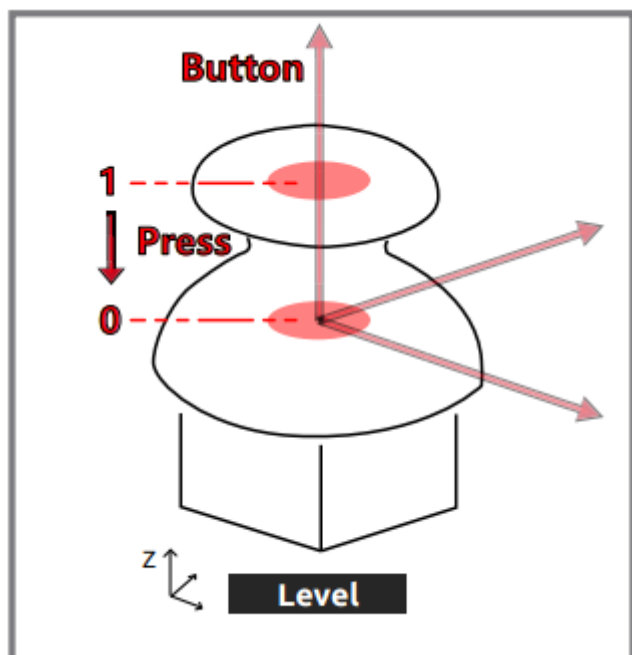
但是电信号容易波动，很难得到绝对稳定的读数，所以我们通常会设置一个数值区间来判断摇杆目前所处位置。

建议边界值设为 3072 和 1024。当摇杆的读数大于 3072 时，认为摇杆是向上（或向右）推动；如果读数小于 1024，则认为摇杆杆向下（或向左）推动。

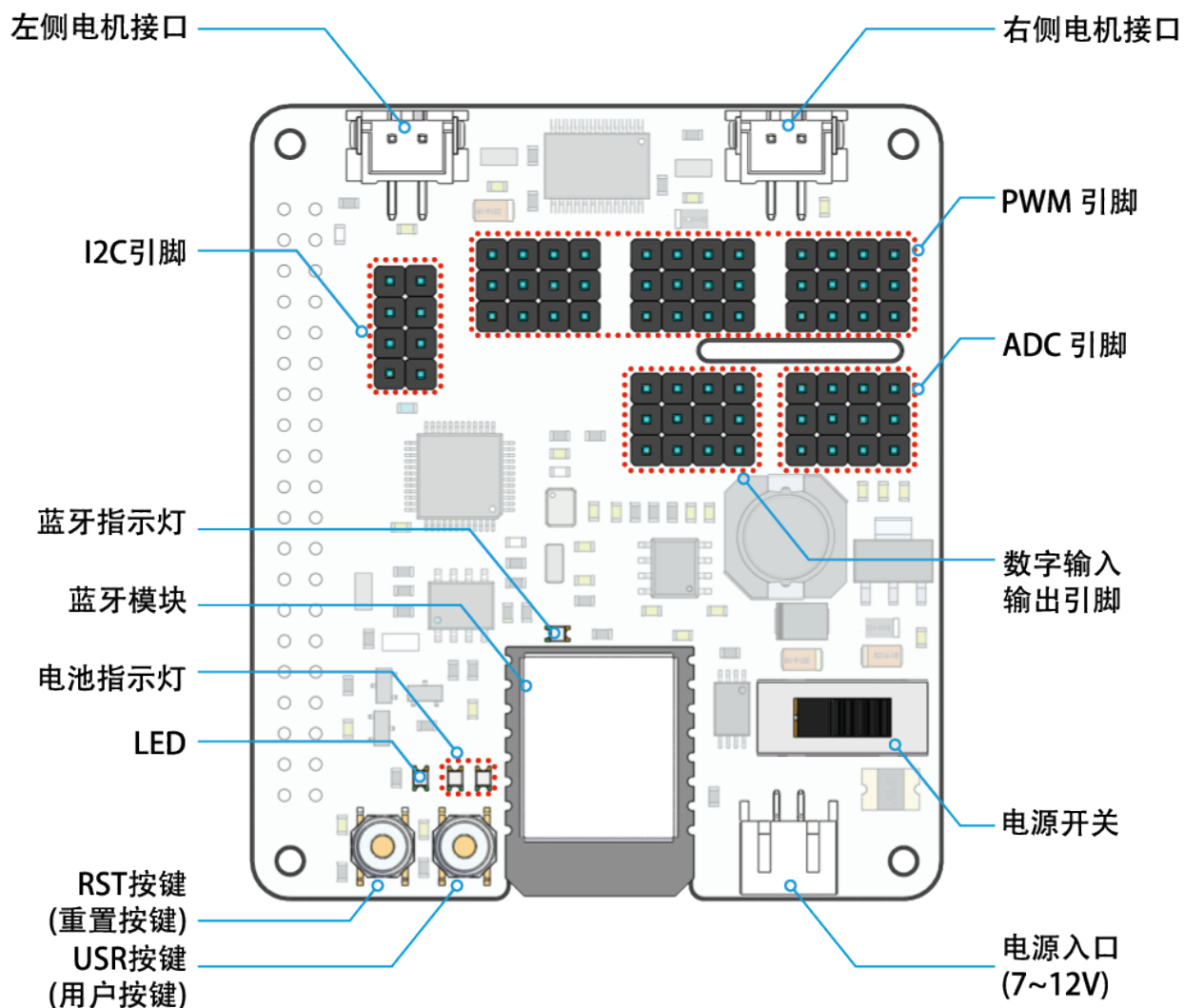
注解：在用 Python 库中，已经将这些值处理成方向指示，如下图所示：



Z 轴按钮按下时输出低电平 (0)，松开时输出高电平 (1)。



2.6 Robot HAT



RST 按钮

- 短按 RST 按钮将导致所有正在运行的程序重置。
- 长按 RST 按钮直到 LED 亮起，然后松开将断开 Robot HAT 的蓝牙芯片。

USR 按钮

- USR 按钮的功能可通过编程进行配置。(按下导致输入 0，松开产生 1 输入)

LED

- 通过编程配置 (输出 1 打开 LED，输出 0 关闭 LED。)

电池指示灯

- 电池电压高于 7.8V 将点亮两个 LED 指示灯。电池电压在 6.7V 到 7.8V 之间只会点亮一个 LED，低于 6.7V 的电压将关闭两个 LED。

蓝牙指示灯

- 蓝牙指示灯 LED 将在蓝牙连接稳定时保持亮起，并在信号传输期间快速闪烁。如果蓝牙断开连接，LED 将每隔 1 秒闪烁一次。

EzBlock 是 SunFounder 所使用的一个软件开发平台，用于帮助初学者和新手在树莓派平台上开始学习编程。EzBlock 有两种编程语言环境：一种是图形用户界面环境，一种是命令行 Python 环境。

EzBlock 几乎可用于所有类型的设备，包括 Mac、PC 和 Android。在使用 EzBlock 之前，请先按照以下教程来完成 EzBlock 下载，安装和使用。

3.1 EzBlock 快速指南

这里是使用 EzBlock 的一些使用指南。在组装的时候，你只需要完成安装 *EzBlock* 镜像和舵机调零。

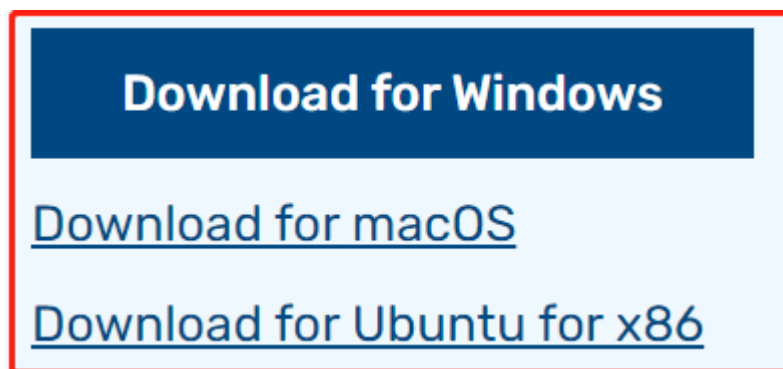
组装完成后再去下载 *EzBlock Studio*，完成快速配置后，就可以根据教程内容对你的机器人进行编程了。

建议按顺序阅读课程内容。

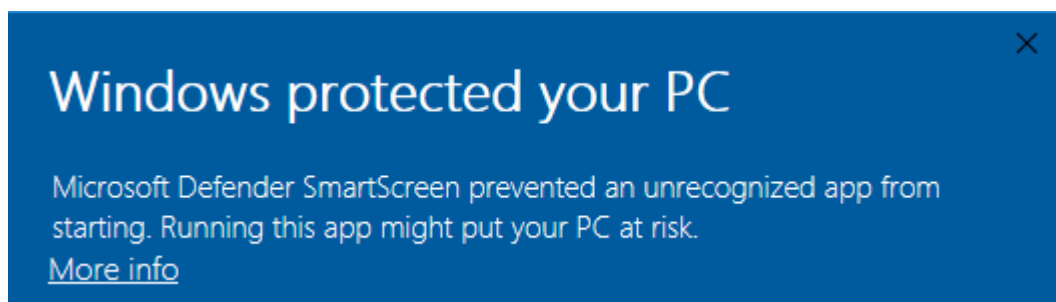
3.1.1 安装 EzBlock 镜像

在组装机器人之前，按照下面的步骤给树莓派的 Micro SD 卡烧录 EzBlock 系统。

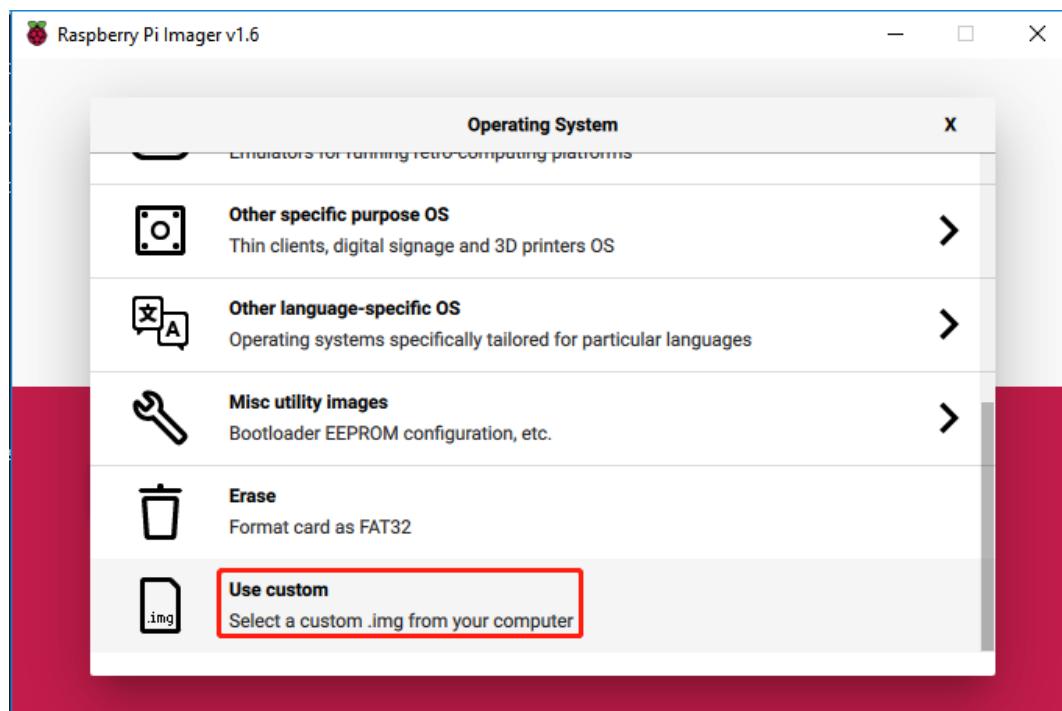
1. 树莓派开发了一个图形 SD 卡写入工具，适用于 Mac OS、Ubuntu 18.04 和 Windows，对于大多数用户来说是最简单的选择，因为它会下载映像并将其自动安装到 SD 卡。访问下载页面：<https://www.raspberrypi.org/software/>。单击与您的操作系统匹配的 Raspberry Pi Imager 链接，下载完成后，单击它以启动安装程序。



2. 当您启动安装程序时，您的操作系统可能会尝试阻止您运行它。例如，在 Windows 上，我收到以下消息：如果出现此消息，请点击 **更多信息**，然后点击 **仍然运行**，然后按照说明安装 Raspberry Pi Imager。

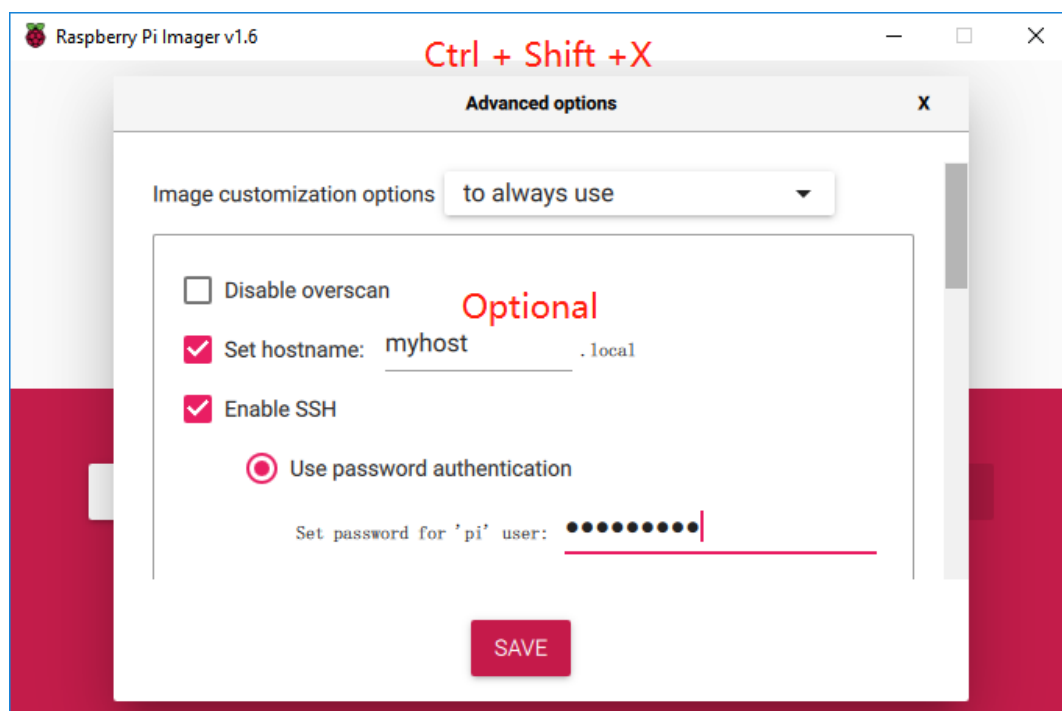


3. 将 SD 卡用读卡器插入计算机或笔记本电脑的 SD 卡插槽。
4. 下载 EzBlock 镜像
- 天翼网盘：链接：<https://cloud.189.cn/t/QNrEJnEjYNRz>
 - 百度网盘：链接：https://pan.baidu.com/s/1ku1VoukCebChq9-OzkHf_g?pwd=ezbl，提取码：ezbl。
 - 由于文件超过 1G，需要在电脑上下载客户端之后才能下载文件。
5. 在 Raspberry Pi Imager 中选择刚下载的 EzBlock 镜像。



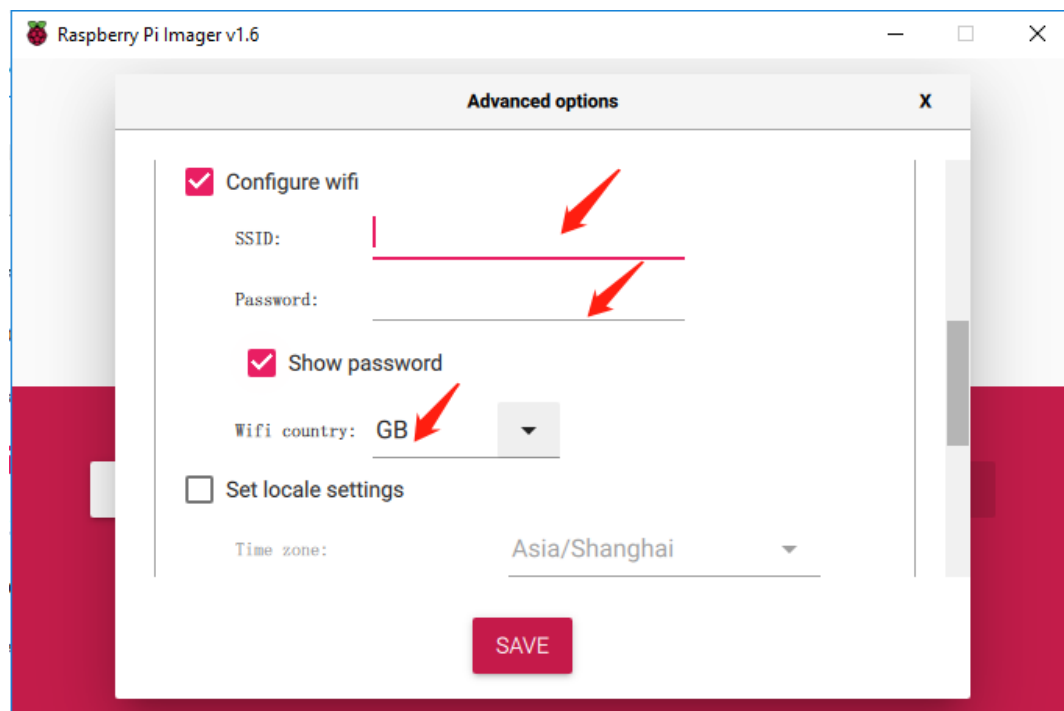
6. 按下 **Ctrl+Shift+X** 或者点击 设置按钮来打开 **Advanced options** 页面来设置 hostname 和启动 SSH。你可以选择 “always use this image customization options”（始终使用该定制选项）。

注解： hostname 是让你在使用网页版 *EzBlock*，可以用它来连接到你的产品，你也可以不设置。

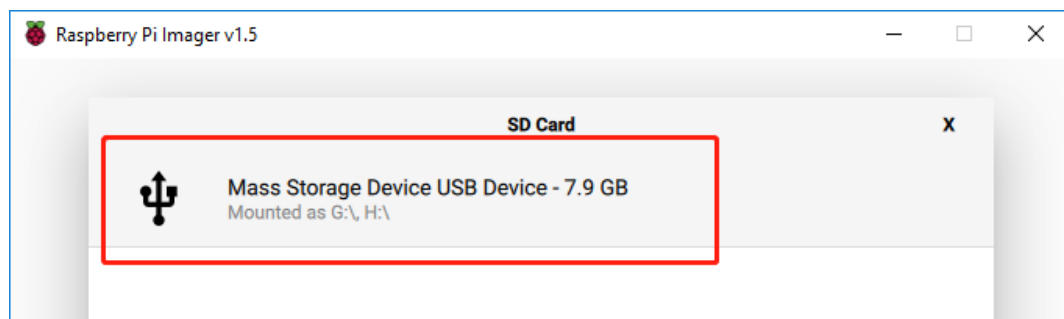


7. 下拉到底完成 WiFi 配置，然后点击 **SAVE**。

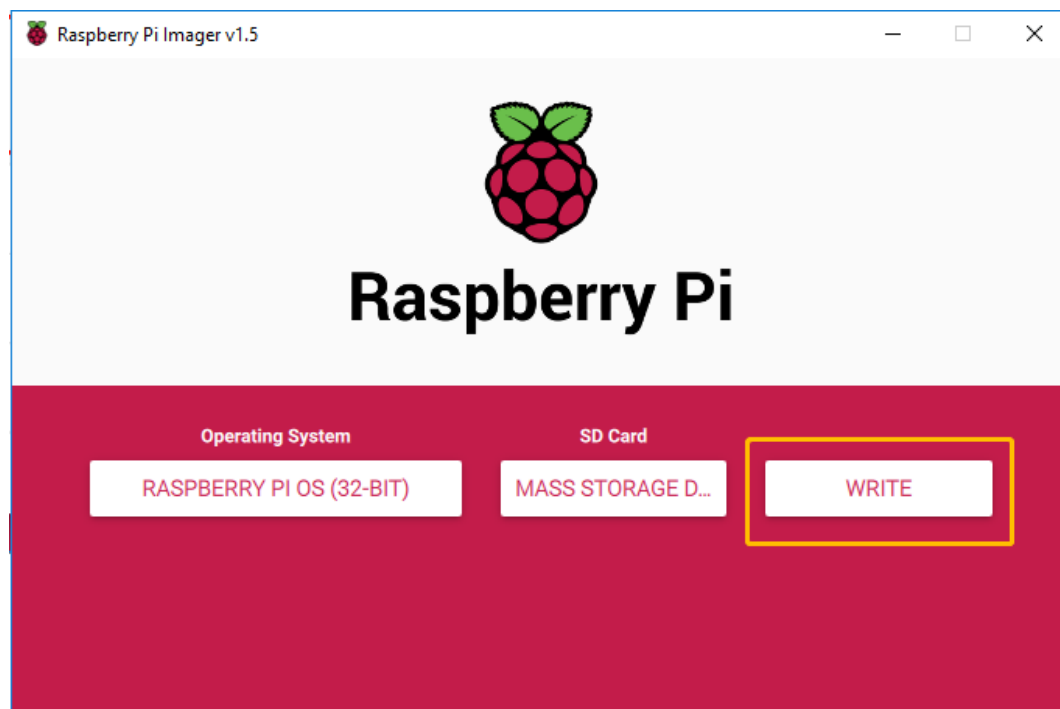
注解: wifi country 选择 CN



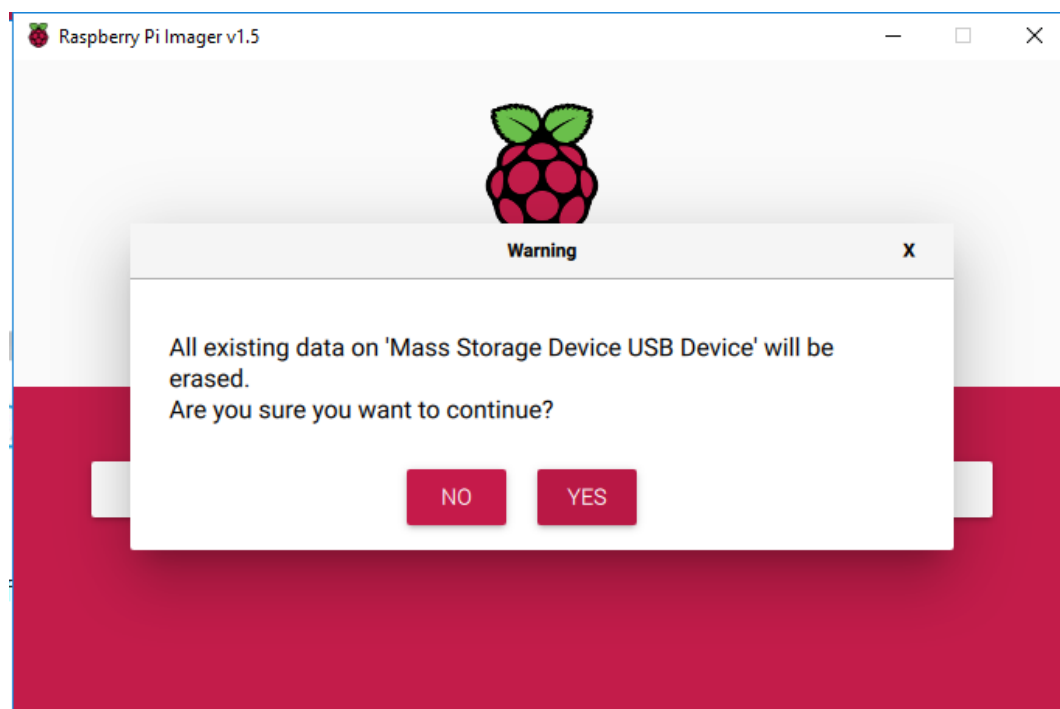
8. 选择您正在使用的 SD 卡。



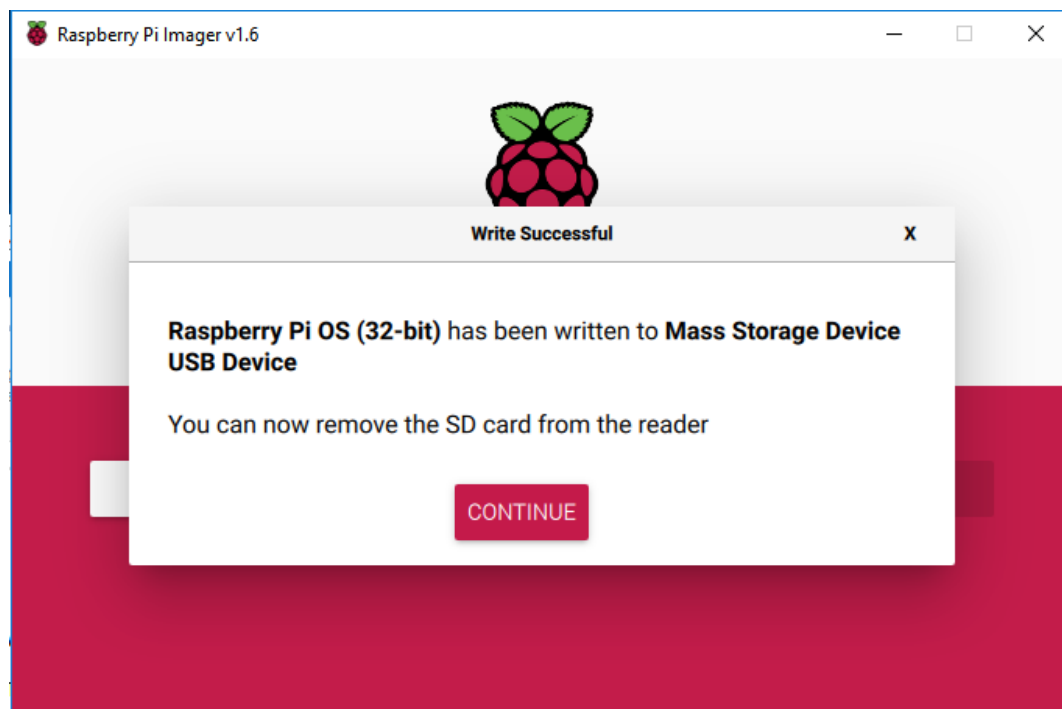
9. 单击 **WRITE** 按钮。



10. 如果您的 SD 卡上当前有任何文件，您可能希望先备份这些文件以防止永久丢失它们。如果没有要备份的文件，请单击 **YES**。

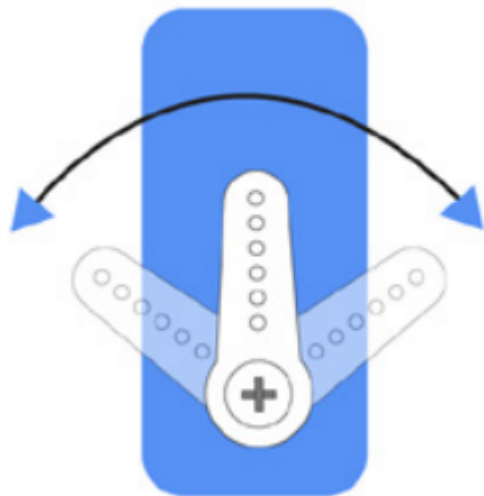


11. 等待一段时间后，会出现如下窗口，代表写入完成。现在你就可以将读卡器从电脑中拔出，并取出 SD 卡插入到树莓派上。

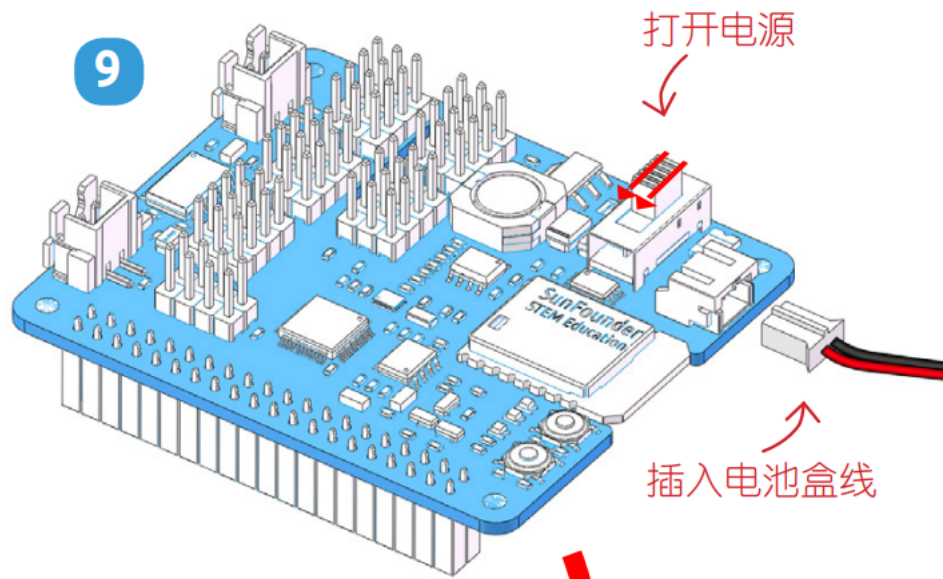


3.1.2 舵机调零

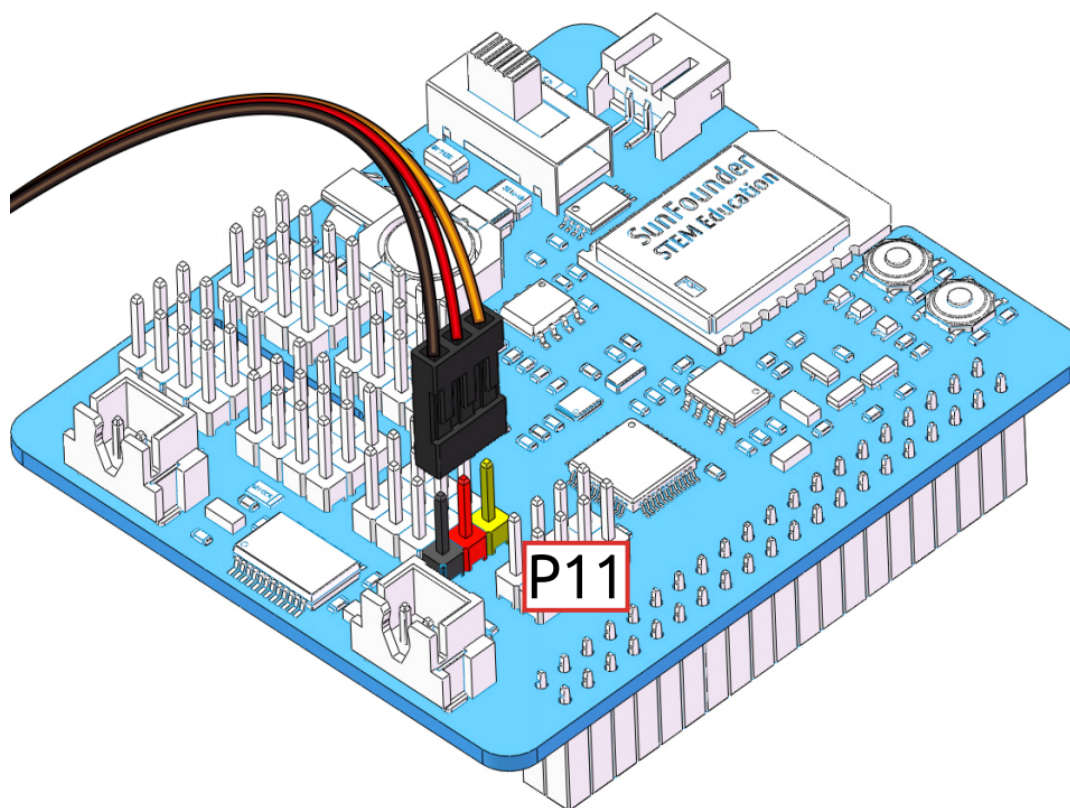
1. 为确保舵机已正确设置为 0° ，首先将摇臂轻轻插入舵机轴，然后将摇臂轻轻旋转到不同的角度。



2. 按照组装折页上的提示，插入电池盒线，将电源开关拨向 ON 的位置。等待 1-2 分钟，会有声音提示树莓派启动成功。



3. 接下来，将舵机线插入 P11 端口，如下所示：



4. 此时你会看到舵机臂转动到特定的位置 (0°)。如果伺服臂没有返回到 0° ，请按 **RST** 按钮重新启动 Robot HAT。
5. 现在你就可以按照组装折页上的指示继续安装。

注解：

- 在用舵机螺丝固定该舵机前，不要拔出该舵机线，可在固定完之后拔出。
 - 不要在上电情况下随意转动舵机以免损坏；如果舵机轴插入的角度不对，需把舵机拔出再重新插入。
 - 在组装每个舵机之前，都需要将舵机电缆插入 P11 并打开电源以将它的角度设置为 0°。
 - 若后面用 EzBlock APP 给机器人下载程序后，此调零功能将失效。
-

3.1.3 下载 EzBlock Studio

扫描下方二维码，下载 EzBlock Studio APP。



注解：

- 对于安卓手机，会提示用浏览器打开。下载成功后，根据提示允许浏览器安装该应用
 - 对于 iPhone 手机，会打开 APP Store 来安装该应用。
 - 对于笔记本和电脑可以使用网页版 [EzBlock](#)。
-

3.1.4 快速配置

在你第一次用 EzBlock 对你的机器人进行编程时，会有一个快速配置的过程。

1. 打开 EzBlock Studio，会弹出一个空设备列表的窗口。您需要在打开产品电源的同时打开移动设备的蓝牙，才会出现产品编号。

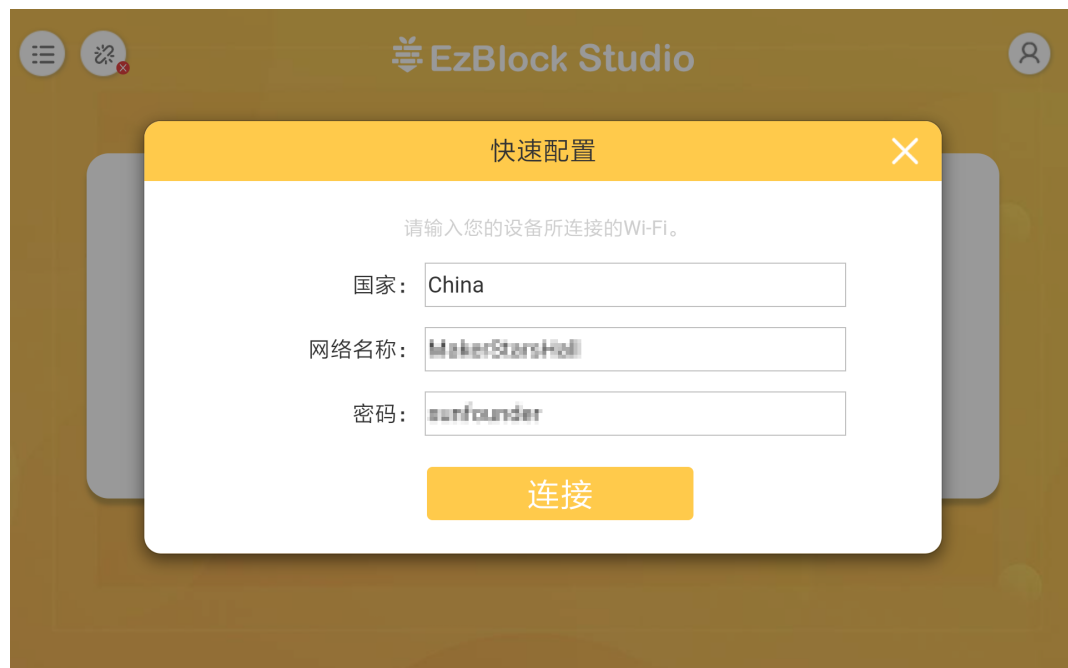


2. 点击右上角的 **完成**，稍等片刻，会提示 **连接成功**。此时您需要点击 **确定**来快速配置您的产品。



3. 输入您的 Wi-Fi 帐户和密码。

注解：如果您已经提前配置了 Wi-Fi，那么 **快速配置**这一步就不会出现，直接进入下一步 **设置名称**。



4. 选择与您匹配的产品。



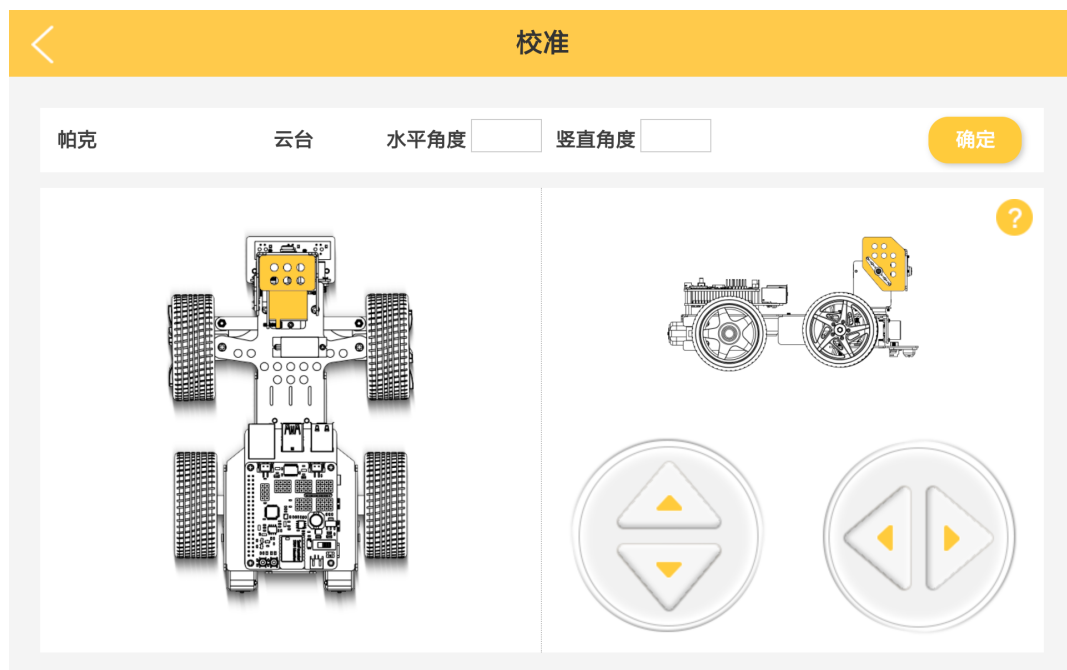
5. 为您的产品输入名称。



6. 如果您的产品需要校准，会有提示告诉您可以通过点击 **立即校准** 进入校准页面。如果不需要，则弹出窗口消失并返回主页。



7. 每个产品的校准页面都不一样，但是都有提示需要校准的部分。您可以点击相应的部分，然后参考 **校准帮助** 进行校准。校准完成后，点击 **确认**。



3.1.5 打开或新建项目

配置完成之后，你就可以开始给你的机器人编写程序，我们已经将所有示例都已上传到 **示例** 页面，你可以直接运行或打开相应的项目。或者根据课程中的提示，新建项目后，通过简单的拖拽来完成相应项目的编程。

打开项目

1. 在首页，点击 **示例**，进入示例页面。如果您只需要简单地测试这些示例，您只需点击 **运行** 即可让您的产品工作。如果要查看和修改里面的代码，那么就需要点击 **编辑** 进入编程页面。



2. 进入编程页面之后，将代码修改之后，点击 **下载/运行** 按钮，就能看到你的机器人做出相应的效果。可点击 **运行/暂停** 按钮来停止运行。



新建项目

1. 点击 新建项目按钮。



2. 填写项目名称。



3. 根据每个项目的提示，编写程序。



4. 点击 **下载/运行** 按钮，就能看到你的机器人做出相应的效果。可点击 **运行/暂停** 按钮来停止运行。



EzBlock 项目

在这里，我们向您展示在 EzBlock Studio 上玩 PiArm 的项目。如果你是新手，可以参考各个项目里面的代码图片进行编程，可以根据 **小提示**来学习积木块的使用。

如果你不想一个一个的写这些项目，我们已经把它们上传到 EzBlock Studio 的示例页面，你可以直接运行它们，也可以在运行它们之前自定义编辑。

3.2 安装并测试三个臂端工具

这是第一个项目，将教会您如何组装和使用 PiArm 的三个臂端工具。

3.2.1 基本代码块提示

- 程序的基本结构由下面两个代码块构成，[开始] 代码块用于做一些初始化（里面没有放置代码块也不能删除）而 [循环] 顾名思义是一个连续的循环从而保证程序一直响应。



- 下面这个代码块用于设置程序的间隔时间，单位为毫秒。



3.2.2 PiArm 代码块提示

在这里可以找到一些 PiArm 专用的代码块。



3.2.3 铲斗

1. 组装铲斗。
2. 开始编写代码来使铲斗动起来。

把 [初始化铲斗引脚为 P3] 代码块放入 [开始] 块中从而将铲斗的引脚初始化为 P3。

注解： 因为在组装图中，臂端工具的舵机是连接的 P3 引脚，当然你也可以自己接到其它的备用引脚。



3. 每隔一秒钟将铲斗从-60 转动到 90°，再往复。
 - [设置铲斗回卷至角度 ()]: 用于设置铲斗角度，舵机本身的旋转角度是-90 ~ 90，但由于结构限制，最终的角度范围是-90 ~ 60。
 - [延时 ()]: 从 基本类别中可以找到, 用于设置两个代码块之间的运行间隔时间，单位为毫秒。



4. 搭建完代码之后，单击右下角的 **下载** 按钮将其下载到 PiArm 中。然后你就可以看见铲斗在来回移动，可以单击运行按钮来停止运行代码。

注解：如果你发现舵机为 0 时，铲斗不是垂直向下的，需要松开舵机螺钉，再重新将铲斗以垂直向下的角度安装一遍。



3.2.4 竖直夹

1. 组装竖直夹。
2. 然后搭建代码块来让竖直夹动起来。

将 [初始化竖直夹引脚为 P3] 代码块放到 [开始] 块中使竖直夹的引脚初始化为 P3。

注解： 因为在组装图中，臂端工具的舵机是连接的 P3 引脚，当然你也可以自己接到其它的备用引脚。



3. 每隔一秒让竖直夹在 0° 到 90° 的范围里来回移动。
 - [设置数值夹合拢至角度 (0)]: 用于设置竖直夹的角度，范围 0-90 度。
 - [延时 (0)]: 从 基本类别中可以找到, 用于设置两个代码块之间的运行间隔时间，单位为毫秒。



4. 搭建完代码之后，单击右下角的 **下载** 按钮将其下载到 PiArm 中。然后你就可以看见竖直夹在不断张合，可以单击运行按钮来停止运行代码。

注解： 如果你发现竖直夹在 90° 时，不是垂直向下并合拢。需要将竖直夹的固定螺钉取下，重新再安装一遍。



3.2.5 电磁铁

1. 组装电磁铁。
2. 然后搭建代码块让电磁铁工作。

将 [初始化电磁铁引脚为 P3] 代码块放入 [开始] 块中使电磁铁的引脚初始化为 P3。

注解：因为在组装图中，臂端工具的舵机是连接的 P3 引脚，当然你也可以自己接到其它的备用引脚。



3. 每隔一秒让电磁铁反复的通电和断电。
 - [电磁铁 (启动/关闭)]: 用于让电磁铁通电和断电。
 - [延时 ()]: 从 基本类别中可以找到, 用于设置两个代码块之间的运行间隔时间, 单位为毫秒。



4. 搭建完代码之后，单击右下角的 **下载** 按钮将其下载到 PiArm 中。然后你会发现电磁铁每秒通电一次，电磁铁上的 LED（D2）亮起，表示它已通电，此时可以用来吸附一些铁制品。



3.3 声音效果

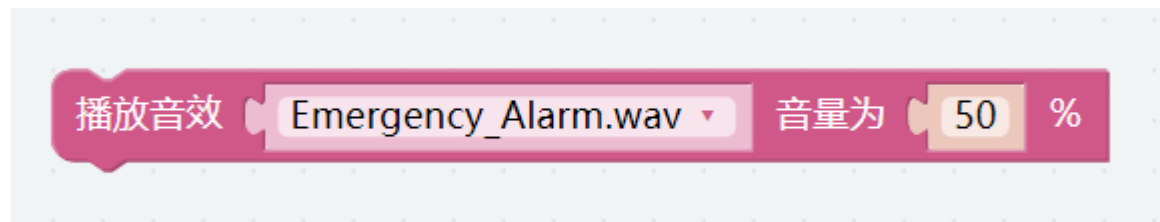
Robot HAT 内置扬声器，可以播放一些音乐和音效，也可以实现 TTS 语音功能。

3.3.1 代码块小提示

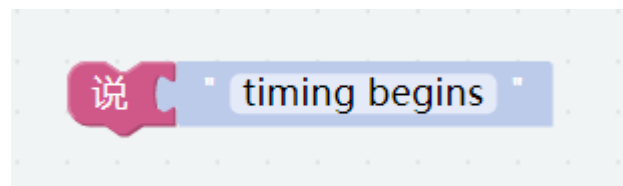
- 这个代码块是一个单独的线程，可以播放一些内置的背景音乐。



- 这个代码块可以播放一些内置的声音效果。



- 这个代码块可以输入一些文字然后让 PiArm 用语音讲出来。



3.3.2 编程

第一步

构建 **函数** 是编程里很重要的一部分，尤其是当执行多次相同的操作时，把这些操作放到一个新声明的函数中，可以极大的简化代码。

点击 **函数** 类别并选择相应的代码块，已经创建好的函数也会出现在这个部分。



我们在这里使用没有返回结果的 **函数块**。



第二步

创建一个名为 [music] 的函数，创建好之后可以在 **函数** 的类别中找到它。

函数 [music] 实现的功能为以 50% 的音量播放背景音乐 peace.mp3。

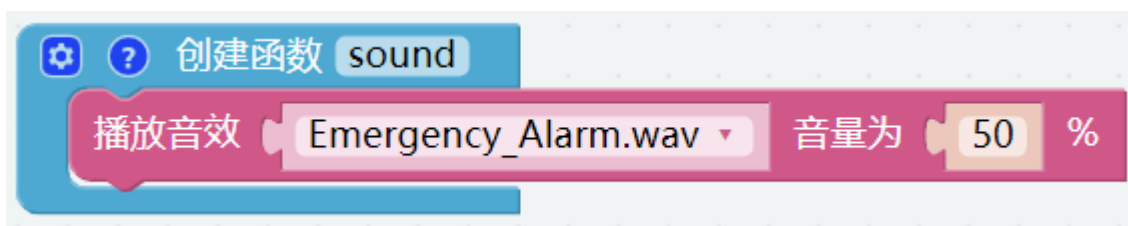
- [设置背景音乐音量为 0]: 用于设置背景音乐的音量，范围为 0%-100%。
- [播放背景音乐 0]: 这个代码块是一个单独的线程，可以播放一些内置的背景音乐。



第三步

创建一个名为 [sound] 的函数，让 PiArm 播放音效。

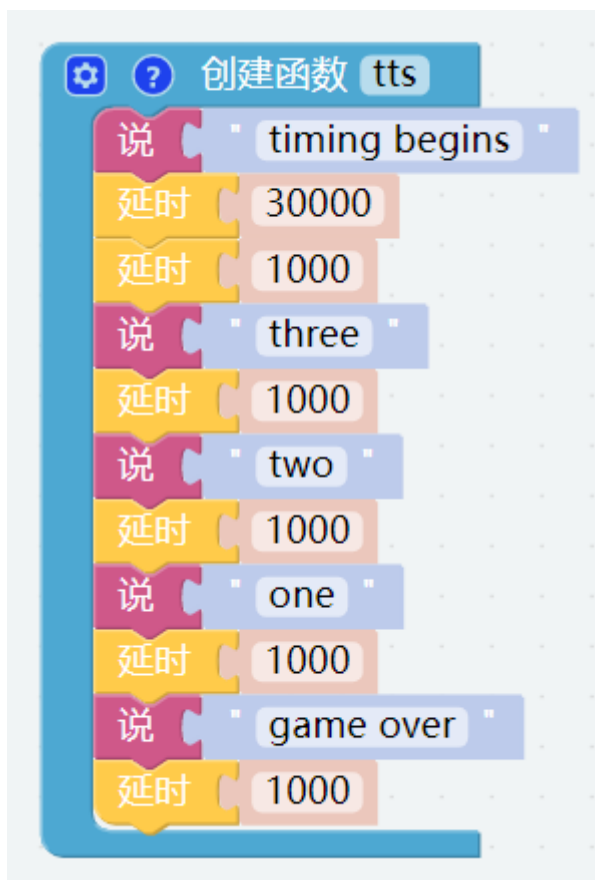
- [播放音效 () 音量为 () %]: 这个代码块可以播放一些内置的音效，音量范围为 0%-100%。



第四步

和上面类似，创建一个名为 [tts] 的函数，功能是让 PiArm 用语言说话。

- [说 ()]: 这个代码块可以将您输入的文字转化成语音，并由 PiArm 读出来。



第五步

在 **函数** 类别中, 将刚刚创建的三个函数拖拽到 [循环] 块中以便按顺序执行它们。



第六步

搭建完代码后, 单击右下角的 **下载** 按钮将其下载到 PiArm。

现在你会发现 PiArm 先在 [sound] 函数中播放音效, 然后在 [music] 函数中播放背景音乐。播放背景音乐时, 运行 [tts] 功能进行计时, 然后在 30 秒后用语音播报倒计时。

注解: 您也可以在 Ezblock Studio 的示例页面找到同名的代码, 直接点击运行或编辑查看代码块。



3.4 双摇杆控制

我们可以分两部分控制 PiArm，手臂和臂端工具。在第一个项目中，您已经学习了如何组装和测试 PiArm 的 3 个臂端工具。在这个项目中，我们将使用套件附带的双摇杆模块来控制 PiArm 的手臂。

在本项目中，用角度模式和双摇杆模块操控机械臂，后面的项目也是在此基础上增添了三个臂端工具的控制代码，达到用双摇杆同时控制 PiArm 的手臂和臂端工具部分的目的。

- 手臂 - 摇杆控制
- 铲斗 - 摇杆控制
- 竖直夹 - 摇杆控制
- 电磁铁 - 摇杆控制



3.4.1 手臂 - 摇杆控制

第一步

在编程的时候常常用变量来简化程序，点击 变量类别中的 创建变量来创建出五个变量 (HIGH, LOW, α , β and γ)。

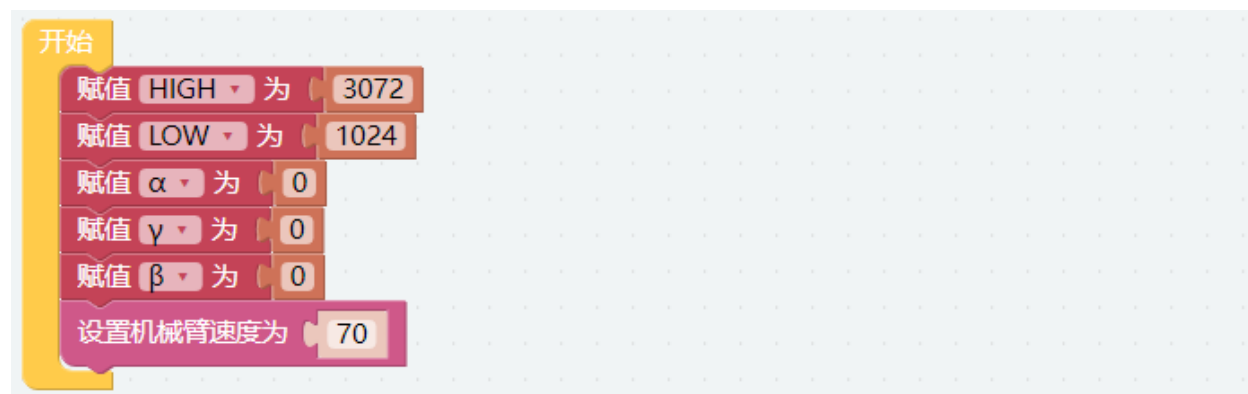
注解：已经创建好的变量也会存放在 变量类别中。



第二步

为这些变量设置初始值并将舵机的转动速度设置为 70%。

注解：变量 **HIGH** 和 **LOW** 如何取值请参考双摇杆模块。

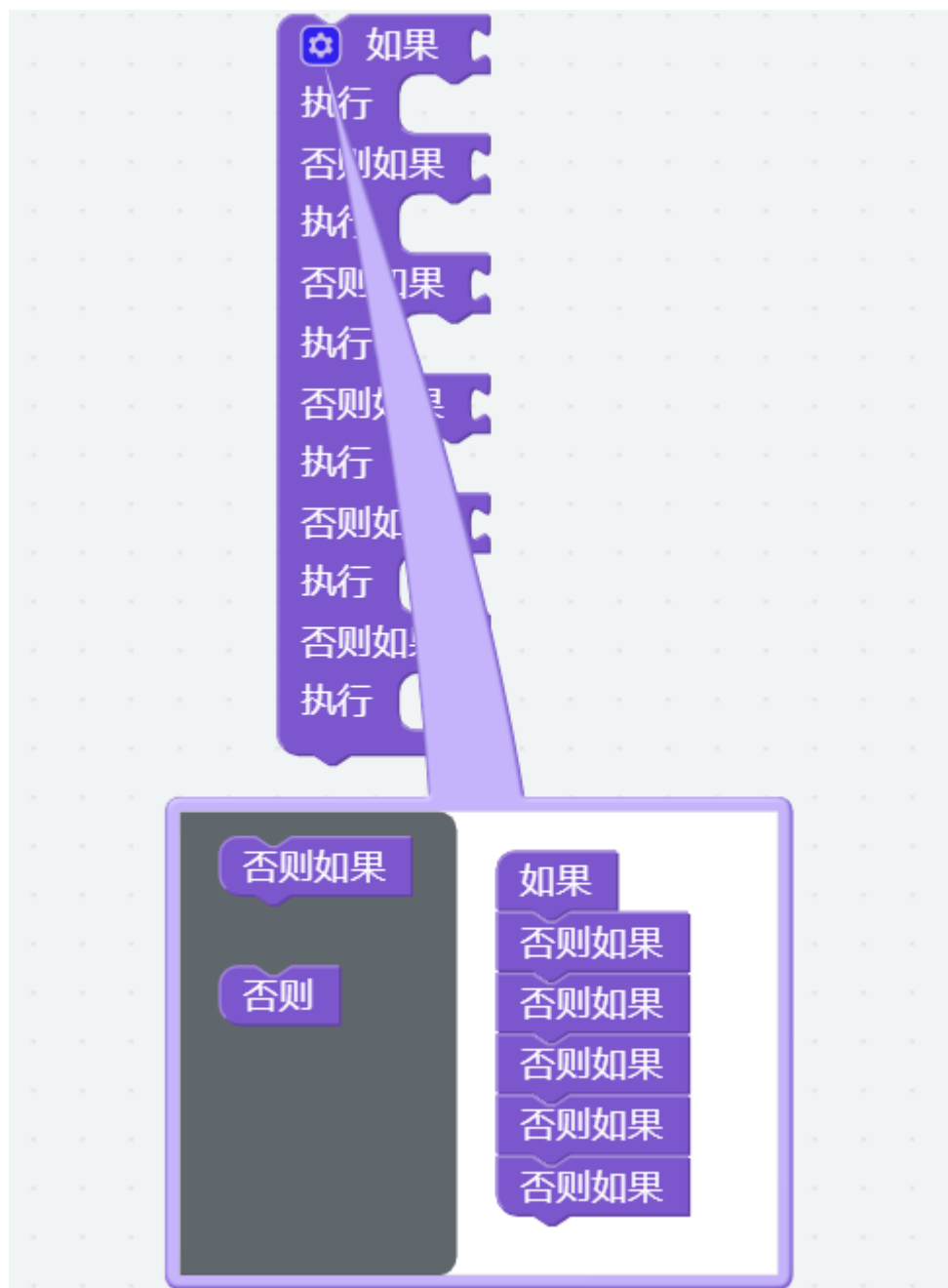


第三步

用 [如果... 执行...] 代码块来做条件判断。

- [否则如果]: 用于条件判断的代码块, 可以点击设置图标将 [否则] 或 [否则如果] 拖拽到 [如果] 下方来创

建多个条件判断。

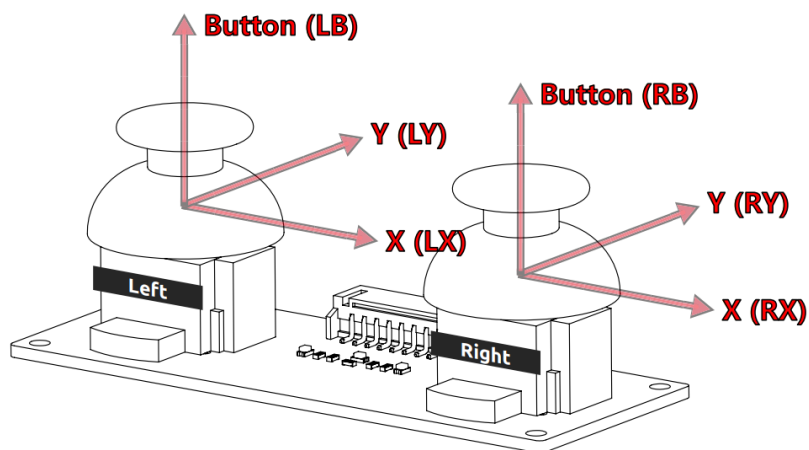


第四步

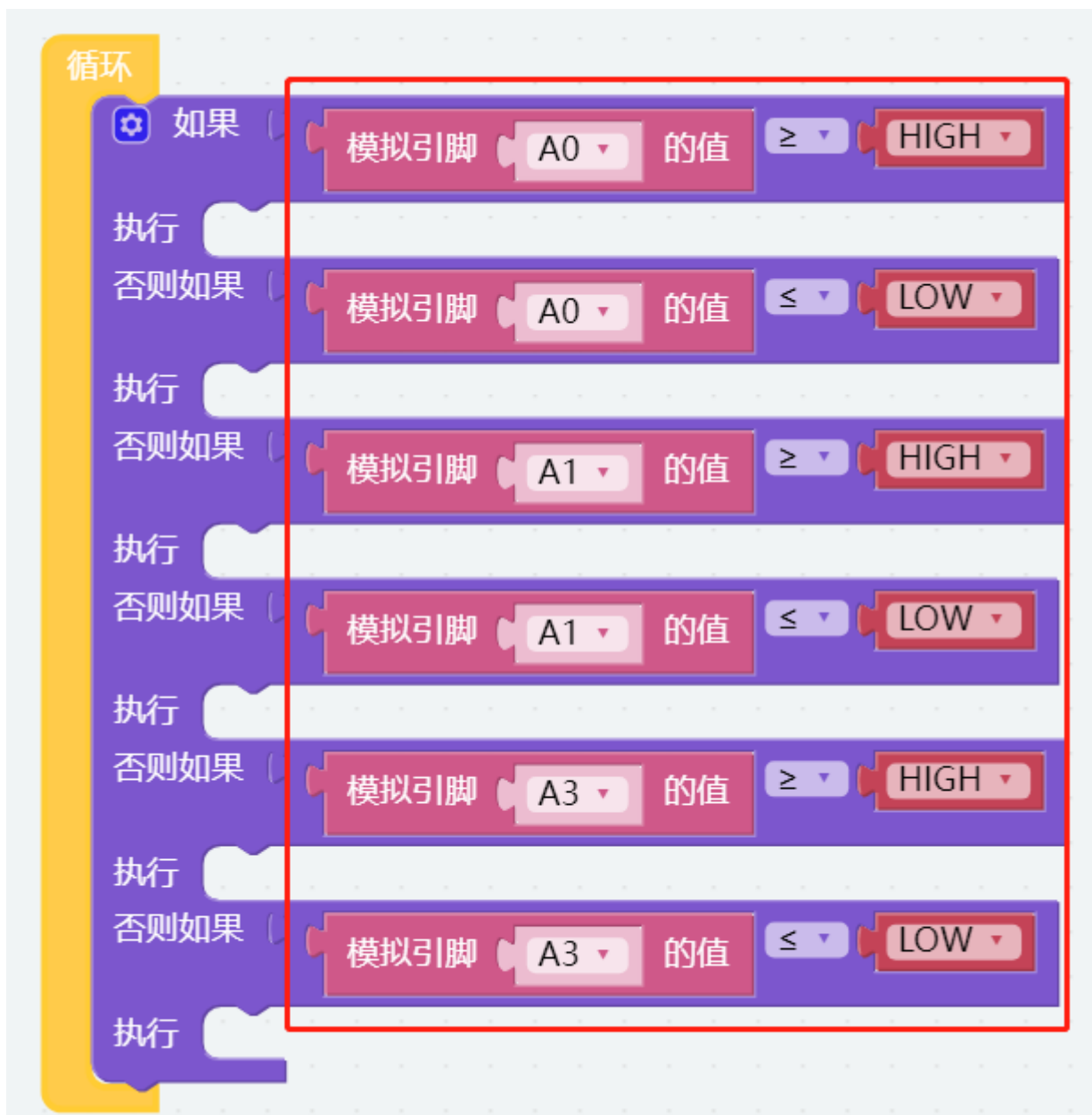
双摇杆模块左右连接如下图所示也可以参考[双摇杆模块](#)。

- 左摇杆的 X 轴连接到 A0，Y 轴连接到 A1。
- 右摇杆的 X 轴连接到 A2，Y 轴连接到 A3。

假定用左摇杆的 XY 轴和右摇杆的 Y 轴来控制机械臂的三个舵机，首先要设置判断条件来判断左右摇杆是否被推动。



- 如果 **A0 (LX)** 大于 **HIGH (3072)**, 则可以确定 左摇杆向右边推动。
- 如果 **A0 (LX)** 小于 **LOW (1024)**, 则可以确定 左摇杆向左边推动。
- 如果 **A1 (LY)** 大于 **HIGH (3072)**, 则可以确定 左摇杆向前推动。
- 如果 **A1 (LY)** 小于 **LOW (1024)**, 则可以确定 左摇杆向后推动。
- 如果 **A3 (RY)** 大于 **HIGH (3072)**, 则可以确定 右摇杆向前推动。
- 如果 **A3 (RY)** 小于 **LOW (1024)**, 则可以确定 右摇杆向后推动。



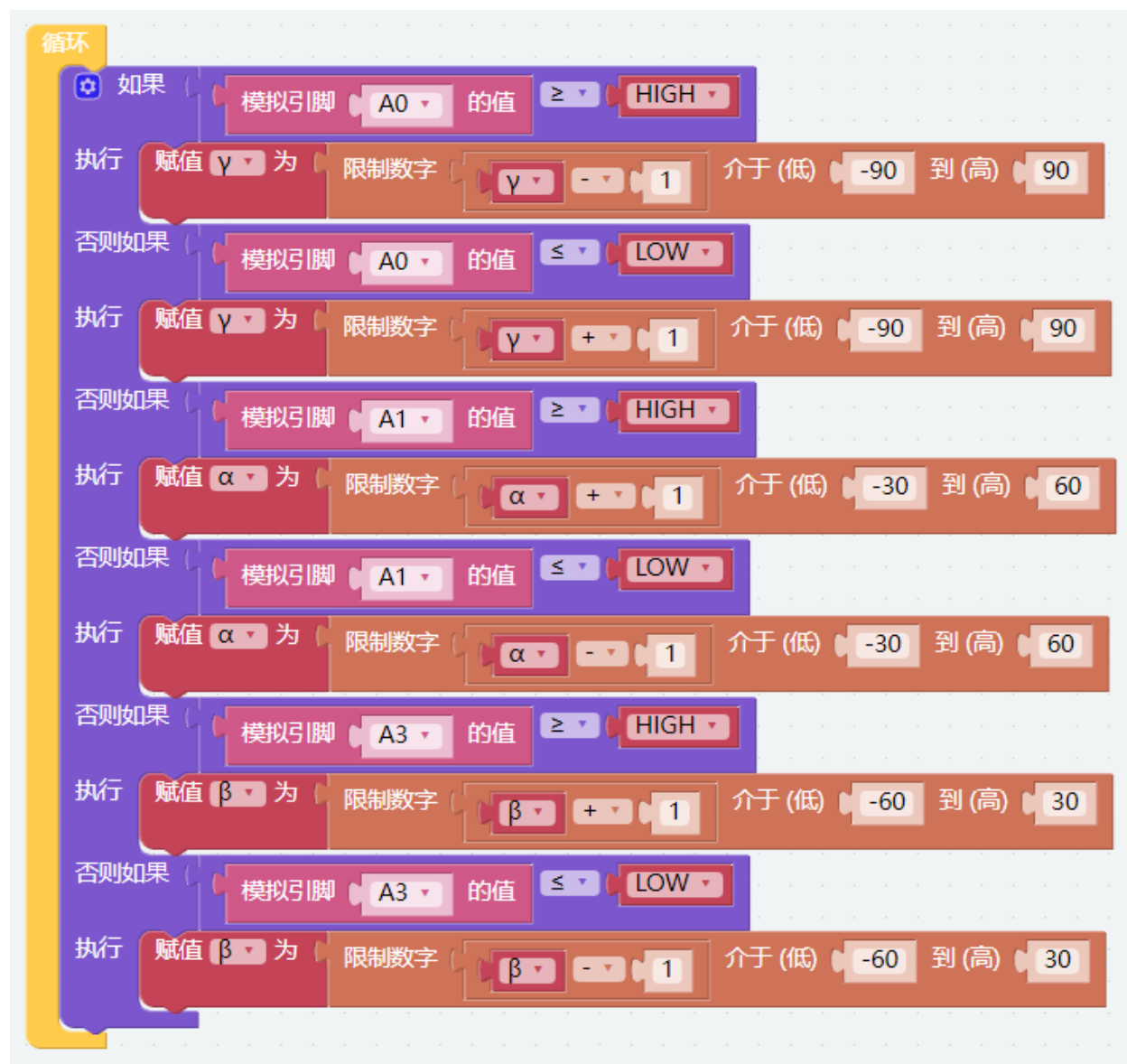
第五步

然后根据左右摇杆的推动方向来设置机械臂移动方向。

- 如果 左摇杆向右推动, 则让 PiArm 向右转动。
- 如果 左摇杆向左推动, 则让 PiArm 向左转动。
- 如果 左摇杆向前推动, 则让 PiArm 向前伸长。
- 如果 左摇杆向后推动, 则让 PiArm 向后缩回。
- 如果 右摇杆向前推动, 则让 PiArm 向上抬高。
- 如果 右摇杆向后推动, 则让 PiArm 向下降低。

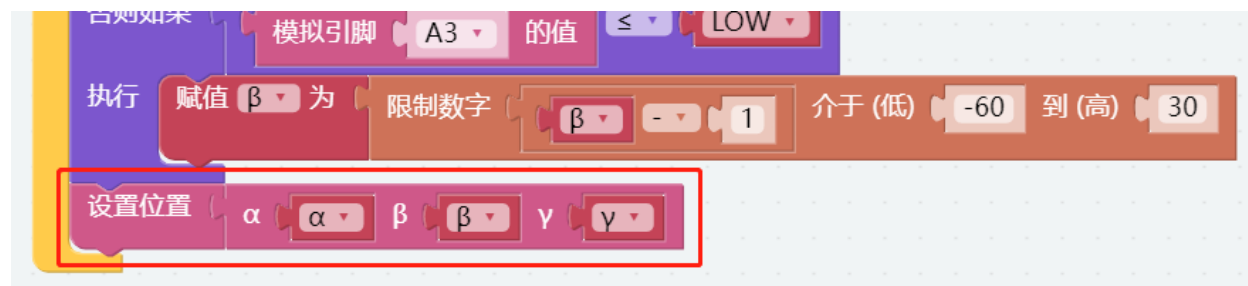
注解:

- α , β 和 γ 表示 PiArm 上三个舵机的旋转范围, 参考: 角度模式。
- [限制数字... 介于 (低) ... 到 (高) ...]: 可以在数学类中找到, 用于设置一个变量的变化范围。



第六步

将获取到的 α , β 和 γ 角度值放入 $[\alpha() \beta() \gamma()]$ 代码块中, 然后用 [设置位置] 代码块使 PiArm 旋转到该位置。

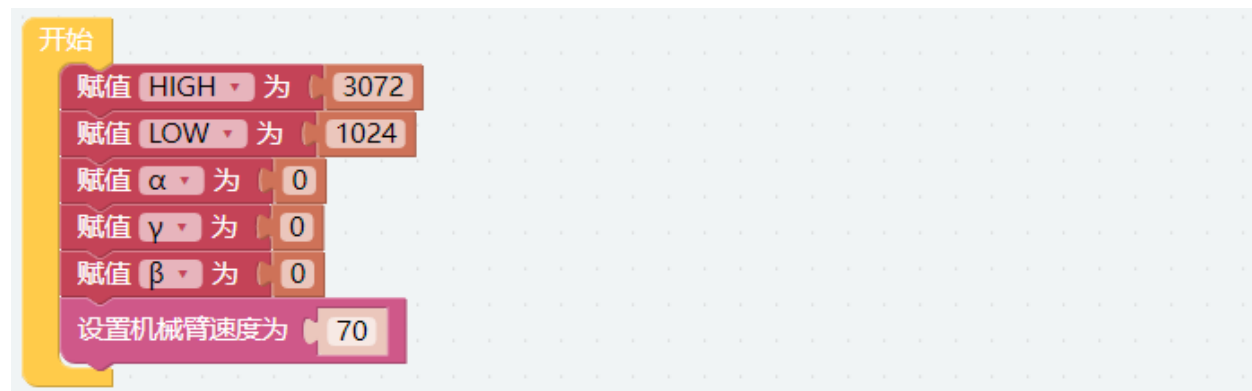


第七步

单击下载按钮后，便可以用双摇杆模块来控制 PiArm 了。

- 左摇杆向左或向右推动, 机械臂将向左或向右转动。
- 左摇杆向前或向后推动, 机械臂会伸出或者缩回。
- 右摇杆向前或向后推动, 机械臂会抬高或降低。

注解：您也可以在 Ezblock Studio 的 **示例**页面找到同名的代码，直接点击运行或编辑查看代码块。

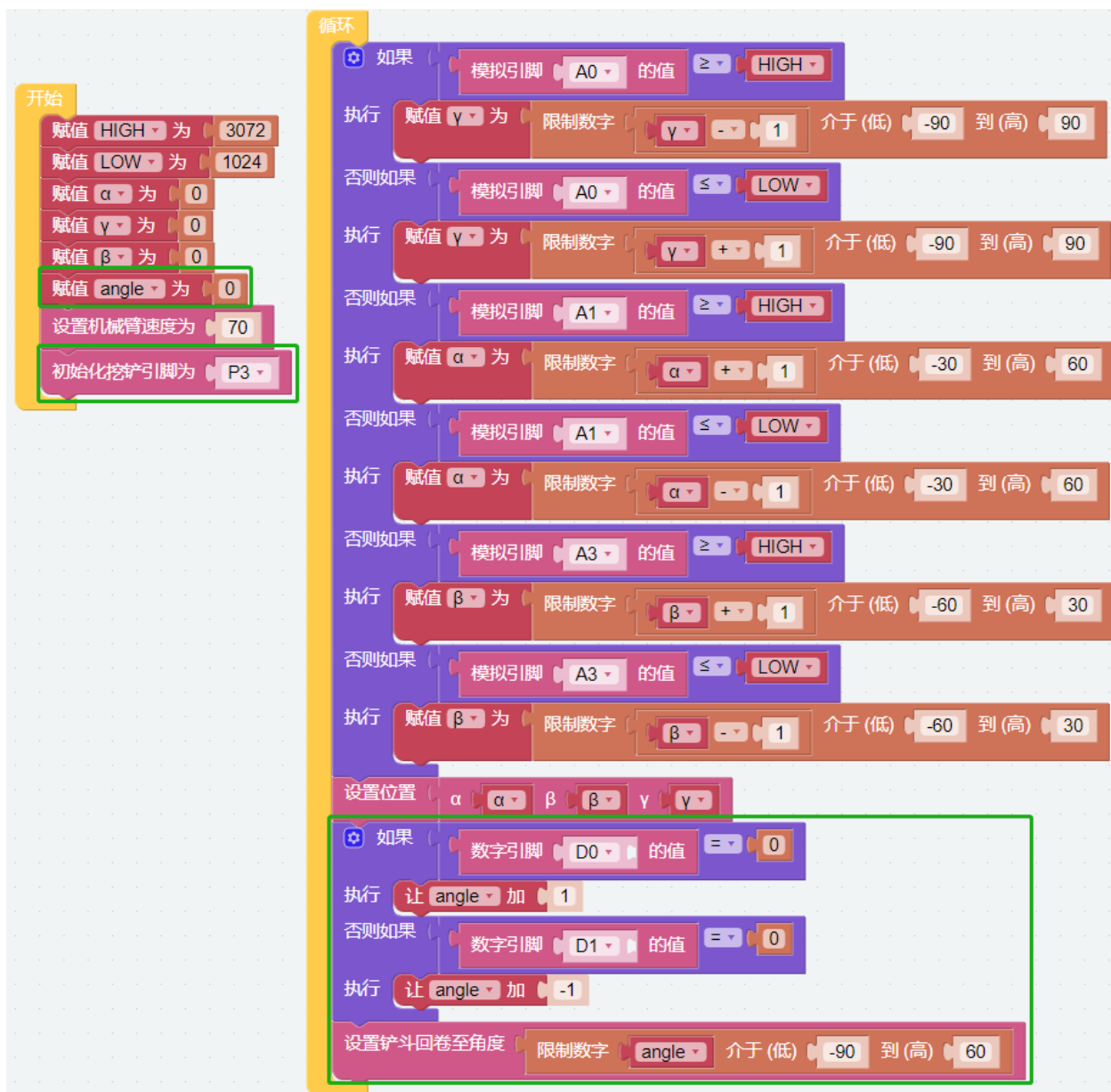




3.4.2 铲斗 - 摇杆控制

现在在控制手臂的代码基础上加上铲斗的控制代码。

注解：您也可以在 Ezblock Studio 的 [示例](#) 页面找到同名的代码，直接点击运行或编辑查看代码块。



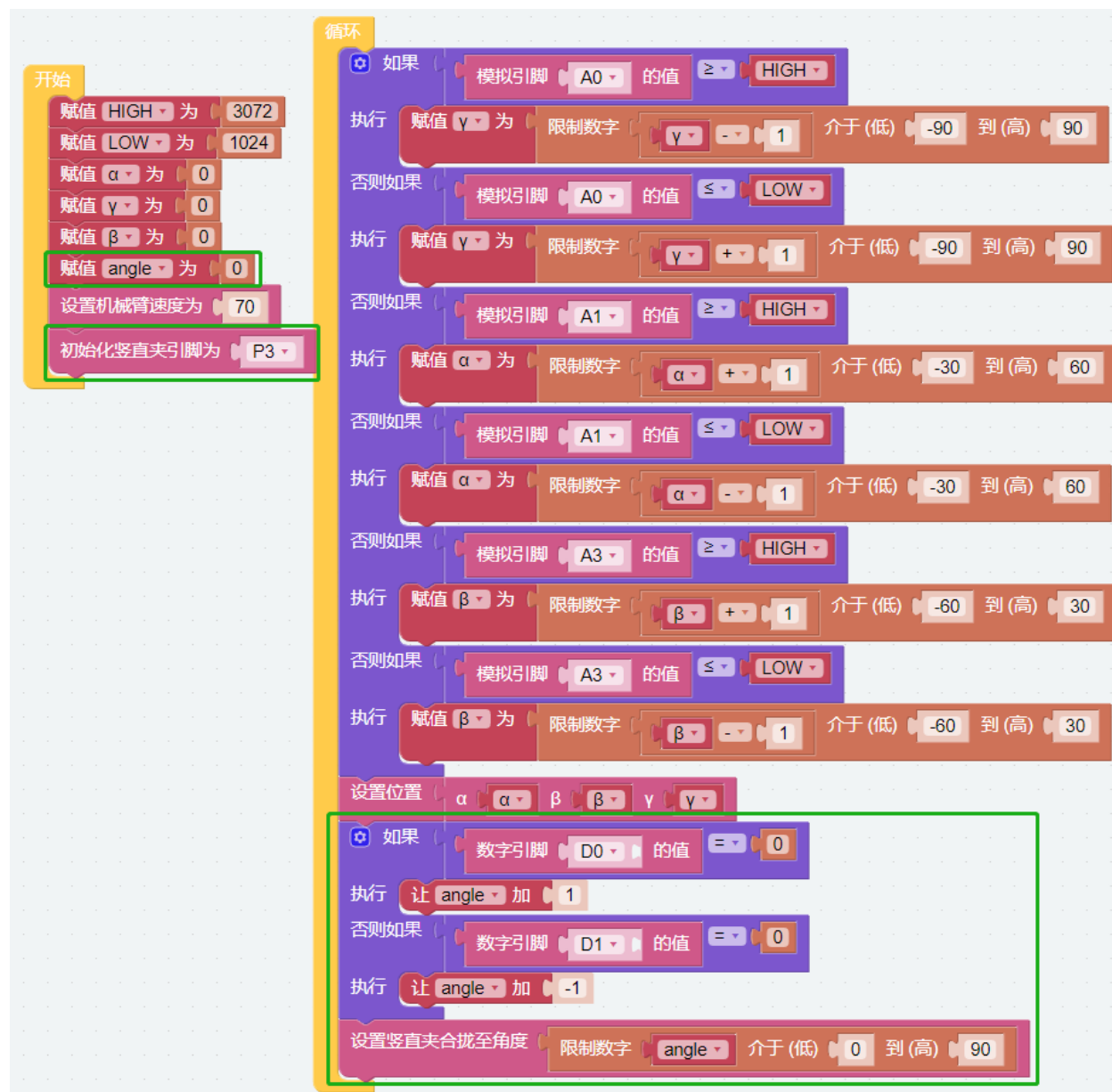
代码运行后，你可以用双摇杆模块同时控制 PiArm 的手臂和铲斗。但你需要先将铲斗安装到 PiArm 上。

- 左摇杆向左或向右推动，机械臂将向左或向右转动。
- 左摇杆向前或向后推动，机械臂会伸出或者缩回。
- 右摇杆向前或向后推动，机械臂会抬高或降低。
- 按下左摇杆，铲斗向内回卷。
- 按下右摇杆，铲斗向外伸展。

3.4.3 竖直夹 - 摇杆控制

现在在控制手臂的代码基础上加上竖直夹的控制代码。

注解：您也可以在 Ezblock Studio 的 示例 页面找到同名的代码，直接点击运行或编辑查看代码块。



代码运行后，你可以用双摇杆模块同时控制 PiArm 的手臂和竖直夹。但你需要先将竖直夹 安装到 PiArm 上。

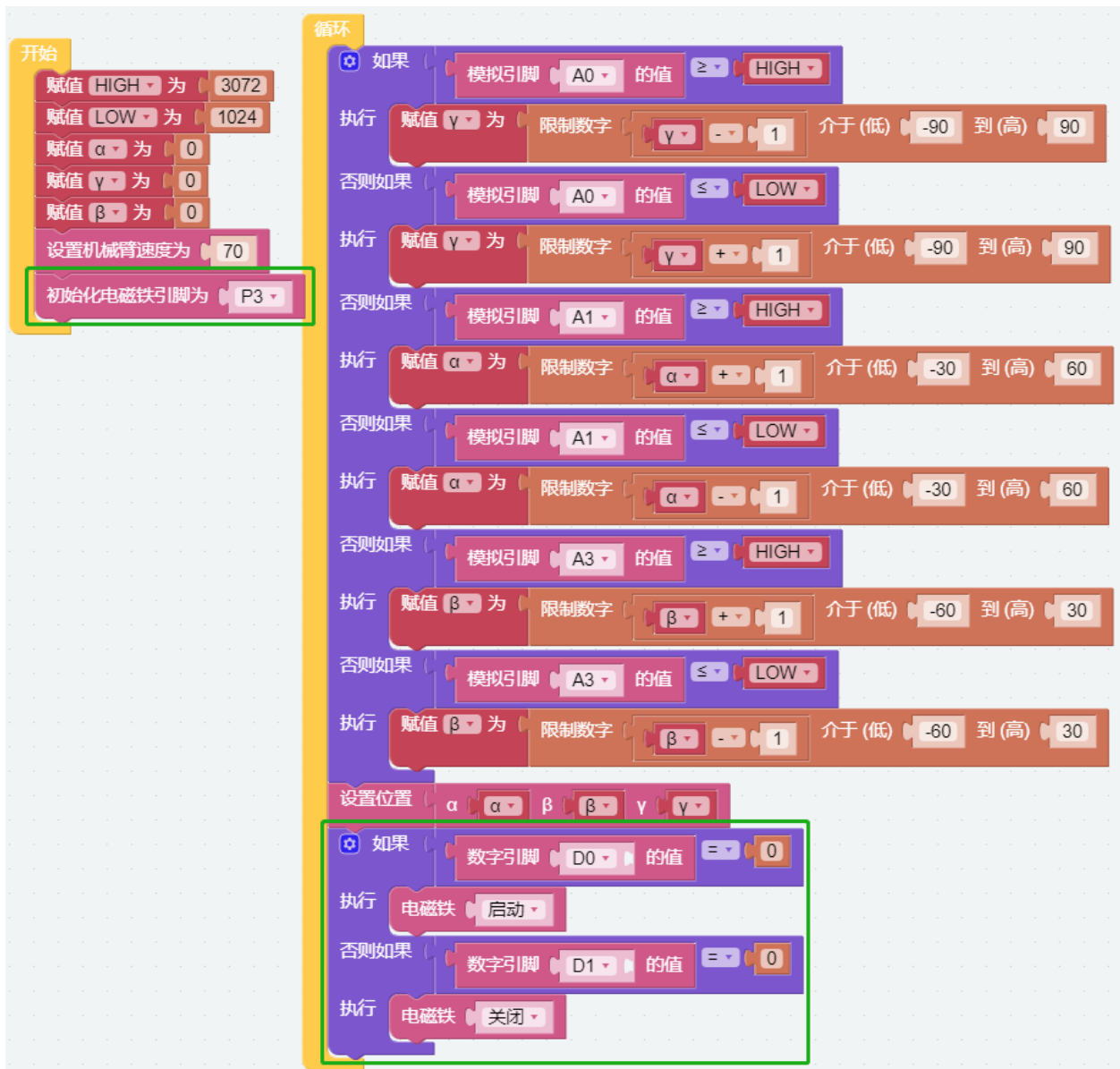
- 左摇杆向左或向右推动, 机械臂将向左或向右转动。
- 左摇杆向前或向后推动, 机械臂会伸出或者缩回。
- 右摇杆向前或向后推动, 机械臂会抬高或降低。
- 按下左摇杆, 竖直夹合拢。

- 按下右摇杆，竖直夹张开。

3.4.4 电磁铁 - 摇杆控制

现在在控制手臂的代码基础上加上电磁铁的控制代码。

注解：您也可以在 Ezblock Studio 的 **示例** 页面找到同名的代码，直接点击运行或编辑查看代码块。



代码运行后，你可以用双摇杆模块同时控制 PiArm 的手臂和电磁铁。但你需要先将电磁铁安装到 PiArm 上。

- 左摇杆向左或向右推动，机械臂将向左或向右转动。
- 左摇杆向前或向后推动，机械臂会伸出或者缩回。
- 右摇杆向前或向后推动，机械臂会抬高或降低。

- 按下左摇杆，电磁铁开启。
- 按下右摇杆，电磁铁关闭。

3.5 远程遥控

除了双摇杆模块，我们还可以用 **EzBlock Studio** 里 **远程遥控** 页面上的控件来控制 **PiArm** 移动。

- 手臂 - 远程遥控
- 创建库
- 铲斗 - 远程遥控
- 竖直夹 - 远程遥控
- 电磁铁 - 远程遥控



3.5.1 手臂 - 远程遥控

PiArm 的手臂有两种控制模式：**角度**和 **坐标**。

- **角度模式**: 向手臂上的 3 个舵机写入一定的角度，使 PiArm 到达特定位置。
- **坐标模式**: 为手臂建立空间坐标系并设置控制点，将空间坐标写入该控制点，使 PiArm 达到特定位置。

这个项目用的是**角度模式**。

第一步

要使用远程控制功能，您需要从主页面左侧进入远程控制页面，然后拖动一个方向键和 2 个按钮到中央区域。



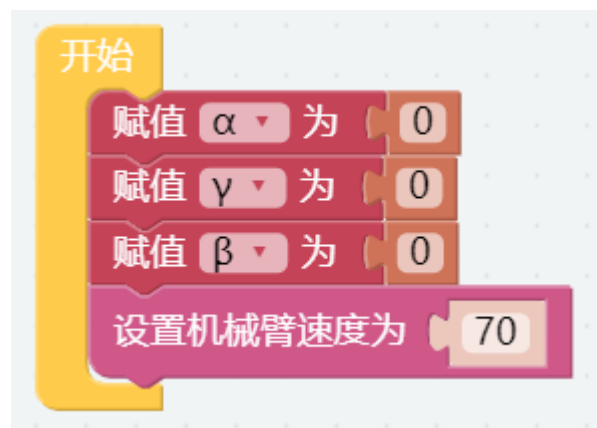
回到编程页面，您将看到一个附加的远程控制类别，其中出现方向盘和按键代码块。

- [按键 () 获取值]: 这个代码块用于读取按钮的值，按下时为 1，松开时为 0。
- [按键 () 被 (按下/松开)]: 这个代码块和 按键 () 获取值 (0/1) 有着相似的效果，可以直接判断按钮是否被按下。
- [方向盘 () 获取 () 值]: 这个代码块用于读取方向盘各个方向上的值，按下时为 1，松开为 0。



第二步

创建 3 个变量 (α , β and γ) 并设置初始值, 并将 PiArm 的旋转速度设置为 70%。



第三步

创建一个名为 [arm_control] 的函数, 根据方向键和按钮值设置 PiArm 的旋转方向。

注解：函数名不能包含空格，两个单词之间可以用 _ 连接。

注解：

- α , β 和 γ 表示 PiArm 手臂上三个舵机的旋转范围, 可参考: 角度模式。
- [限制数字... 介于 (低) ... 到 (高) ...]: 可以在数学类中找到, 用于设置一个变量的变化范围。
- [否则如果]: 用于条件判断的代码块, 可以点击设置图标将 [否则] 或 [否则如果] 拖拽到 [如果] 下方来创建多个条件判断。



- 如果按下方向盘的上按钮 (▲), 则让 PiArm 向前伸长。
- 如果按下方向盘的下按钮 (▼), 则让 PiArm 向后缩回。
- 如果按下方向盘的左按钮 (◀), 则让 PiArm 向左转动。
- 如果按下方向盘的右按钮 (▶), 则让 PiArm 向右转动。
- 如果按下按钮 A, 则让 PiArm 下降。

- 如果按下按钮 B, 则让 PiArm 上升。

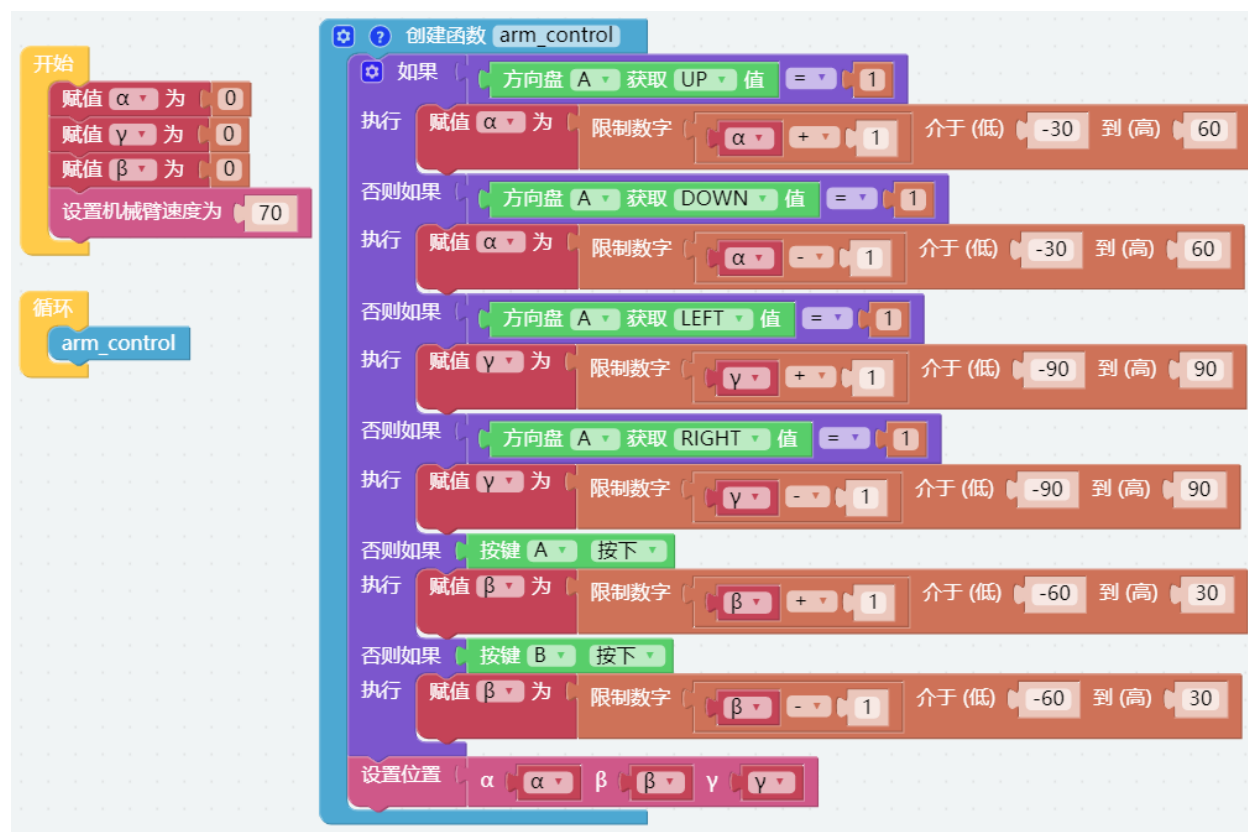
第四步

将函数 [arm_control] 放在 [循环] 块中循环执行, 然后点击 **下载** 按钮来运行代码。

在这之后就可以用 **远程遥控** 页面上的方向盘和按钮 A/B 来控制机械臂的移动了。

注解:

- 放置这些函数块时, 函数必须放在 [开始] 和 [循环] 块之前。
- 您也可以在 EzBlock Studio 的示例页面找到同名的代码, 直接点击运行或编辑查看代码块。



3.5.2 创建库

为了方便以后能在其他代码中使用 [arm_control] 函数, 可以将其创建为库, 并在需要使用时导入。

第一步

打开右上角的菜单图标, 然后选择 **创建库**。



第二步

选择函数，我们这里只创建了一个函数，所以默认选择 `arm_control`。



第三步

为库命名并填写描述，一边以后更好地区分。

请添加名称和描述

名称:

arm_control

描述:

通过方向盘和两个按钮控制机械臂

完成

第四步

等待提示保存成功，然后该库将保存在您个人页面的 **我的库** 里面。随后便可以在导入库的时候看见它了。

<

我的库 | 所有库 | 系统库

Q

arm_control	CQ 2021/12/22	导入
通过方向盘和两个按钮控制机械臂		

3.5.3 铲斗 - 远程遥控

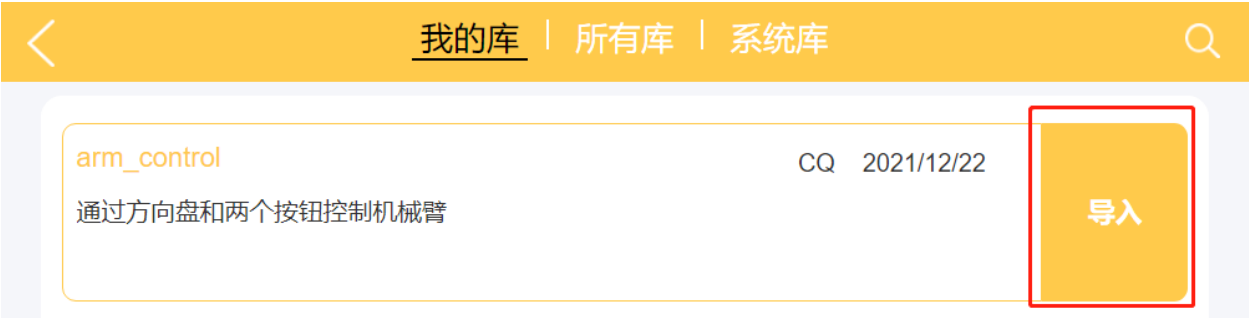
创建一个新项目并为其搭建代码块，然后我们就可以在控制手臂的同时控制铲斗了。

第一步

导入 [arm_control] 库, 如果之前没有创建过这个库, 请参考: [创建库](#)。



在 **我的库** 页面中, 选择您创建的库并单击 **导入**。



导入后, 库为折叠样式。您可以右键单击它然后选择 **展开块**, 这样就可以看到它的内部代码。



第二步

进入到远程控制界面, 重新拖一个 **D-pad**, 两个 **按键** 出来, 因为导入库不会把控件也导进来, 需要重新拖。另外再另外添加两个 **按键** 控件用来控制铲斗的角度。



第三步

创建变量 (α , β , γ 和 angle) 并设置初始值为 0, 然后初始化 PiArm 的旋转速度和铲斗的引脚。



第四步

新建一个函数 [shovel], 按如下步骤编写代码, 即可通过按钮来控制铲斗了。

- 使用 [如果... 执行...] 代码块作判断条件。如果按下按钮 C, 则让变量 angle 加 1; 如果按下按钮 D, 则让变量 angle 减 1。

- 用 [限制数字 angle 介于 (低) -90 到 (高) 60] 将变量 angle 的值限制为 -90 ~ 60。
- 根据变量 angle 设置铲斗的角度。



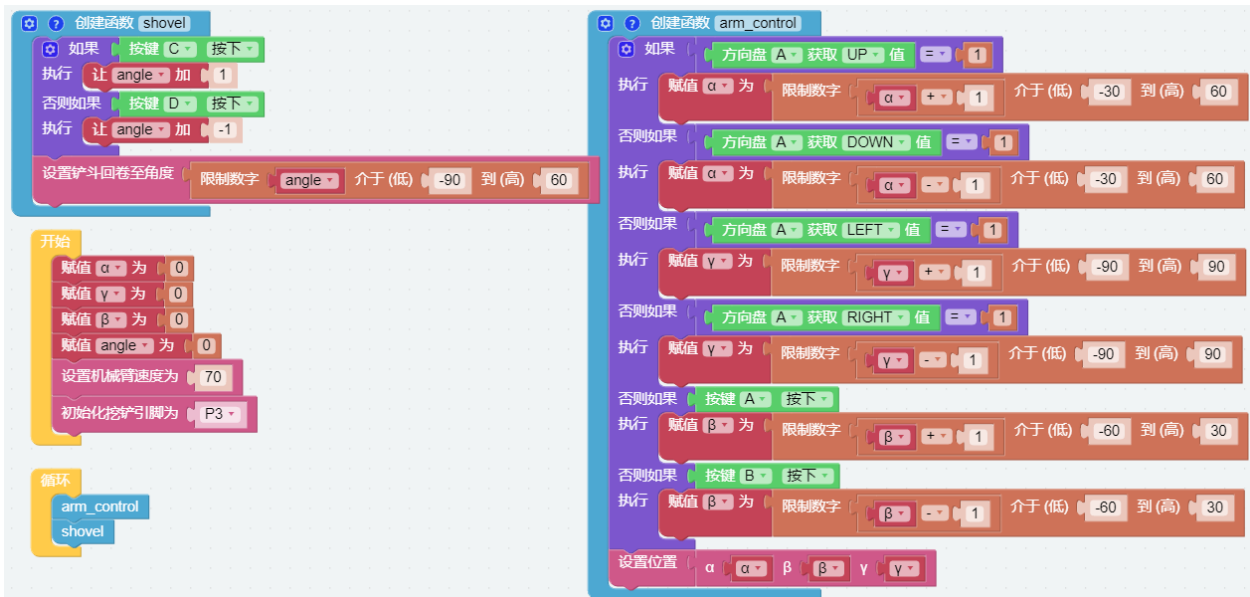
第五步

将 [arm_control] 和 [shovel] 函数分别从 函数类别中拖到 [循环] 块里。

点击下载按钮后，用远程遥控页面上的方向盘和按钮 A/B 来控制机械臂的移动，然后用按钮 C/D 来控制铲斗角度的加/减。

注解：

- 放置这些函数块时，函数必须放在 [开始] 和 [循环] 块之前。
- 您也可以在 EzBlock Studio 的示例页面找到同名的代码，直接点击运行或编辑查看代码块。



3.5.4 竖直夹 - 远程遥控

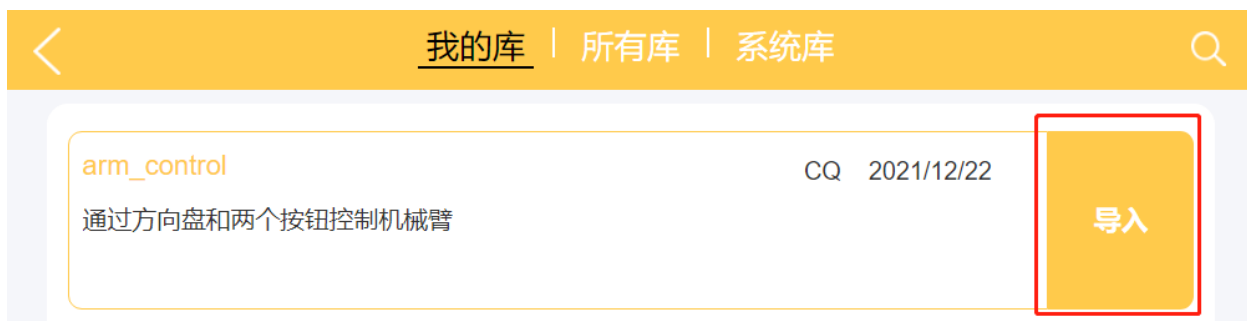
创建一个新项目并为其编写代码，这样我们就可以在控制机械臂的同时控制竖直夹。

第一步

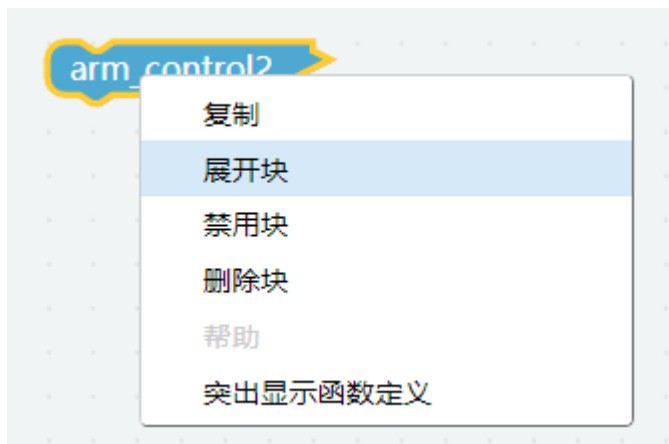
导入 [arm_control] 库，如果之前没有创建过这个库，请参考: [创建库](#)。



在 **我的库** 页面, 选择您创建的库并单击 **导入**.



导入后，此库为折叠样式。您可以右键单击它并单击 **展开块**，这样您就可以看到它的内部代码。



第二步

进入到远程控制界面，重新拖一个 **D-pad**，两个 **按键** 出来，因为导入库不会把控件也导进来，需要重新拖。另外再另外添加两个 **按键** 控件用来控制竖直夹的角度。



第三步

创建变量 (α , β , γ 和 angle) 设置初始值为 0, 然后初始化 PiArm 的转速和竖直夹引脚。



第四步

新建一个函数 [clip]，按如下步骤编写代码，即可通过按钮控制竖直夹。

- 使用 [如果... 执行...] 代码块作判断条件。如果按下按钮 C，则让变量 angle 加 1；如果按下按钮 D，则让变量 angle 减 1。
- 用 [限制数字 angle 介于（低）0 到（高）90] 将变量 angle 的值限制为 0-90，因为竖直夹的活动范围是 0-90。
- 根据变量 angle 设置竖直夹活动的角度。



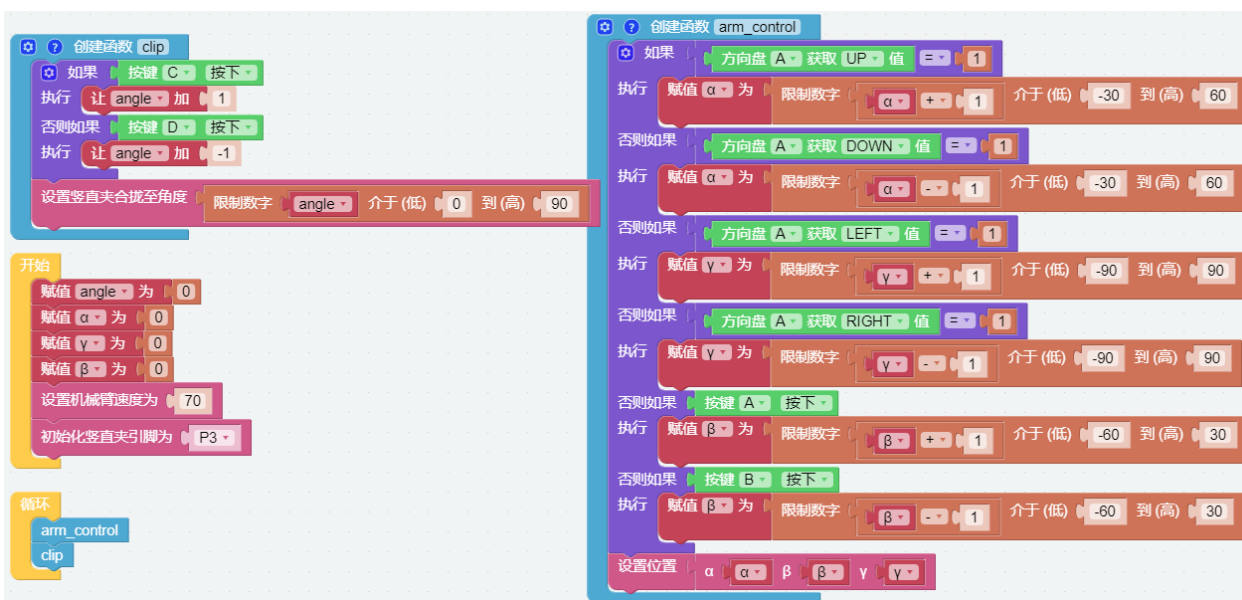
第五步

将 [arm_control] 和 [clip] 函数分别从 函数类别 拖到 [循环] 块。

点击下载按钮后，使用远程遥控页面上的方向键和按钮 A/B 来控制机械臂的移动，然后用按钮 C/D 来控制竖直夹的张合。

注解：

- 放置这些函数块时，函数必须放在 [开始] 和 [循环] 块之前。
- 您也可以在 EzBlock Studio 的示例页面找到同名的代码，直接点击运行或编辑查看代码块。



3.5.5 电磁铁 - 远程遥控

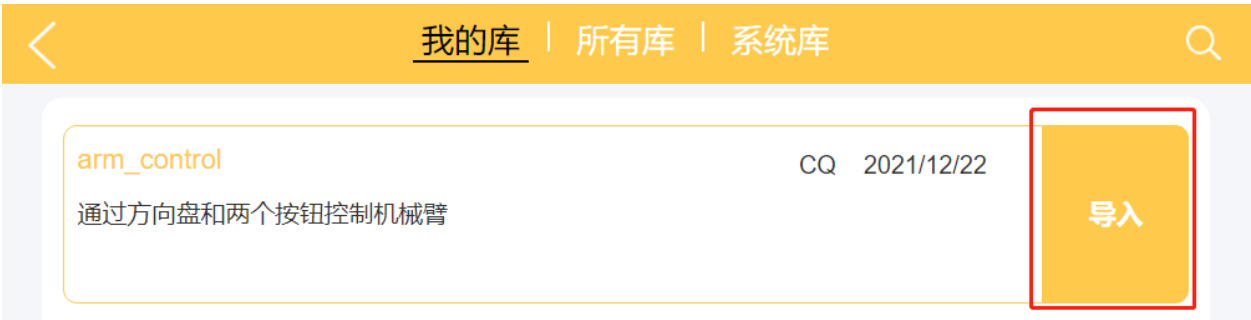
创建一个新项目并为其编写代码，这样我们就可以在控制机械臂的同时控制电磁铁。

第一步

导入 [arm_control] 库, 如果之前没有创建过这个库, 请参考: [创建库](#)。



在 **我的库** 页面, 选择您创建的库并单击 **导入**。



导入后, 此库为折叠样式。您可以右键单击它并选择 **展开块**, 这样您就可以看到它的内部代码。



第二步

进入到远程控制界面，重新拖一个 **D-pad**, 两个 **按键** 出来，因为导入库不会把控件也导进来，需要重新拖。另外再添加一个 **开关** 控件用来控制电磁铁的开/关。



第三步

创建变量 (α , β and γ) 并将初始值设置为 0, 然后初始化 PiArm 的旋转速度和电磁铁的引脚。



第四步

创建一个新的函数 [electromagnet], 按照如下步骤编写代码，这样就可以通过开关控件控制电磁铁了。

- 使用 [如果... 执行...] 块作为判断条件。开关是开，电磁铁启动；开关是关，电磁铁关闭。



第五步

将 [arm_control] 和 [electromagnet] 函数分别从 函数类别拖到 [循环] 块。

点击下载按钮后，当开关切换到开，电磁铁开启（此时是磁性的，可以用铁吸附材料）；当开关切换到关，电磁铁关闭。同时，您可以使用遥控器页面上的方向键和按钮 A/B 来控制手臂的移动。

注解：

- 放置这些函数块时，函数必须放在 [开始] 和 [循环] 块之前。
- 您也可以在 EzBlock Studio 的示例页面找到同名的代码，直接点击运行或编辑查看代码块。

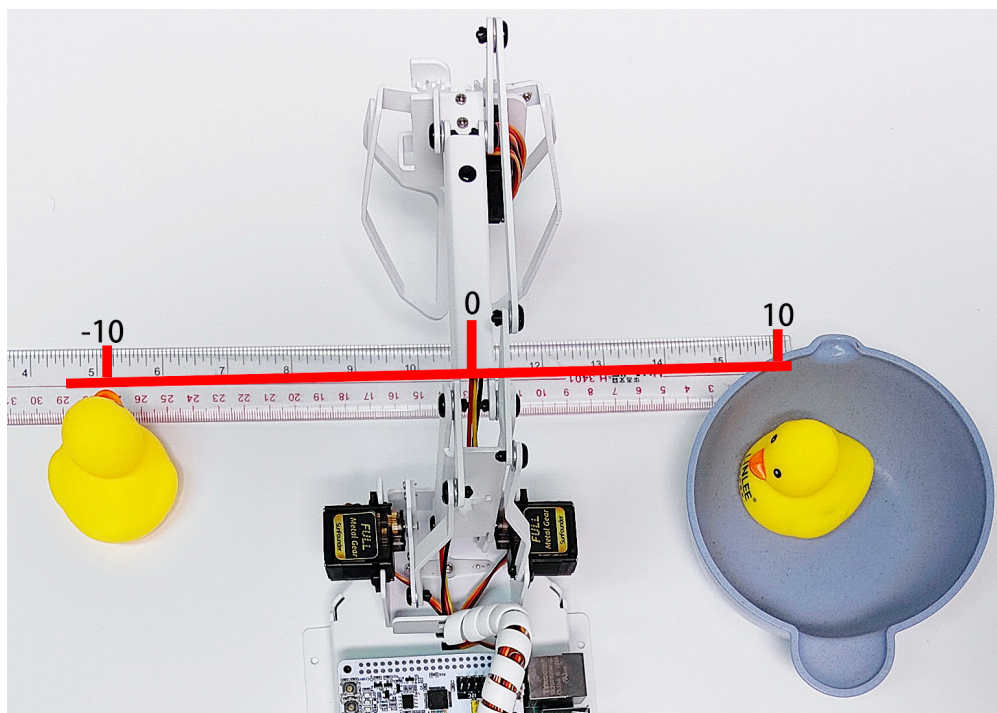


3.6 坐标模式

PiArm 的手臂有 2 种控制模式: **角度**和 **坐标**。

- **角度模式**: 向 PiArm 手臂上的 3 个舵机写入一定的角度, 使其到达特定位置。
- **坐标模式**: 为 PiArm 建立空间坐标系并设置控制点, 将空间坐标写入该控制点, 使其达到特定位置。

本项目通过坐标模式, 设定 2 个坐标点, 让机械臂将左边的橡皮鸭夹到右边的碗里。但你需要先将**竖直夹**装到 PiArm 上。



3.6.1 编程

第一步

初始化竖直夹的引脚及设置机械臂的速度为 60%。



第二步

设置 2 个点的坐标。由于左边橡皮鸭和右边的碗在同一条线上, 你会发现它们的 Y 坐标值是一样的。

- [start_coord]: 左边橡皮鸭的坐标。
- [start_coord_up]: 左边橡皮鸭的正上方坐标。
- [end_coord]: 碗的坐标。
- [end_coord_up]: 碗的正上方坐标。

注解:

- 这里的坐标都是指的控制点的坐标，但是装好臂端工具后，X 和 Y 坐标的实际距离大一点。
- 不同的臂端工具，误差距离不一样。比如竖直夹和电磁铁为 3-4cm, 铲斗为 6-7cm。
- 比如在这里 X 坐标写的是 100，但实际距离是 13-14cm。
- 一般建议 X 的坐标是 -80 ~ 80，但由于这里 Y 坐标值较小（建议范围是 30~130），所以设置为 100 也是能到的。但如果你增大了 Y 坐标值，由于连杆作用，X 坐标值需要根据实际情况调小一点。



第三步

在 [循环] 块中，让 PiArm 完成以下动作。

- PiArm 先张开竖直夹（20°），然后转动到左边橡皮鸭的位置（start_coord），再合拢竖直夹（90°）。
- PiArm 抬起头（start_coord_up），再转动到右边的碗的正上方（end_coord_up）。
- PiArm 低头（end_coord_up），再张开竖直夹（20°）让橡皮鸭掉落到碗里，最后再抬起头（end_coord_up）。



第四步

点击右下角的 **下载** 按钮，你就能看到 PiArm 重复上面描述的动作。

注解：您也可以在 EzBlock Studio 的示例页面找到同名的代码，直接点击运行或编辑查看代码块。

开始



循环



3.7 记录功能

PiArm 的手臂有 2 种控制模式: **角度**和 **坐标**。

- **角度模式**: 向 PiArm 手臂上的 3 个舵机写入一定的角度, 使其到达特定位置。
- **坐标模式**: 为 PiArm 建立空间坐标系并设置控制点, 将空间坐标写入该控制点, 使其达到特定位置。

本项目用的是**坐标模式**。

在本项目中, 我们将使用双摇杆模块在坐标模式下控制 PiArm 移动, 并通过摇杆按钮记录其移动轨迹, 以便 PiArm 可以沿着记录的轨迹重复移动。

3.7.1 编程

第一步

创建五个变量 (HIGH, LOW, xAxis, yAxis 和 zAxis) 并为它们设置初始值。



第二步

创建一个名为 [set_position] 的函数, 用双摇杆模块在坐标模式下控制 PiArm 。

- 如果 **左摇杆**向右推动, 则让 PiArm 向右转动。
- 如果 **左摇杆**向左推动, 则让 PiArm 向左转动。
- 如果 **左摇杆**向前推动, 则让 PiArm 向前伸长。
- 如果 **左摇杆**向后推动, 则让 PiArm 向后缩回。
- 如果 **右摇杆**向前推动, 则让 PiArm 向上抬高。
- 如果 **右摇杆**向后推动, 则让 PiArm 向下降低。



注解:

- 关于 X、Y、Z 坐标方向, 请参考: [坐标模式](#)。
- 双摇杆的连接和方向请参考 [双摇杆模块](#)。
- [限制数字... 介于 (低) ... 到 (高) ...]: 可以在数学类中找到, 用于设置一个变量的变化范围。
- [否则如果]: 用于条件判断的代码块, 可以点击设置图标将 [否则] 或 [否则如果] 拖拽到 [如果] 下方来创建多个条件判断。

第三步

创建了一个新函数 [record] 来记录当前动作并允许 PiArm 重现它们。

- 双摇杆模块的左右按钮分别连接到 D0 (左按钮)、D1 (右按钮)。
- 按键按下时输出低电平 (0), 松开时输出高电平 (1)。
- 当按下左摇杆的按钮时, 此时会记录 PiArm 的动作, 并有语音提示, 表示记录完成。

- 当按下右摇杆的按钮时，PiArm 会重复已经记录好的动作。



注解:

- 代码块 [如果... 执行...], [... 和...] 以及 [=] 都来自 **逻辑** 类别。
- [运行动作集 (0) 设置延迟]: 该代码块用于设置记录好的每组动作的时间间隔，如果为 0 则表示没有间隔时间连续重复之前的动作。

第四步

将 [set_position] 和 [record] 函数放入 [循环] 块中依次执行，最后点击下载按钮运行代码。

现在你可以使用摇杆来控制 PiArm，按下左摇杆的按钮记录需要的动作，记录几组后，按下右摇杆的按钮，让 PiArm 复现这些动作。

注解: 您也可以在 Ezblock Studio 的 **示例** 页面找到同名的代码，直接点击运行或编辑查看代码块。



3.7.2 拓展

你也可以在这个项目中添加单独的臂端工具控制代码，这样你就可以同时控制 PiArm 的手臂和臂端工具了。

- 如果想要控制铲斗，请参考铲斗 - 摇杆控制 来搭建代码。
- 如果想要控制竖直夹，请参考竖直夹 - 摇杆控制 来搭建代码。
- 如果想要控制电磁铁，请参考电磁铁 - 摇杆控制 来搭建代码。

3.8 游戏 - 抓娃娃

现在让我们玩一个抓娃娃的游戏，看看谁能在给定的时间内用 PiArm 抓到更多的娃娃。为了玩这个游戏，我们需要实现两个功能，第一个是用双摇杆模块控制 PiArm，第二个是计时，当倒计时结束时，我们便不能继续 PiArm 了。这两部分必须同时进行。



3.8.1 编程

第一步

创建变量 (HIGH, LOW, α , β , γ , flag, angle) 并为它们设置初始值。然后初始化 PiArm 旋转速度和竖直夹的引脚。

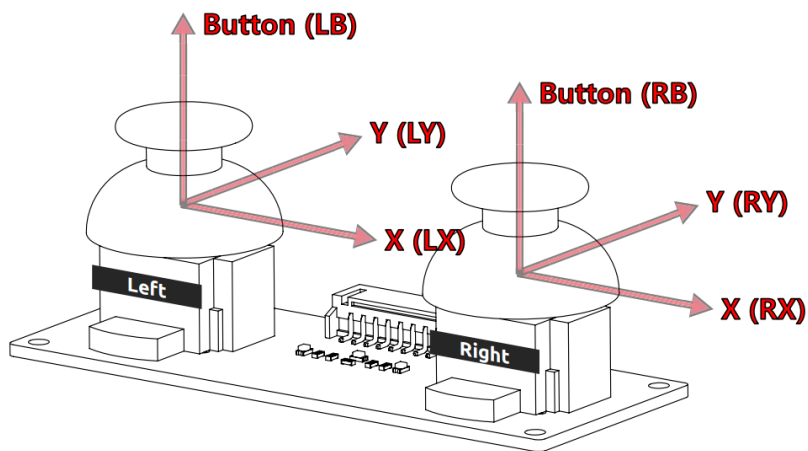
注解:

- 变量 HIGH 和 LOW 的取值参考双摇杆模块.
 - α , β and γ 表示 PiArm 手臂上三个舵机的旋转角度范围, 可参考: 角度模式.
-



第二步

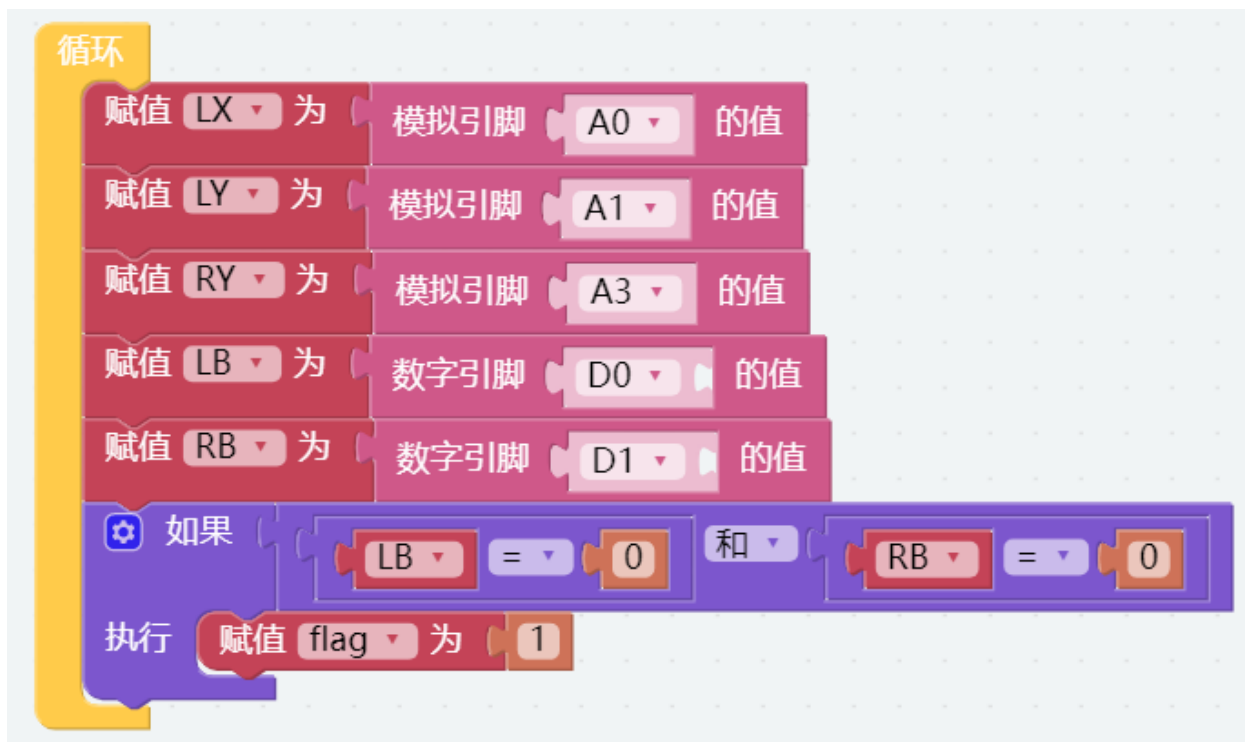
创建另外五个变量 (LX, LY, RY, LB, RB) 来分别读取双摇杆模块的 X、Y 轴和按下的值。





第三步

当 LB 和 RB 同时读取为 0，表示左右摇杆被按下，此时游戏开始，同时计时并设置 flag 为 1。



第四步

创建一个名为 [clip] 的函数来控制竖直夹。

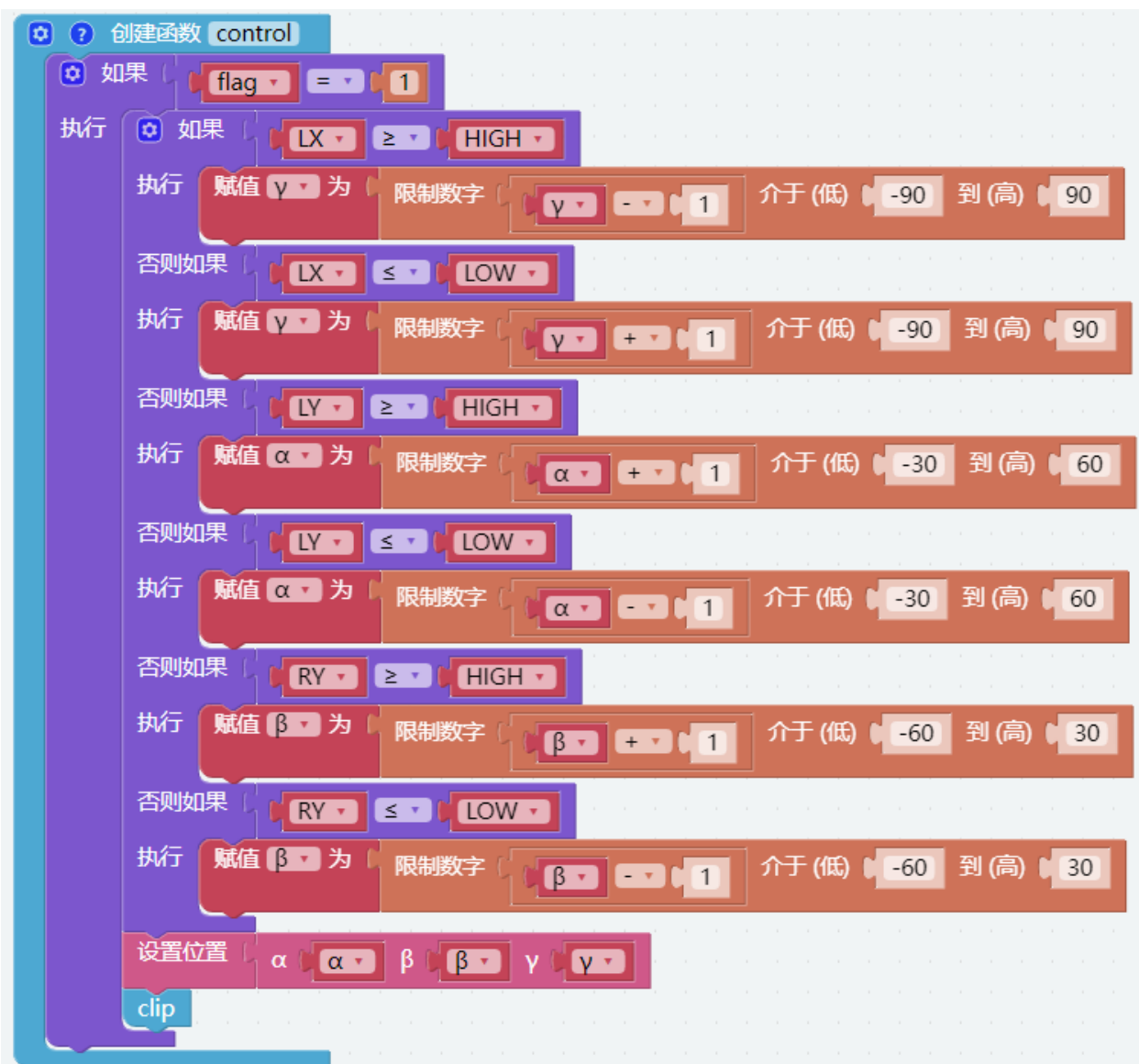
- 当按下左摇杆时，竖直夹会慢慢夹紧。
- 当按下右摇杆时，竖直夹会慢慢松开。



第五步

参考双摇杆模块创建一个函数 [control] 来设置 PiArm 的移动方向。

- 当 flag 为 1 时，表示游戏开始。这时候就可以开始控制 PiArm 了。
- 如果 左摇杆 (LX) 向右推动，则让 PiArm 向右转动。
- 如果 左摇杆 (LX) 向左推动，则让 PiArm 向左转动。
- 如果 左摇杆 (LY) 向前推动，则让 PiArm 向前伸长。
- 如果 左摇杆 (LY) 向后推动，则让 PiArm 向后缩回。
- 如果 右摇杆 (RX) 向前推动，则让 PiArm 向上抬高。
- 如果 右摇杆 (RY) 向后推动，则让 PiArm 向下降低。
- 竖直夹的控制代码也在这里被调用。这可以让您同时控制 PiArm 的手臂和竖直夹部分



第六步

将 [control] 函数块放入 [循环] 代码块中。



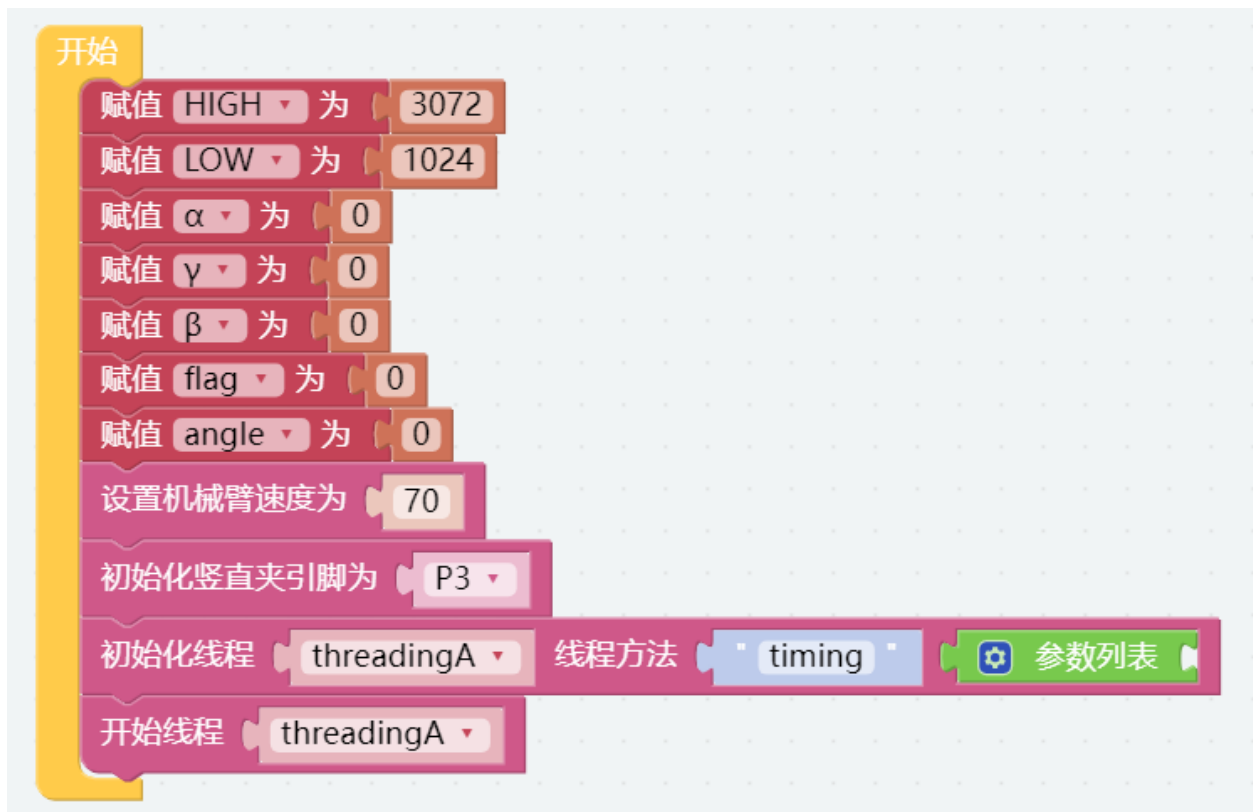
第七步

创建一个名为 [timing] 的函数以用于计时。游戏时间设置为 60 秒 (60000)，最后 3 秒会响起倒计时，让你知道时间快到了。



第八步

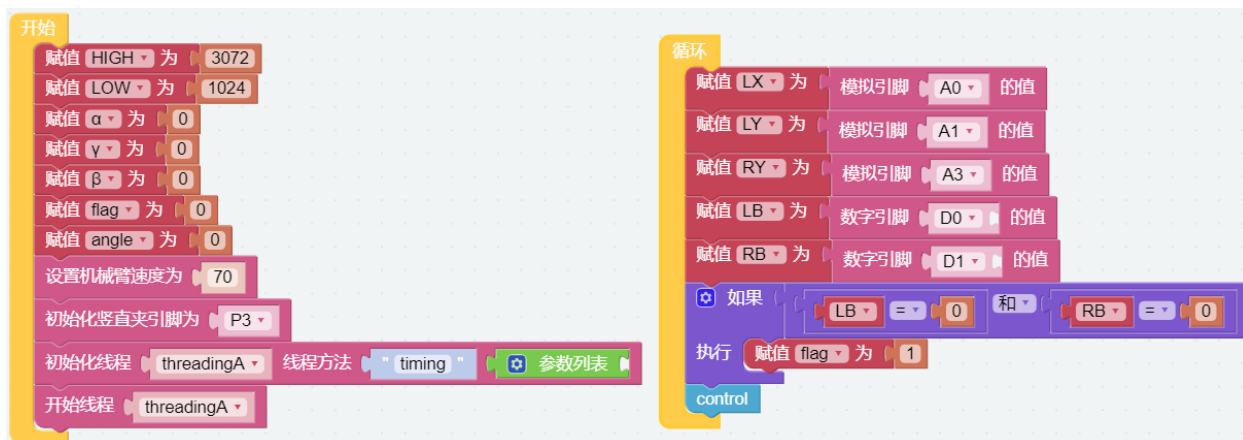
让 [timing] 函数在单独的线程中运行。这可以让您在记时的同时控制 PiArm。



完整代码如下：

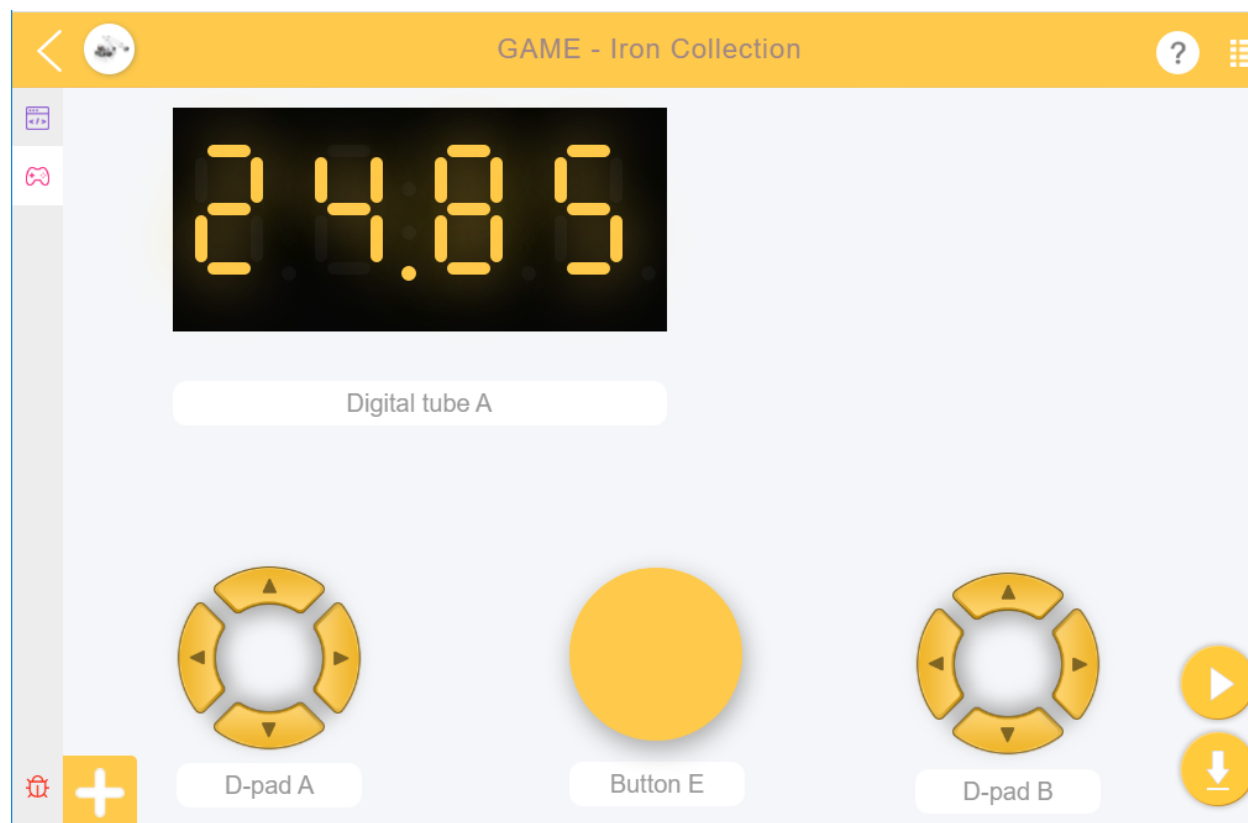
注解：您也可以在 Ezblock Studio 的 **示例** 页面找到同名的代码，直接点击运行或编辑查看代码块。





3.9 游戏 - 收集铁片

在这个项目中，准备好 3 种不同形状的铁片，PiArm 会随机说一种形状，你需要控制 PiArm 将相应形状的铁片放入各自的盒子里面。



3.9.1 编程

第一步

创建五个变量 (α , β , γ , flag, shape) 并为它们设置初始值。然后初始化 PiArm 的旋转速度和 电磁铁的引脚

注解:

- α , β and γ 表示 PiArm 手臂上三个舵机的旋转角度范围, 可参考: [角度模式](#).



第二步

从远程控制界面拖 2 个 D-pad 用来控制 PiArm, 一个按键来启动游戏, 以及一个数码管来显示时间。



第三步

创建一个叫做 [magnet] 的函数用来实现让 D-pad B 左右方向的按钮控制电磁铁的开和关。



第四步

创建一个叫做 [control] 函数用来实现让 D-pad A 和 D-pad B 的上下来控制 Arm of PiArm。

- 如果按下方向盘 A 的上按钮 (▲)，则让 PiArm 向前伸长。
- 如果按下方向盘 A 的下按钮 (▼)，则让 PiArm 向后缩回。
- 如果按下方向盘 A 的左按钮 (◀)，则让 PiArm 向左转动。
- 如果按下方向盘 A 的右按钮 (▶)，则让 PiArm 向右转动。
- 如果按下方向盘 B 的上按钮 (▲)，则让 PiArm 向上抬起。
- 如果按下方向盘 B 的下按钮 (▼)，则让 PiArm 向下降低。

注解:

- α , β and γ 表示 PiArm 手臂上三个舵机的旋转角度范围, 可参考: 角度模式.
- [限制数字... 介于 (低) ... 到 (高) ...]: 可以在数学类中找到, 用于设置一个变量的变化范围。
- [否则如果]: 用于条件判断的代码块, 可以点击设置图标将 [否则] 或 [否则如果] 拖拽到 [如果] 下方来创建多个条件判断。



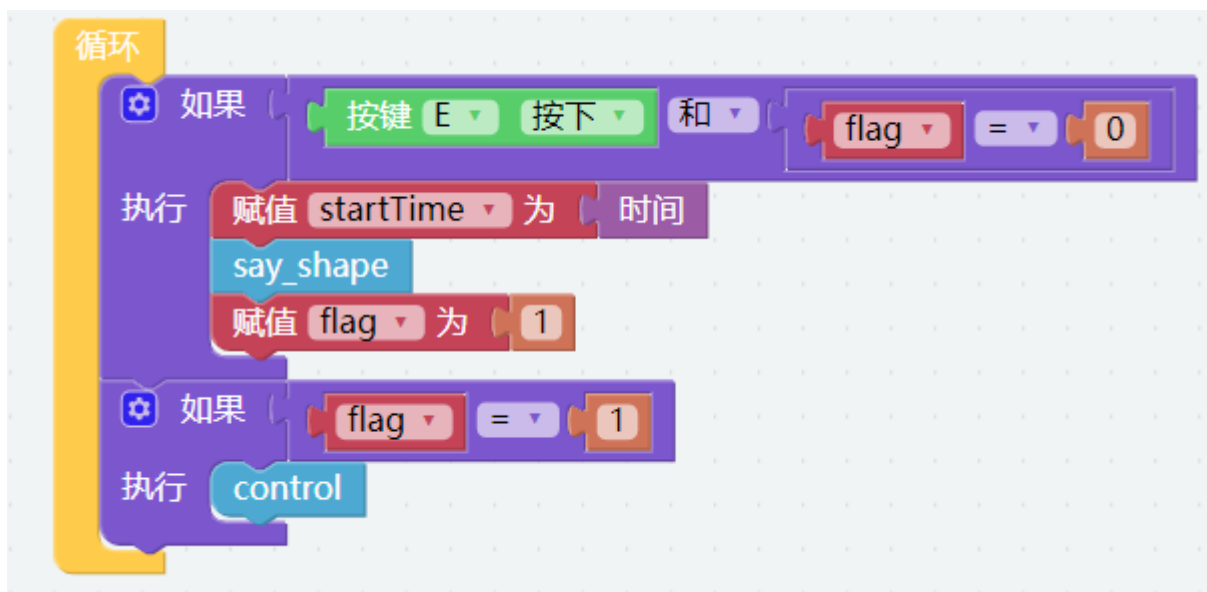
第五步

创建函数 [say_shape]，让 PiArm 随机说一个形状。



第六步

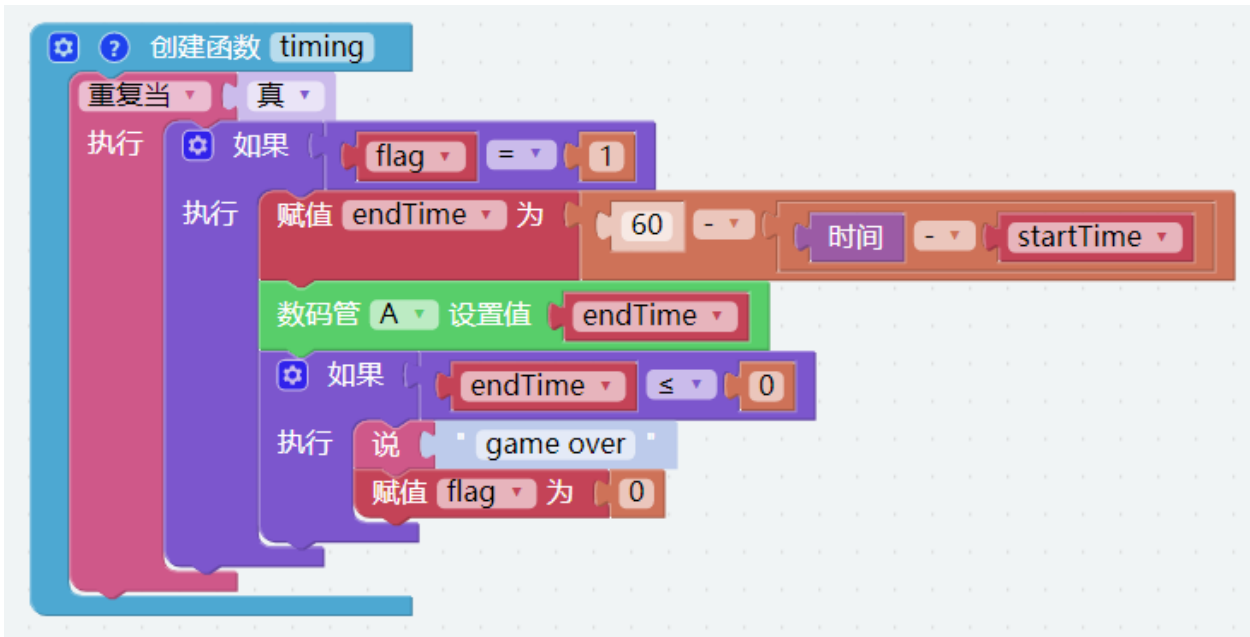
设置代码的主要流程：当按键 E 按下时，开始计时，此时 PiArm 会随机说一个形状。变量 flag 等于一时表示开始计时并可以控制 PiArm 了。



第七步

创建一个名为 [timing] 的函数以用于计时。游戏时间设定为 60 秒，时间到了之后，PiArm 将用语音播报游戏结束，并且你将无法再控制它。

这里的 [timing] 函数不同于上一个项目，我们在这里用到了 [时间] 模块来进行计时，在循环里面当判断出按键 E 被按下时，计时开始，[时间 - startTime] 代码块表示从计时开始到现在过去了多少秒。



第八步

让 [timing] 函数在单独的线程中运行。这可以让您在计时的同时控制 PiArm。



完整代码如下：



如果你想用 `python` 编程，那么你需要学习一些基本的 `Python` 编程技巧和树莓派的基础知识，请先根据 `Python` 快速指南配置树莓派。

4.1 Python 快速指南

本节教大家如何安装树莓派操作系统，配置树莓派 `wifi`，远程访问树莓派运行相应代码。

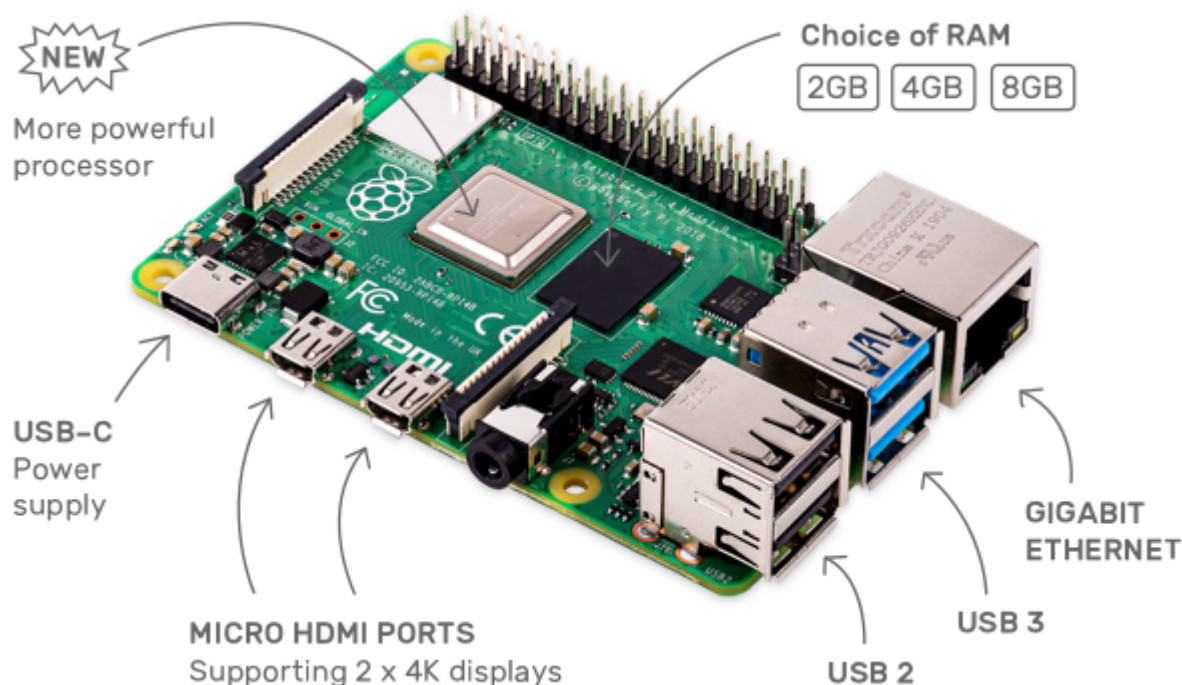
如果您熟悉树莓派并且可以成功打开命令行，那么您可以跳过前 3 部分，然后完成最后一部分。

4.1.1 我们需要什么？

所需组件

树莓派

树莓派是一款低成本、信用卡大小的计算机，可插入进入电脑显示器或电视，并使用标准键盘和鼠标。这是一款功能强大的小型设备，可让所有年龄段的人探索计算，并学习如何使用 `Scratch` 和 `Python` 等语言进行编程。



电源适配器

为了连接到电源插座，树莓派有一个微型 USB 端口（在许多手机上都可以找到）。您将需要一个提供至少 2.5 安培电流的电源。

微型 SD 卡

您的树莓派需要一张 Micro SD 卡来存储其所有文件和树莓派操作系统。您需要一张容量至少为 8 GB 的微型 SD 卡。

可选组件

屏幕

要查看树莓派的桌面环境，需要使用的屏幕可以是电视屏幕，也可以是电脑显示器。如果屏幕有内置扬声器，Pi 通过它们播放声音。

鼠标和键盘

使用屏幕时，还需要一个 USB 键盘和一个 USB 鼠标。

HDMI

树莓派有一个 HDMI 输出端口，与大多数现代电视和计算机显示器的 HDMI 端口兼容。如果您的屏幕只有 DVI 或 VGA 端口，则需要使用合适的转换线。

外壳

你可以把树莓派放在一个盒子里；通过这种方式，您可以保护您的设备。

声音或耳机

树莓派配备了一个 3.5mm 左右的音频接口，可以在你的屏幕没有内置扬声器或者没有屏幕操作的情况下使用。

4.1.2 安装操作系统

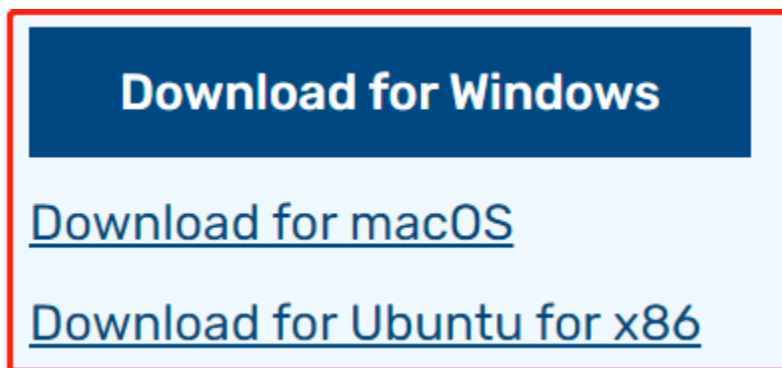
必需组件

任意树莓派	1 * 个人计算机
1 * 微型 SD 卡	

第 1 步

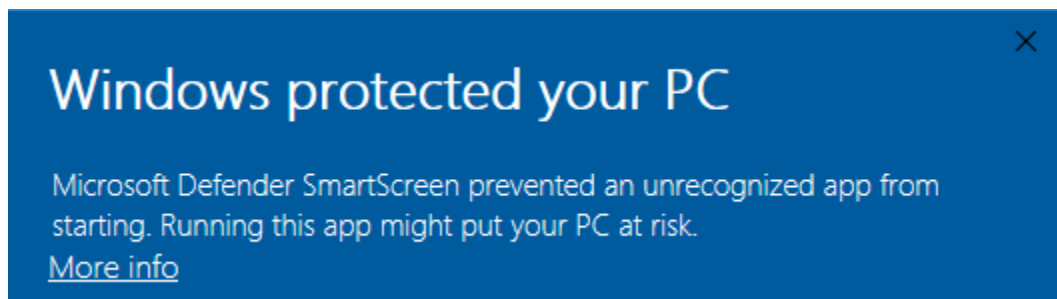
树莓派开发了一个图形 SD 卡写入工具，适用于 Mac OS、Ubuntu 18.04 和 Windows，对于大多数用户来说是最简单的选择，因为它会下载映像并将其自动安装到 SD 卡。

访问下载页面：<https://www.raspberrypi.org/software/>。单击与您的操作系统匹配的 Raspberry Pi Imager 链接，下载完成后，单击它以启动安装程序。



第 2 步

当您启动安装程序时，您的操作系统可能会尝试阻止您运行它。例如，在 Windows 上，我收到以下消息：如果出现此消息，请点击 **更多信息**，然后点击 **仍然运行**，然后按照说明安装 Raspberry Pi Imager。



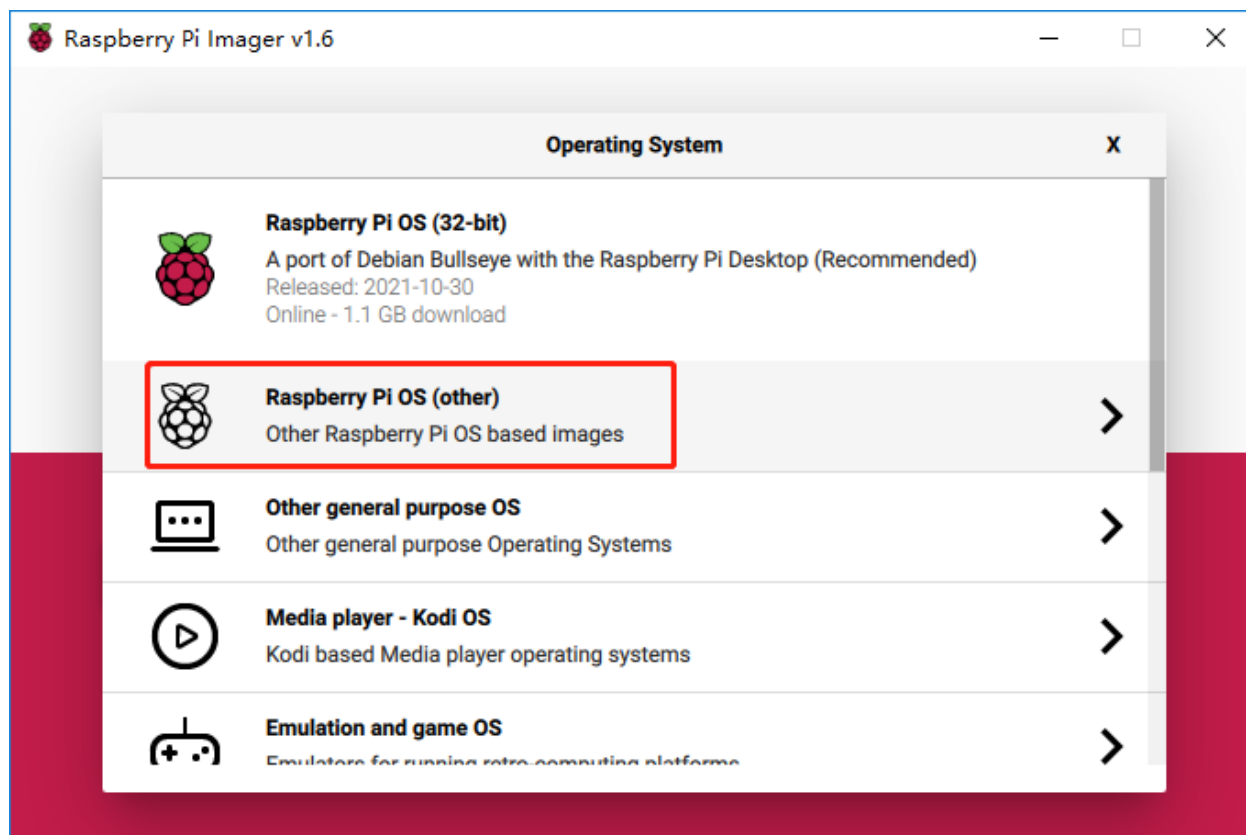
第 3 步

将 SD 卡插入计算机或笔记本电脑的 SD 卡插槽。

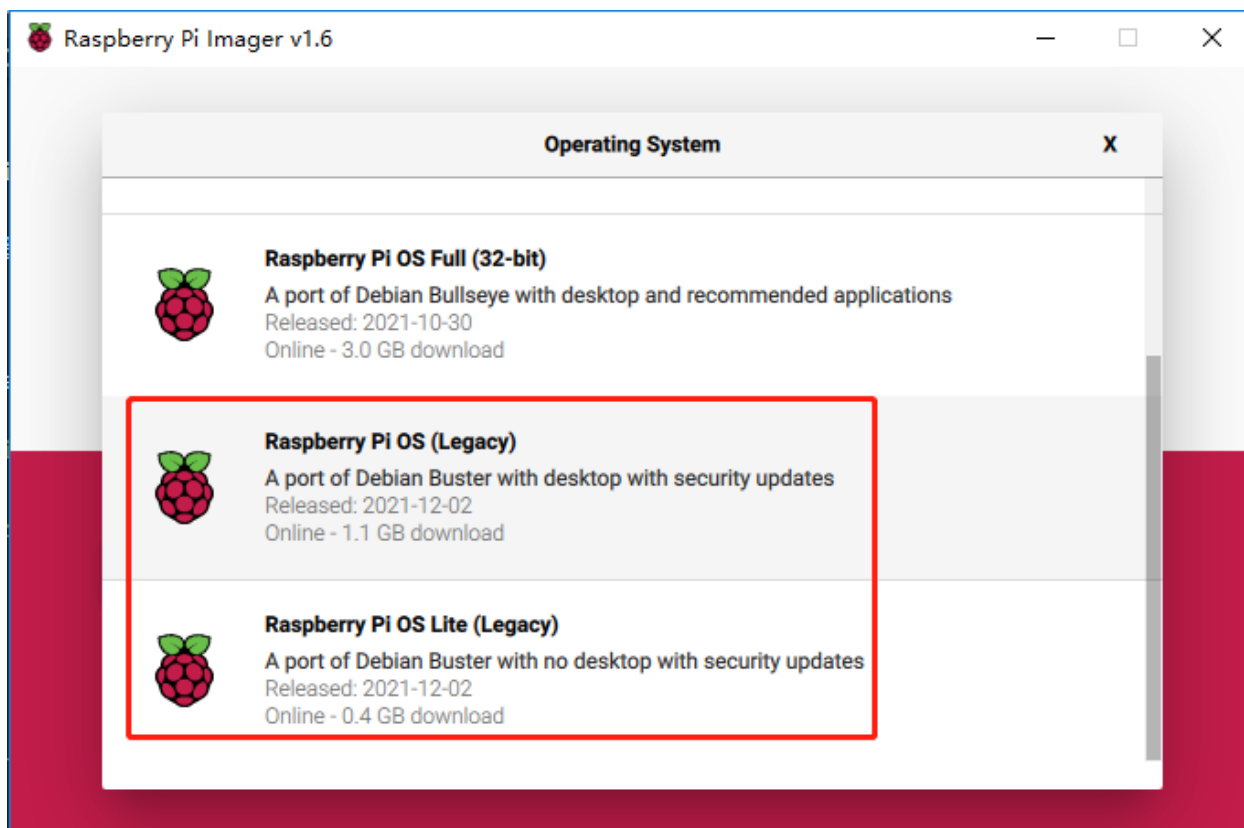
第 4 步

警告： Raspberry Pi OS 升级到 **Debian Bullseye** 后，会导致有些功能不能使用，建议还是继续使用 **Debian Buster** 版本。

在 Raspberry Pi Imager 中，点击 **CHOOSE OS -> Raspberry Pi OS(other)**。

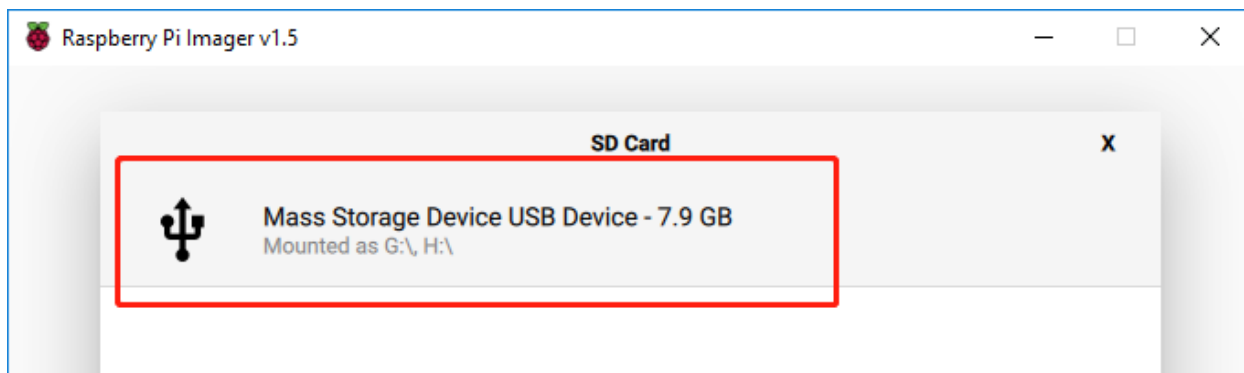


将新打开的页下拉到最后面，你会看到 **Raspberry Pi OS(Legacy)** 和 **Raspberry Pi OS Lite(Legacy)**，这 2 个是对 Debian Buster 安全更新，它们之间的区别是带不带桌面。建议安装 **Raspberry Pi OS(Legacy)**，这个带桌面的系统。



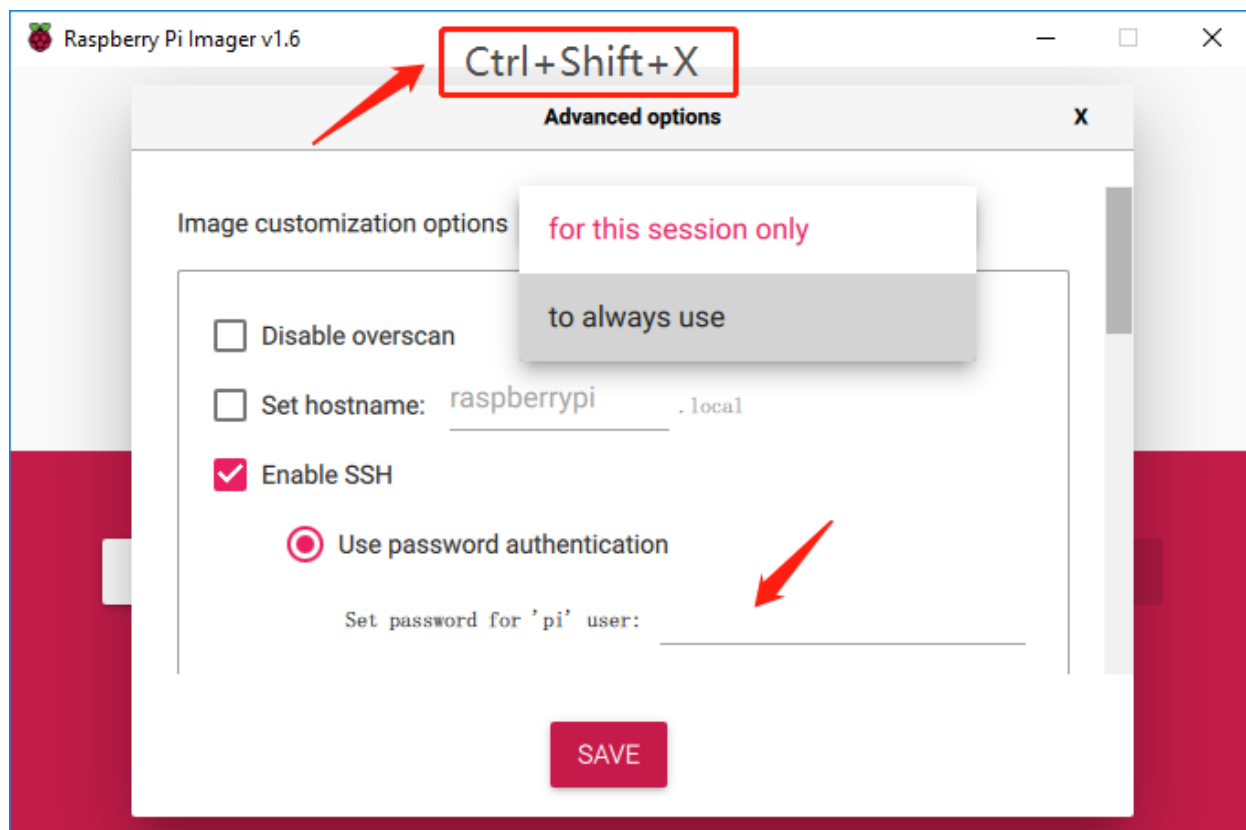
第 5 步

选择您正在使用的 SD 卡。



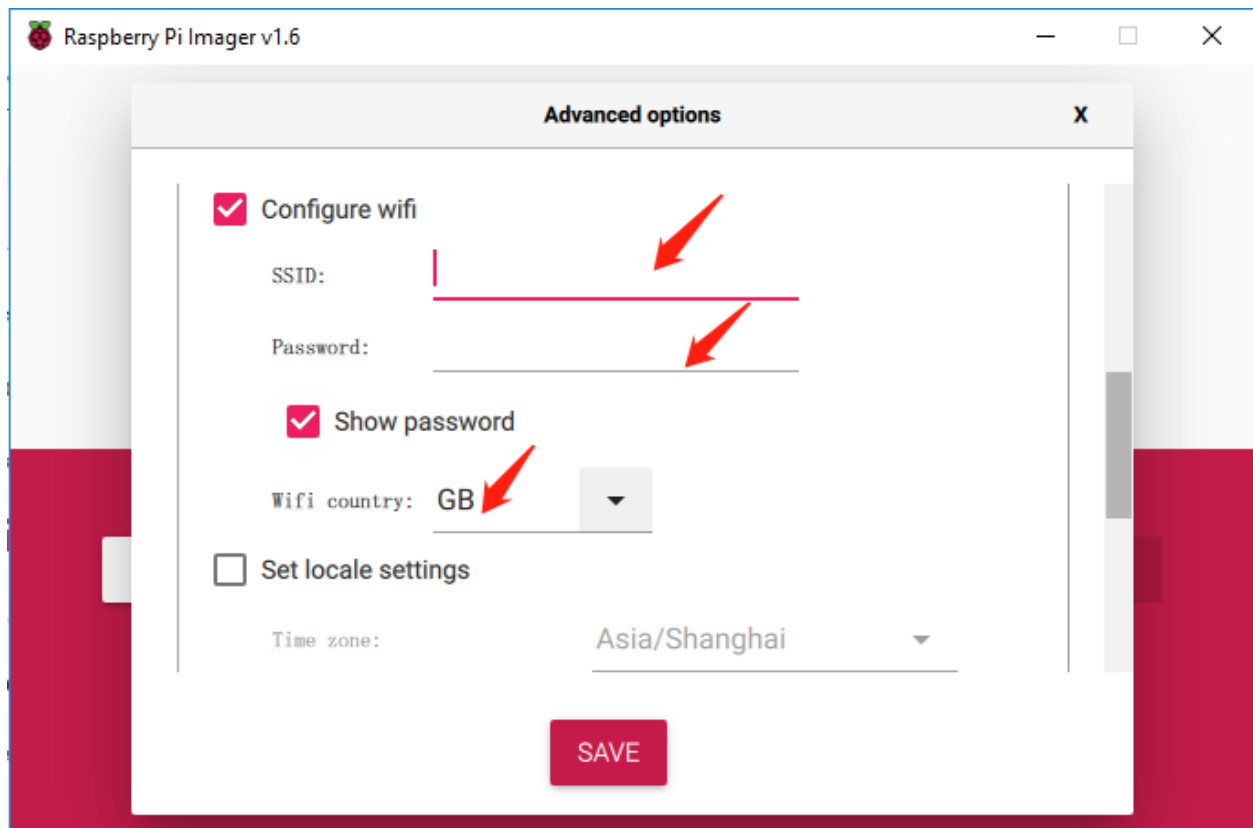
第 6 步

按 **Ctrl+Shift+X** 或者点击 **设置**按钮来打开 **高级选项**页面启用 SSH 和配置 wifi，这 2 项必须设置，其他取决于你的选择。您可以选择始终使用此图像自定义选项。



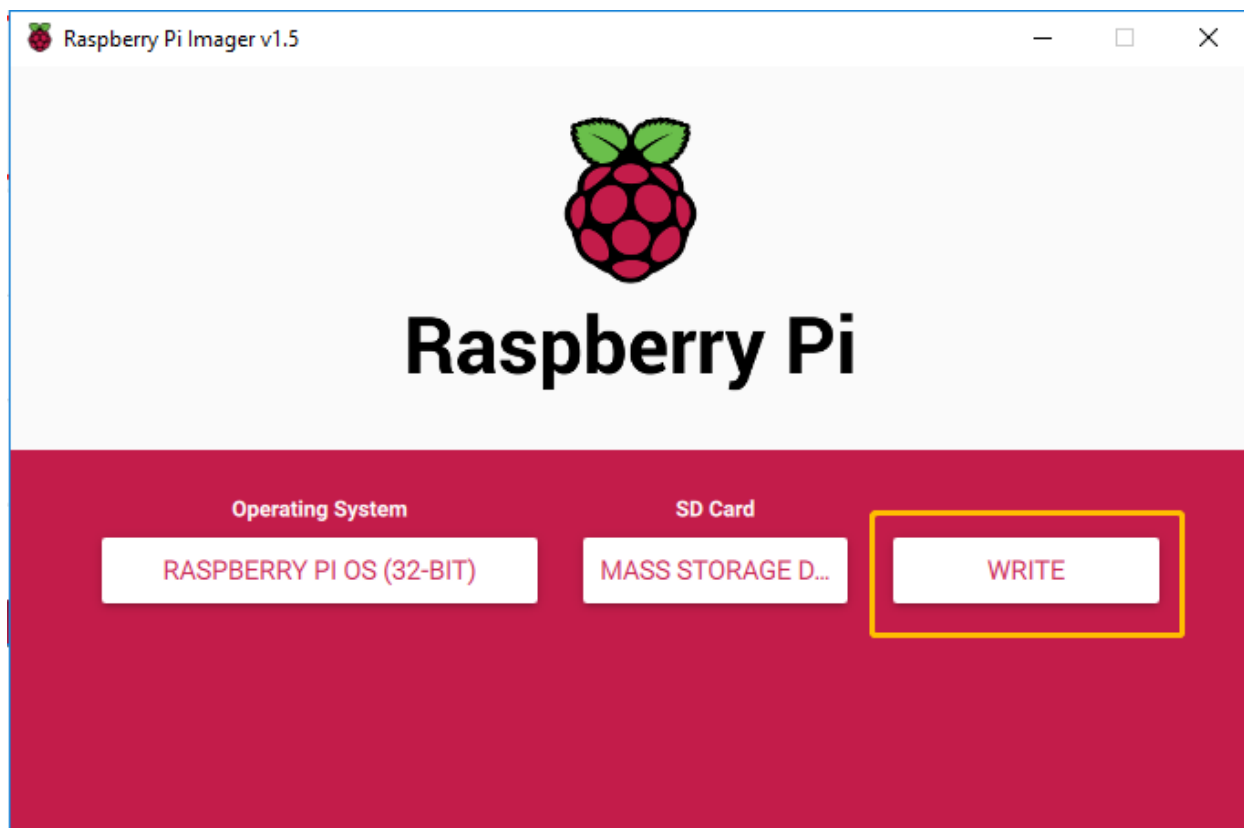
然后向下滚动以完成 wifi 配置并单击 **SAVE** 。

注解： **wifi country** 选择 CN。



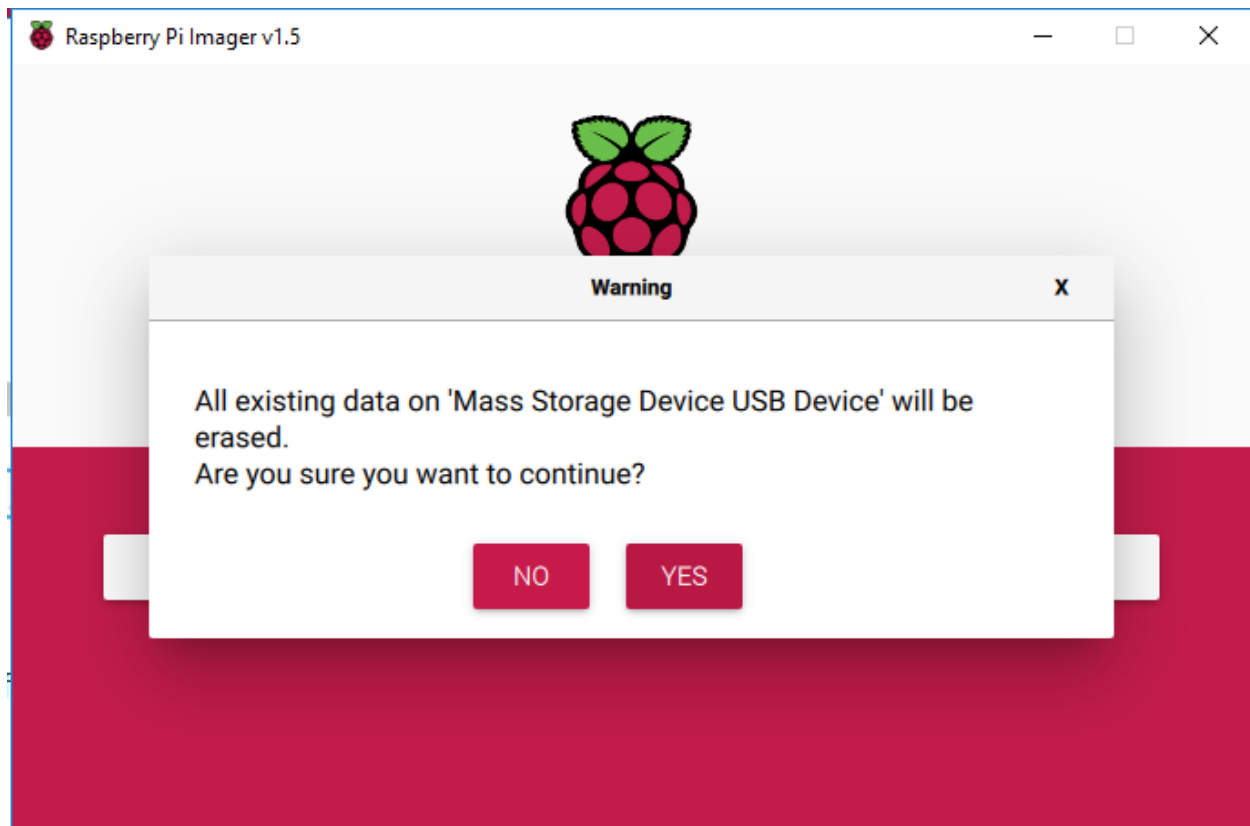
第 7 步

单击 **WRITE** 按钮。



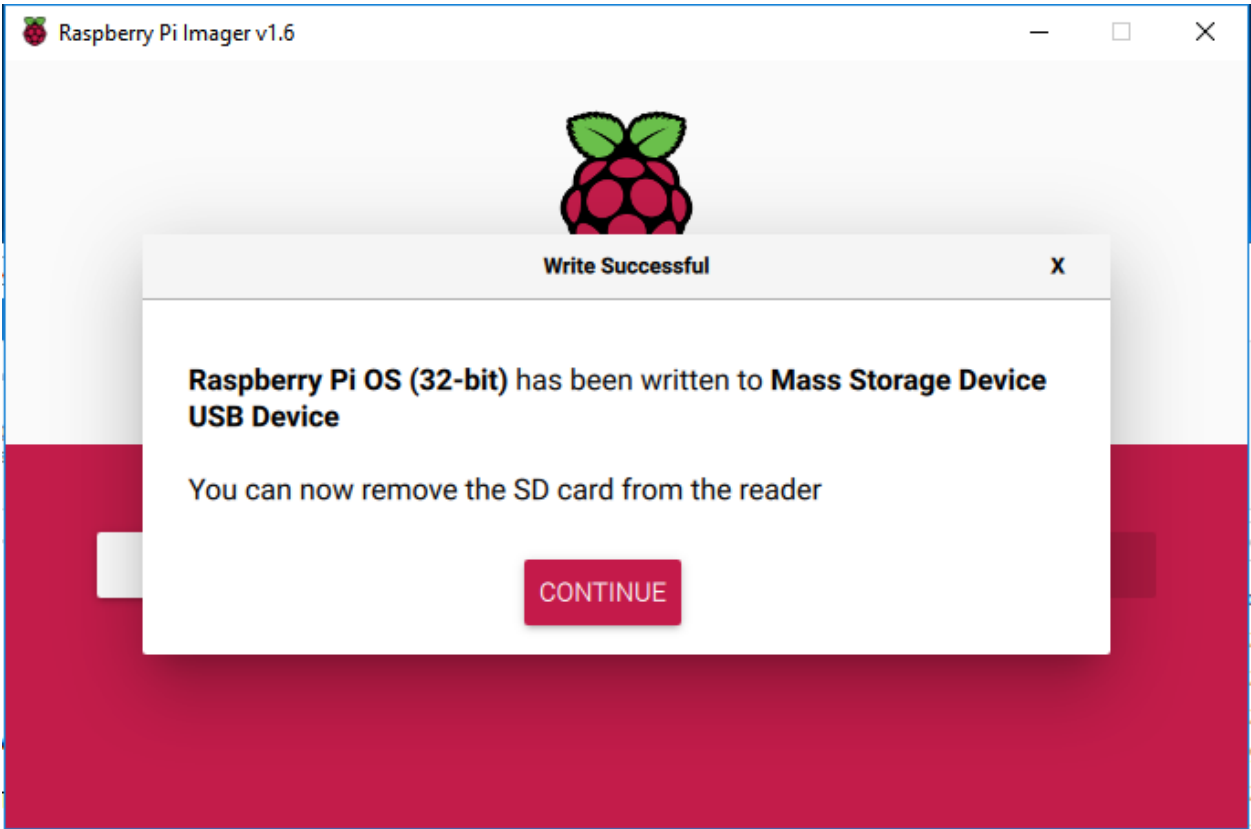
第 8 步

如果您的 SD 卡上当前有任何文件，您可能希望先备份这些文件以防止永久丢失它们。如果没有要备份的文件，请单击 **YES**。



第9步

等待一段时间后，会出现如下窗口，代表写入完成。



4.1.3 设置你的树莓派

如果你有屏幕

如果你有一个屏幕，你会很容易在屏幕上操作树莓派。

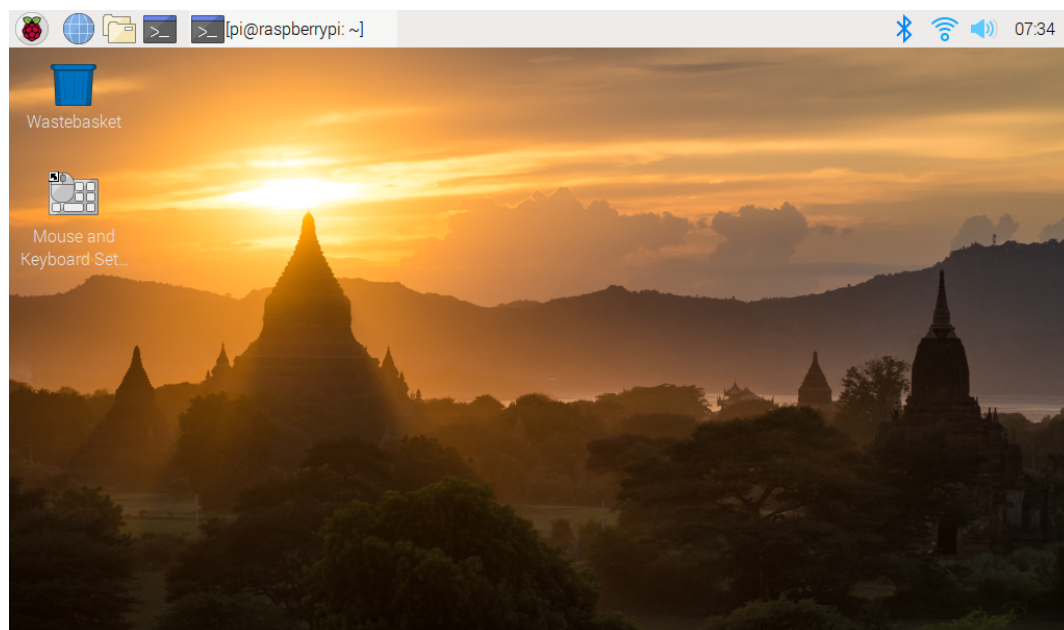
必需组件

任意树莓派	1 * 电源适配器
1 * Micro SD 卡	1 * 屏幕电源适配器
1 * HDMI 线	1 * 屏幕
1 * 鼠标	1 * 键盘

1. 将您在 Raspberry Pi OS 上设置的 SD 卡插入到树莓派底部的 micro SD 卡插槽中。
2. 插入鼠标和键盘。
3. 将屏幕连接到树莓派的 HDMI 端口，并确保您的屏幕已插入壁式插座并已打开。

注解：如果您使用的是树莓派 4，则需要将屏幕连接到 HDMI0（最靠近电源输入端口）。

4. 使用电源适配器为树莓派供电。几秒钟后，将显示 Raspberry Pi OS 桌面。



如果你没有屏幕

如果没有显示器，可以远程登录树莓派，但在此之前，您需要获取树莓派的 IP。

获取 IP 地址

树莓派连接 WIFI 后，我们需要获取它的 IP 地址。知道 IP 地址的方法有很多种，下面列出了其中的两种。

1. 通过路由器检查

如果您有权限登录路由器（如家庭网络），您可以在路由器的管理界面查看分配给树莓派的地址。

树莓派操作系统的默认主机名是 **raspberrypi**，你需要找到它。（如果你使用的是 ArchLinuxARM 系统，请找 **alarmpi**。）

2. 网段扫描

您还可以使用网络扫描来查找树莓派的 IP 地址。您可以应用软件，**Advanced IP scanner** 等。

扫描设置的 IP 范围，将显示所有已连接设备的名称。同样，树莓派操作系统的默认主机名是 **raspberrypi**，如果你没有修改过的话。

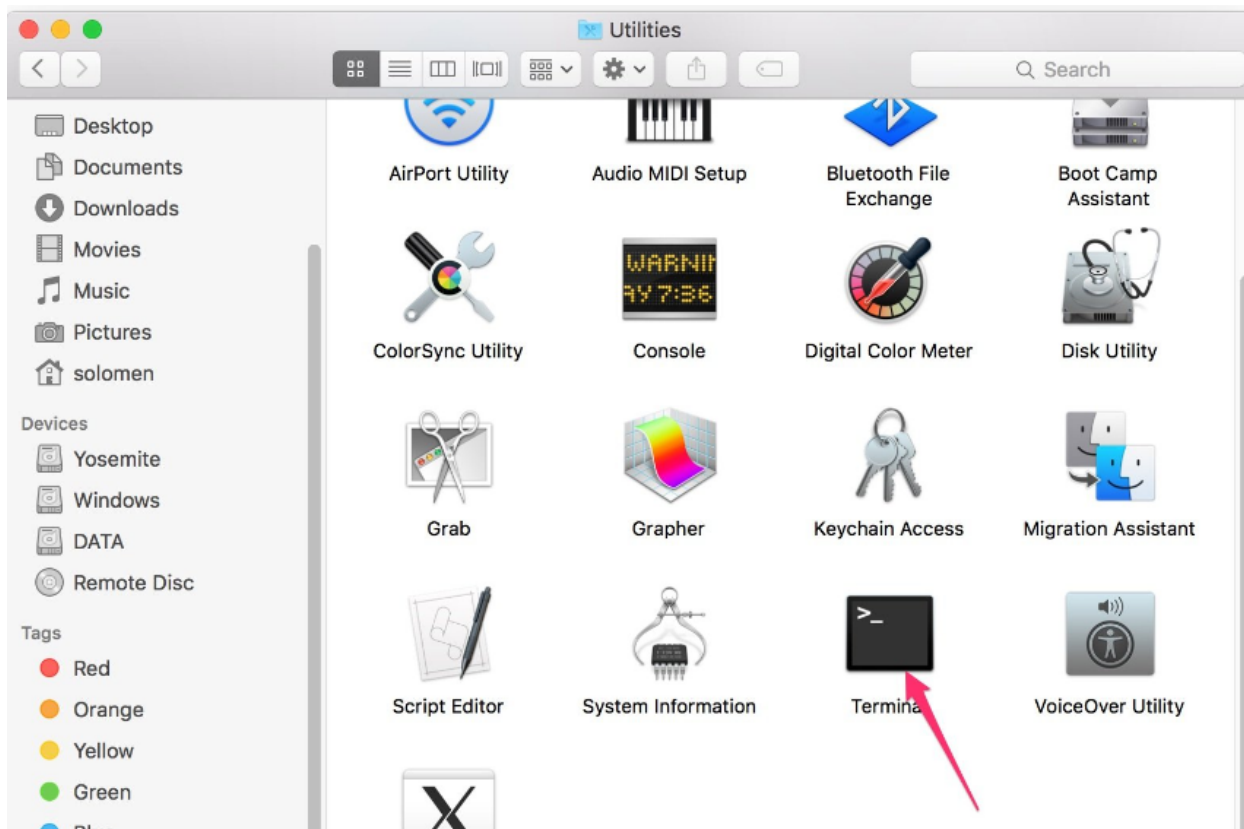
使用 SSH 远程控制

我们可以通过应用 SSH 打开树莓派的 Bash Shell。Bash 是 Linux 的标准默认 shell。Shell 本身是一个用 C 编写的程序，它是连接客户和 Unix/Linux 的桥梁。此外，它可以帮助完成大部分所需的工作。

适用于 Linux 或 Mac OS X 用户

第 1 步

进入 **Applications -> Utilities**，找到 **Terminal**，然后打开它。



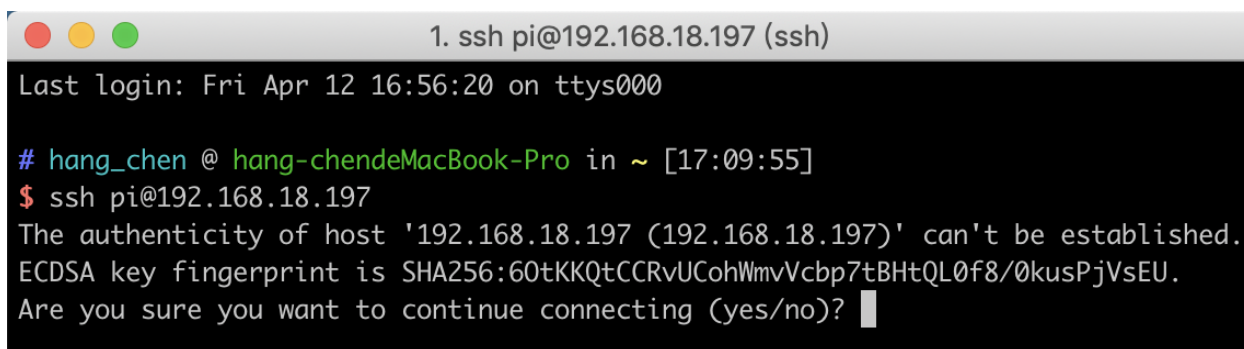
第 2 步

输入 `ssh pi@ip_address`。“pi” 是您的用户名，“ip_address” 是您的 IP 地址。例如：

```
ssh pi@192.168.18.197
```

第 3 步

输入“yes”。



第 4 步

输入密码，默认密码为 **raspberry**。

```

1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password: 

```

第5步

我们现在已连接树莓派，并准备进行下一步。

```

1. pi@raspberrypi: ~ (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:60tKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

pi@raspberrypi:~ $ 

```

注解： 当您输入密码时，字符不会相应显示在窗口中，这是正常的。您只需要输入正确的密码即可。

对于 Windows 用户

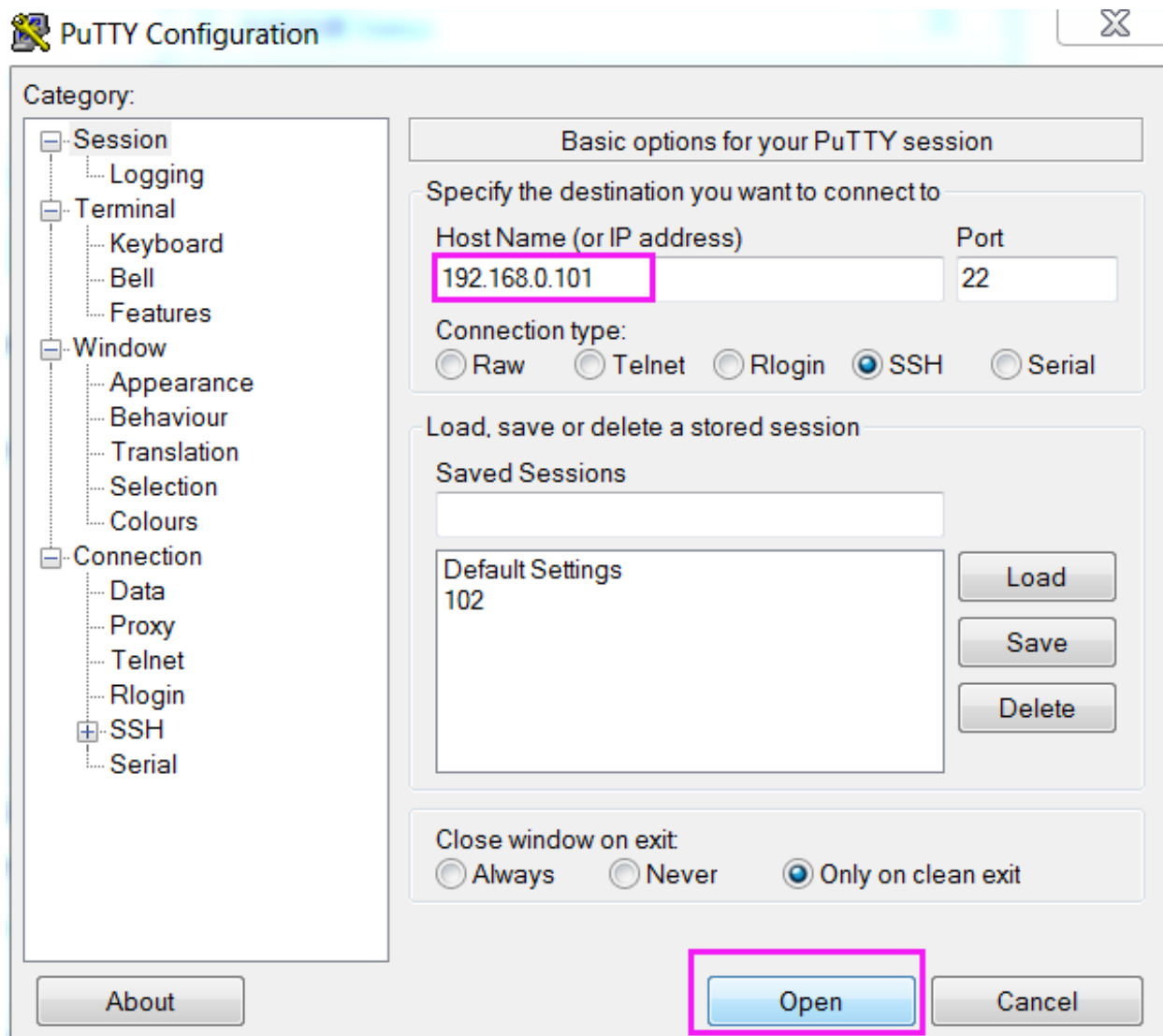
如果您是 Windows 用户，则可以通过某些软件的应用程序使用 SSH。在这里，我们推荐 **PuTTY**。

第 1 步

下载 PuTTY。

第 2 步

打开 PuTTY 并单击左侧树状结构上的 **Session**。在 **Host Name (or IP address)** 下的文本框中输入 RPi 的 IP 地址，**端口**下输入 **22**（默认为 22）。



第 3 步

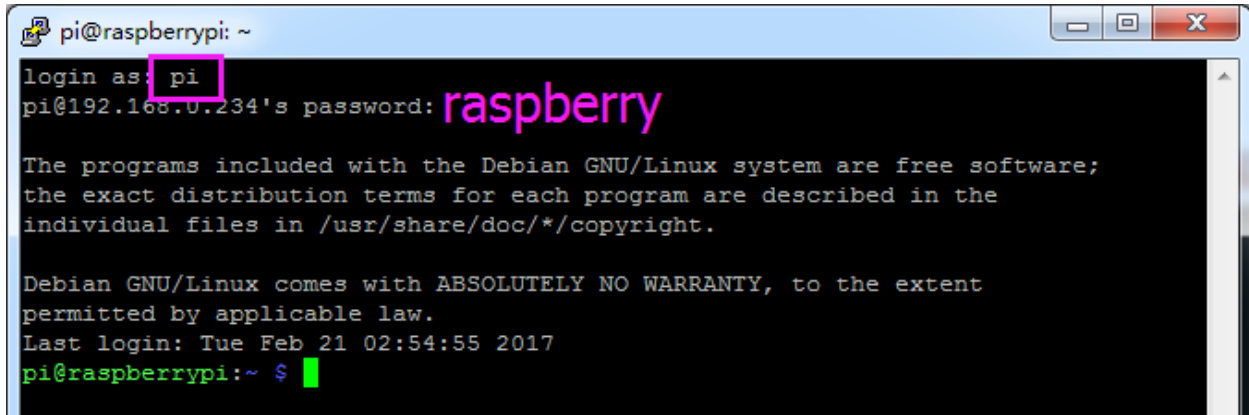
点击 **Open**。注意第一次用 IP 地址登录树莓派时，会提示安全提示。只需单击 **Yes**。

第四步

当 PuTTY 窗口提示 **login as:** 时，输入“pi”（树莓派的用户名）。提示 **password:** 时候，输入“raspberrypi”（默认密码）。

注解： 当您输入密码时，字符不会相应显示在窗口中，这是正常的。您只需要输入正确的密码即可。

如果 PuTTY 旁边出现 **inactive**，则表示连接已断开，需要重新连接。



```
pi@raspberrypi: ~
login as: pi
pi@192.168.0.234's password: raspberry

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 21 02:54:55 2017
pi@raspberrypi:~ $
```

第 5 步

在这里，我们连接了树莓派，是时候进行下一步了。

注解： 如果您对使用命令窗口控制树莓派不满意，还可以使用远程桌面功能，它可以帮助我们轻松管理树莓派中的文件。

有关如何执行此操作的详细信息，请参阅[远程桌面](#)。

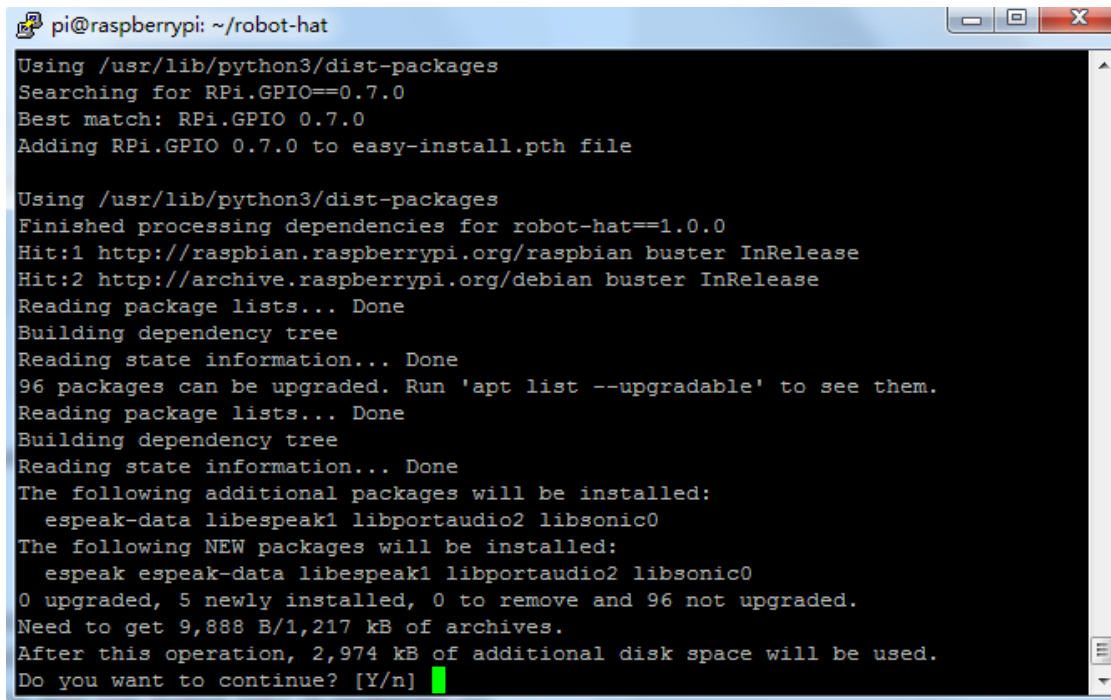
4.1.4 下载并运行代码

注解： 在下面的安装过程中，可能会由于网络问题导致失败，你需要参考[apt](#) 和 [pip](#) 更换国内源 来修改配置。

首先通过在命令行中使用 `git clone` 下载库文件 `robot-hat`。

```
cd /home/pi/
git clone https://gitee.com/sunfounder/robot-hat.git
cd robot-hat
sudo python3 setup.py install
```

注解： 运行 `setup.py` 将下载一些必要的组件。由于网络问题，你可能无法下载成功。你可能需要重新下载。在这种情况下，输入 `Y` 并按 `Enter`。



```
pi@raspberrypi: ~/robot-hat
Using /usr/lib/python3/dist-packages
Searching for RPi.GPIO==0.7.0
Best match: RPi.GPIO 0.7.0
Adding RPi.GPIO 0.7.0 to easy-install.pth file

Using /usr/lib/python3/dist-packages
Finished processing dependencies for robot-hat==1.0.0
Hit:1 http://raspbian.raspberrypi.org/raspbian buster InRelease
Hit:2 http://archive.raspberrypi.org/debian buster InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
96 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  espeak-data libespeak1 libportaudio2 libsonic0
The following NEW packages will be installed:
  espeak espeak-data libespeak1 libportaudio2 libsonic0
0 upgraded, 5 newly installed, 0 to remove and 96 not upgraded.
Need to get 9,888 B/1,217 kB of archives.
After this operation, 2,974 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

然后下载代码并安装 piarm 库。

```
cd /home/pi/
git clone -b 2.0.0 https://gitee.com/sunfounder/piarm.git
cd piarm
sudo python3 setup.py install
```

这一步需要一点时间，所以请耐心等待。

最后需要运行脚本 i2samp.sh 安装 i2s 功放所需的组件，否则它可能会没有声音。

```
cd /home/pi/piarm
sudo bash i2samp.sh
```

```

pi@raspberrypi: ~/pisloth
File "/usr/local/lib/python3.7/dist-packages/robot_hat-1.0.0-py3.7.egg/robot_hat/robot.py", line 91, in servo_move
    time.sleep(step_delay)
KeyboardInterrupt
pi@raspberrypi:~/pisloth/examples $ cd ..
pi@raspberrypi:~/pisloth $ sudo bash i2samp.sh
Support for your operating system is experimental. Please visit
forums.adafruit.com if you experience issues with this product.

This script will install everything needed to use
i2s amplifier

--- Warning ---

Always be careful when running scripts and commands
copied from the internet. Ensure they are from a
trusted source.

If you want to see what this script does before
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N] █

```

输入 y 并按 Enter 继续运行脚本。

```

pi@raspberrypi: ~/pisloth
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap

Commenting out Blacklist entry in
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] █

```

输入 y 并按 Enter 让 /dev/zero 在后台运行。

```
pi@raspberrypi: ~/pisloth
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] y

Created symlink /etc/systemd/system/multi-user.target.wants/aplay.service → /etc
/systemd/system/aplay.service.

All done!

Enjoy your new i2s amplifier!

Some changes made to your system require
your computer to reboot to take effect.

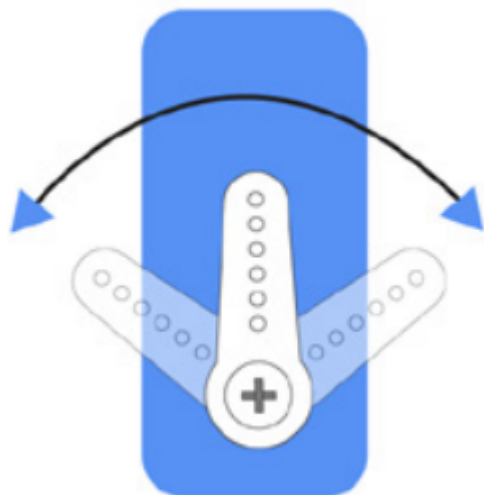
Would you like to reboot now? [y/N] █
```

输入 y 并按 Enter 重新启动机器。

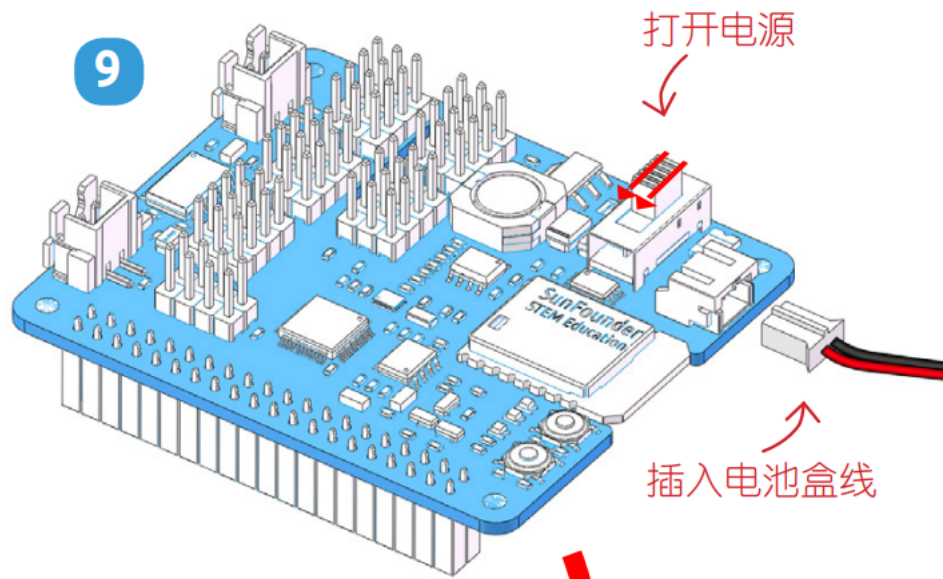
注解： 如果重新启动后没有声音，你可能需要多次运行 i2samp.sh 脚本。

4.1.5 舵机调零

1. 为确保舵机已正确设置为 0° ，首先将摇臂轻轻插入舵机轴，然后将摇臂轻轻旋转到不同的角度。



2. 按照组装折页上的提示，插入电池盒线，将电源开关拨向 ON 的位置。等待 1-2 分钟，会有声音提示树莓派启动成功。

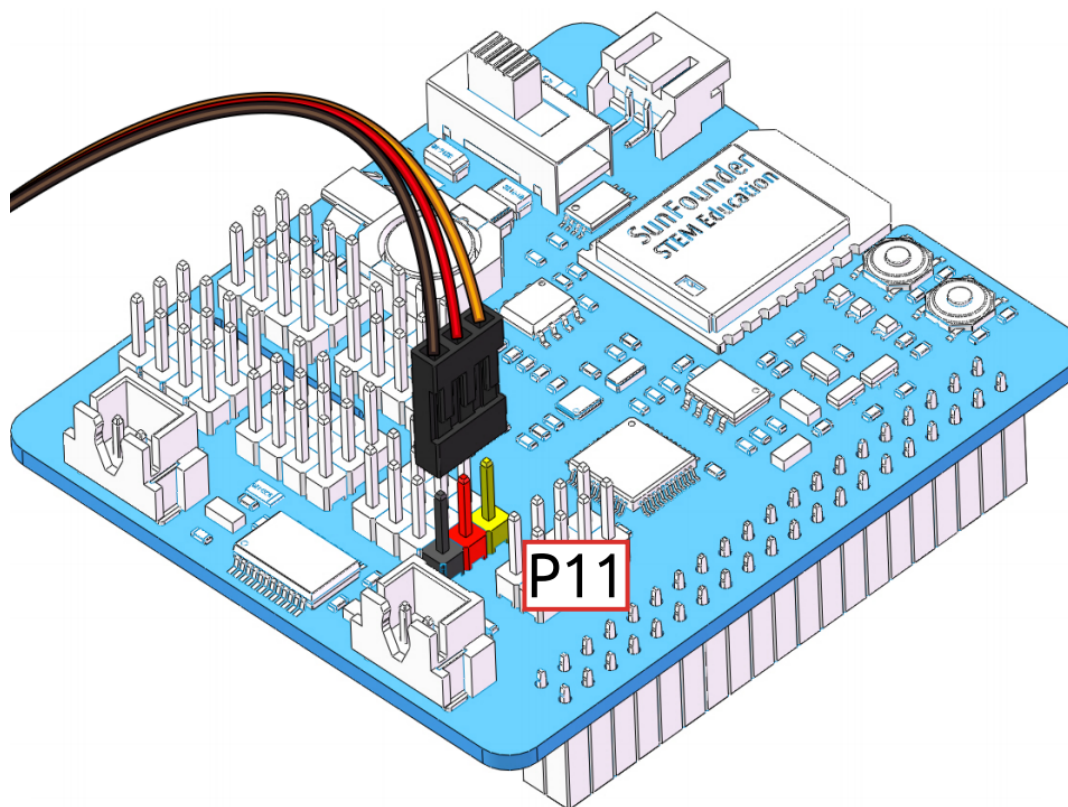


3. 现在，运行 examples/ 文件夹中的 servo_zeroing.py .

```
cd /home/pi/piarm/examples
sudo python3 servo_zeroing.py
```

注解：如果报错，请尝试重新启用树莓派的 I2C 端口，请参阅：[I2C 配置](#)。

4. 接下来，将舵机线插入 P11 端口，如下所示：



5. 此时你会看到舵机臂转动到特定的位置 (0°)。如果伺服臂没有返回到 0° ，请按 **RST** 按钮重新启动 Robot HAT。
6. 现在你就可以按照组装折页上的指示继续安装。

注解:

- 在用舵机螺丝固定该舵机前，不要拔出该舵机线，可在固定完之后拔出。
 - 不要在上电情况下随意转动舵机以免损坏；如果舵机轴插入的角度不对，需把舵机拔出再重新插入。
 - 在组装每个舵机之前，都需要将舵机电缆插入 P11 并打开电源以将它的角度设置为 0° 。
-

组装完成后，你可以尝试运行下面的项目。

4.2 组装和测试 3 个臂端工具

这是第一个程序，也是你必须看到的程序。

在这个项目中，你将学习如何组装和使用 PiArm 的 3 个臂端工具。

4.2.1 铲斗

运行代码

```
cd /home/pi/piarm/examples
sudo python3 shovel.py
```

运行代码后，你将看到铲斗来回移动。但你需要先组装铲斗。

注解：如果你发现舵机为 0 时，铲斗不是垂直向下的，需要松开舵机螺钉，再重新将铲斗以垂直向下的角度安装一遍。

代码

```
from robot_hat import Robot, Servo, PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1, 2, 3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0, 0, 0])

if __name__ == "__main__":
    while True:
        arm.set_bucket(-50)
        sleep(1)
        arm.set_bucket(90)
        sleep(1)
```

它是怎么工作的

```
from robot_hat import Robot, Servo, PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm
```

- 首先，从 `robot_hat` 中导入 `Robot`, `servo`, `PWM` 等类。
- 从 `robot_hat.utils` 模块中导入 `reset_mcu` 类，用来复位 MCU，以免程序间的冲突，导致通信错误。
- 从 `time` 模块中导入 `sleep` 类，用来实现延时的功能，单位：秒。
- 从 `piarm` 模块中导入 `PiArm` 类别，这是用来控制 `PiArm`。

```
reset_mcu()
sleep(0.01)
arm = PiArm([1, 2, 3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0, 0, 0])
```

先初始化 MCU，然后初始化 `PiArm` 的各个舵机连接引脚以及铲斗的连接引脚。

- `PiArm()`：初始化手臂上的 3 个舵机引脚。
- `bucket_init()`：设置 `bucket` 的引脚。

- `set_offset()`: 设置 arm 上的 3 个舵机的偏移值。

```
while True:
    arm.set_bucket(-50)
    sleep(1)
    arm.set_bucket(90)
    sleep(1)
```

这段代码是用来让铲斗在-50 和 90 度之间来回移动，时间间隔为 1 秒。

- `set_bucket()`: 用来控制铲斗的转动角度。

4.2.2 竖直夹

运行代码

```
cd /home/pi/piarm/examples
sudo python3 clip.py
```

运行代码后，你将看到竖直夹重复的打开/闭合。但你需要先组装[竖直夹](#)。

注解: 如果你发现竖直夹在 90° 时，不是垂直向下并合拢。需要将竖直夹的固定螺钉取下，重新再安装一遍。

代码

```
from robot_hat import Robot, Servo, PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1, 2, 3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0, 0, 0])

if __name__ == "__main__":
    while True:
        arm.set_hanging_clip(-50)
        sleep(1)
        arm.set_hanging_clip(90)
        sleep(1)
```

- `hanging_clip_init()`: 用来初始化竖直夹的引脚。
- `set_hanging_clip()`: 用来设置竖直夹的转动角度。

4.2.3 电磁铁

运行代码

```
cd /home/pi/piarm/examples
sudo python3 electromagnet.py
```

运行代码后，你会发现，**电磁铁**每秒钟都会通电（电磁铁上的 LED（D2）亮起，表明它通电了，这时可以用铁吸附一些材料。）。但你需要先组装**电磁铁**。

代码

```
from robot_hat import Robot, Servo, PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1, 2, 3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0, 0, 0])

if __name__ == "__main__":
    while True:
        arm.set_electromagnet('on')
        sleep(1)
        arm.set_electromagnet('off')
        sleep(1)
```

- `electromagnet_init()`：用来初始化电磁铁的连接。
- `set_electromagnet()`：用来控制电磁铁的开/关。

4.3 音效

在本例中，我们使用 PiArm（准确地说是 Robot HAT）的音效。它由三部分组成，分别是 Muisc、Sound、Text to Speech。

安装 i2samp

在使用该功能之前，请先激活扬声器，使其启用并可以发出声音。

运行 `i2samp.sh`，此脚本将安装使用 i2s 放大器所需的一切。

```
cd /home/pi/piarm/
sudo bash i2samp.sh
```

会有几个提示要求确认请求。用 Y 响应所有提示。对树莓派系统进行更改后，需要重新启动计算机才能使这些更改生效。

重新启动后，`i2samp.sh` 再次运行脚本以测试放大器。如果扬声器成功播放声音，则配置完成。

运行代码

```
cd /home/pi/piarm/examples
sudo python3 sound_effect.py
```

代码运行后，PiArm 会开始播放背景音乐，同时播放音效和说“timing begins”，“three”，“two”，“one”，“Stop music”。

代码

```
from robot_hat import Music,TTS
from time import sleep

m = Music()
t = TTS()

def sound():
    song = './sounds/sign.wav'
    m.music_set_volume(40)
    m.sound_play(song)

def background_music():
    music = './musics/sports-Ahjay_Stelino.mp3'
    m.music_set_volume(50)
    m.background_music(music)

def tts():
    t.say("timing begins")
    sleep(1)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("Stop music")
    sleep(1)

if __name__ == "__main__":
    background_music()
    sleep(10)
    #sound()
    #tts()
    while True:
        #background_music()
        sound()
        tts()
```

它是如何工作？

这个代码很简单，创建了3个函数 sound()，background() 和 tts()，然后分别调用它们来让 PiArm 播放音乐和说话。

```
def sound():
    song = './sounds/sign.wav'
    m.music_set_volume(40)
    m.sound_play(song)
```

以 40% 的音量播放音效 ./sounds/sign.wav。

- music_set_volume(): 设置音量，范围是 0%-100%。
- sound_play(): 播放特定路径下音效。

```
def background_music():  
    music = './musics/sports-Ahjay_Stelino.mp3'  
    m.music_set_volume(50)  
    m.background_music(music)
```

以 50% 的音量播放背景音乐 ./musics/sports-Ahjay_Stelino.mp3。

- background_music(): 播放特定路径下的背景音乐。

```
def tts():  
    t.say("timing begins")  
    sleep(1)  
    t.say("three")  
    sleep(1)  
    t.say("two")  
    sleep(1)  
    t.say("one")  
    sleep(1)  
    t.say("Stop music")  
    sleep(1)
```

写文本到 PiArm，让它说话。

- say(): 在括号中写入字符或字符串，就能让 PiArm 将它们说出来。

4.4 双摇杆模块

我们可以分两部分控制 PiArm，手臂和臂端工具。在第一个项目中，您已经学习了如何组装和测试 PiArm 的 3 个臂端工具。在这个项目中，我们将使用套件附带的双摇杆模块来控制 PiArm 的手臂。

- 铲斗 - 摇杆控制
- 竖直夹 - 摇杆控制
- 电磁铁 - 摇杆控制



4.4.1 铲斗 - 摇杆控制

```
cd /home/pi/piarm/examples
sudo python3 joystick_module1.py
```

代码运行后，你就能用拨动左右摇杆来控制 PiArm 的手臂的转动，分别按下左右摇杆来控制铲斗的开/合。但你需要先将铲斗 安装到 PiArm 上。

代码

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])
```

(下页继续)

(续上页)

```

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True
    if leftJoystick.read_status() == "pressed":
        bucket += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        bucket -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.
↪component_staus))

if __name__ == "__main__":
    while True:
        _angles_control()
        sleep(0.01)

```

它是如何工作的？

```

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

```

定义左右摇杆的 X,Y 和 Z 的引脚连接。

```

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if leftJoystick.read_status() == "up":

```

(下页继续)

(续上页)

```

        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True
    if leftJoystick.read_status() == "pressed":
        bucket += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        bucket -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.
↪component_status))

```

在这个代码中，创建了 `_angles_control()` 函数用来控制 PiArm。

- alpha,beta 和 gamma 分别指的是手臂上的 3 个舵机的角度，参考：[角度模式](#)。
- 左摇杆向上拨动，alpha 增加，让手臂向前伸。
- 左摇杆向下拨动，alpha 减小，让手臂向里缩。
- 左摇杆向左拨动，gamma 增加，让手臂向左转动。
- 左摇杆向右拨动，gamma 减小，让手臂向右转动。
- 右摇杆向上拨动，beta 增加，让手臂向上。
- 右摇杆向下拨动，beta 减小，让手臂向下。
- 最后，分别用左右摇杆的按键来控制铲斗的角度。

4.4.2 竖直夹 - 摇杆控制

运行代码

```

cd /home/pi/piarm/examples
sudo python3 joystick_module2.py

```

代码运行后，你就能用拨动左右摇杆来控制 PiArm 的手臂的转动，分别按下左右摇杆来控制竖直夹的开/合。但你需要先将[竖直夹](#)安装到 PiArm 上。

代码

```

from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    clip = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True

    if leftJoystick.read_status() == "pressed":
        clip += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        clip -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_hanging_clip(clip)
        print('servo angles: %s , clip angle: %s'%(arm.servo_positions,arm.component_
↪staus))

if __name__ == "__main__":
    while True:
        _angles_control()
        sleep(0.01)

```


在这个代码中，创建了 `_angles_control()` 函数用来控制 PiArm。

- `alpha`, `beta` 和 `gamma` 分别指的是手臂上的 3 个舵机的角度，参考：[角度模式](#)。
- 左摇杆向上拨动，`alpha` 增加，让手臂向前伸。
- 左摇杆向下拨动，`alpha` 减小，让手臂向里缩。
- 左摇杆向左拨动，`gamma` 增加，让手臂向左转动。
- 左摇杆向右拨动，`gamma` 减小，让手臂向右转动。
- 右摇杆向上拨动，`beta` 增加，让手臂向上。
- 右摇杆向下拨动，`beta` 减小，让手臂向下。
- 最后，分别用左右摇杆的按键来控制竖直夹的角度。

4.4.3 电磁铁 - 摇杆控制

运行代码

```
cd /home/pi/piarm/examples
sudo python3 joystick_module3.py
```

代码运行后，你就能用拨动左右摇杆来控制 PiArm 的手臂的转动，分别按下左右摇杆来控制电磁铁的开/关。但你需要先将电磁铁安装到 PiArm 上。

代码

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    status = ""

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
```

(下页继续)

(续上页)

```

if leftJoystick.read_status() == "left":
    gamma += 1
    flag = True
elif leftJoystick.read_status() == "right":
    gamma -= 1
    flag = True
if rightJoystick.read_status() == "up":
    beta += 1
    flag = True
elif rightJoystick.read_status() == "down":
    beta -= 1
    flag = True
if leftJoystick.read_status() == "pressed":
    arm.set_electromagnet('on')
    status = "electromagnet is on"
elif rightJoystick.read_status() == "pressed":
    arm.set_electromagnet('off')
    status = "electromagnet is off"

if flag == True:
    arm.set_angle([alpha,beta,gamma])
    print('servo angles: %s , electromagnet status: %s'%(arm.servo_positions,
→status))

if __name__ == "__main__":
    while True:
        _angles_control()
        sleep(0.01)

```

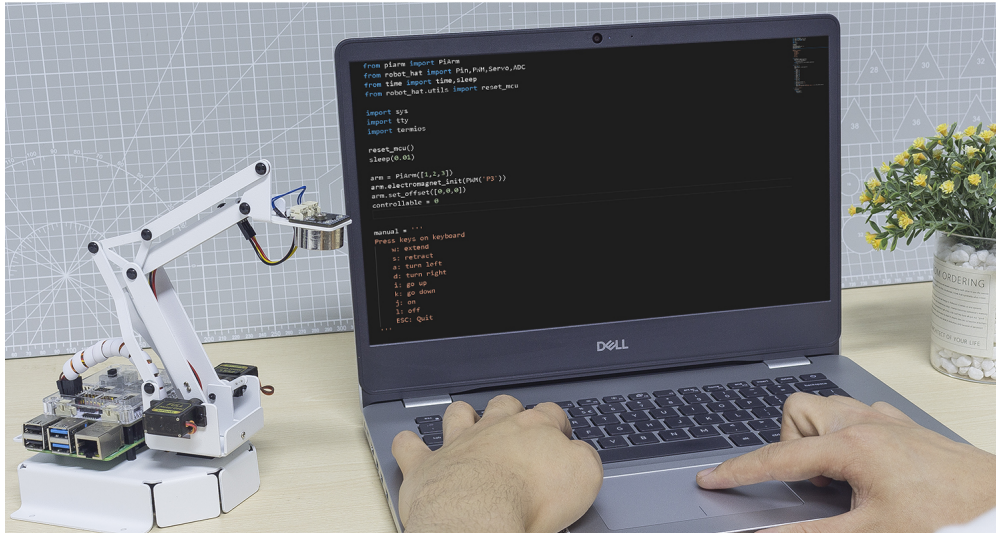
在这个代码中，创建了 `_angles_control()` 函数用来控制 PiArm。

- alpha,beta 和 gamma 分别指的是手臂上的 3 个舵机的角度，参考：[角度模式](#)。
- 左摇杆向上拨动，alpha 增加，让手臂向前伸。
- 左摇杆向下拨动，alpha 减小，让手臂向里缩。
- 左摇杆向左拨动，gamma 增加，让手臂向左转动。
- 左摇杆向右拨动，gamma 减小，让手臂向右转动。
- 右摇杆向上拨动，beta 增加，让手臂向上。
- 右摇杆向下拨动，beta 减小，让手臂向下。
- 最后，分别用左右摇杆的按键来控制电磁铁的开/关。

4.5 键盘控制

除了双摇杆模块，我们还可以用键盘上的按键来控制 PiArm 移动。

- 铲斗 - 键盘控制
- 竖直夹 - 键盘控制
- 电磁铁 - 键盘控制



4.5.1 铲斗 - 键盘控制

运行代码

```
cd /home/pi/piarm/examples
sudo python3 keyboard_control1.py
```

运行代码之后，按照提示，按下键盘上的按键来控制 PiArm 的手臂和铲斗。
但你需要先将铲斗 安装到 PiArm 上。

注解:

- w, s, a, d, i 和 k 用来控制手臂的转动。
- j 和 l 用来控制铲斗的角度。

代码

```
from piarm import PiArm
from robot_hat import Pin, PWM, Servo, ADC
from time import time, sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1, 2, 3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0, 0, 0])
controllable = 0
```

(下页继续)

(续上页)

```
manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: open
    l: close
    ESC: Quit
'''

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True
    if key == 'j':
        bucket -= 1
        flag = True
    elif key == 'l':
        bucket += 1
        flag = True

    if flag == True:
```

(下页继续)

(续上页)

```

        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.
↪component_staus))

if __name__ == "__main__":

    print(manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break

```

它是如何工作的？

```

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

```

该功能引用标准输入流并返回读取的数据流的第一个字符。

- `tty.setraw(sys.stdin.fileno)` 就是将标准输入流改为 raw 模式，即传输过程中所有字符都不会被转义，包括特殊字符。
- `old_settings = termios.tcgetattr(fd)` 和 `termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)` 并起到备份和恢复的作用。

```

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3

```

(下页继续)

(续上页)

```

        flag = True
    elif key == 'k':
        beta -= 3
        flag = True
    if key == 'j':
        bucket -= 1
        flag = True
    elif key == 'l':
        bucket += 1
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.
↪component_staus))

```

在这个代码中，创建了 `control()` 函数来通过读取键盘上的键值来控制 PiArm。

- `alpha`, `beta` 和 `gamma` 分别指的是手臂上的 3 个舵机的角度，参考：角度模式。
- 按下键盘上的 `w` 键，`alpha` 增加，让手臂向前伸。
- 按下键盘上的 `s` 键，`alpha` 减小，让手臂向里缩。
- 按下键盘上的 `a` 键，`gamma` 增加，让手臂向左转动。
- 按下键盘上的 `d` 键，`gamma` 减小，让手臂向右转动。
- 按下键盘上的 `i` 键，`beta` 增加，让手臂向上。
- 按下键盘上的 `k` 键，`beta` 减小，让手臂向下。
- 最后，分别用 `k` 和 `l` 按键来控制铲斗的角度。

```

while True:
    key = readchar().lower()
    control(key)
    if key == chr(27):
        break

```

在主程序中调用 `readchar()` 来读取按键值，然后将读取的键值传入到 `control()` 函数中，这样 PiArm 就会根据不同的按键来移动。`key == chr(27)` 代表按键 `Esc` 按键。

4.5.2 竖直夹 - 键盘控制

运行代码

```

cd /home/pi/piarm/examples
sudo python3 keyboard_control2.py

```

运行代码之后，按照提示，按下键盘上的按键来控制 PiArm 的手臂和竖直夹。

但你需要先将竖直夹 安装到 PiArm 上。

注解:

- `w`, `s`, `a`, `d`, `i` 和 `k` 用来控制手臂的转动。

- j 和 l 用来控制竖直夹的角度。

代码

```
from piarm import PiArm
from robot_hat import Pin, PWM, Servo, ADC
from time import time, sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1, 2, 3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0, 0, 0])
controllable = 0

manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: open
    l: close
    ESC: Quit
'''

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha, beta, gamma = arm.servo_positions
    clip = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
```

(下页继续)

(续上页)

```

        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True

    if key == 'j':
        clip -= 1
        flag = True
    elif key == 'l':
        clip += 1
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_hanging_clip(clip)
        print('servo angles: %s , clip angle: %s'%(arm.servo_positions,arm.component_
↪staus))

if __name__ == "__main__":

    print (manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break

```

在这个代码中，创建了 control() 函数来通过读取键盘上的键值来控制 PiArm。

- alpha,beta 和 gamma 分别指的是手臂上的 3 个舵机的角度，参考：[角度模式](#)。
- 按下键盘上的 w 键，alpha 增加，让手臂向前伸。
- 按下键盘上的 s 键，alpha 减小，让手臂向里缩。
- 按下键盘上的 a 键，gamma 增加，让手臂向左转动。
- 按下键盘上的 d 键，gamma 减小，让手臂向右转动。
- 按下键盘上的 i 键，beta 增加，让手臂向上。
- 按下键盘上的 k 键，beta 减小，让手臂向下。
- 最后，分别用 k 和 l 按键来控制竖直夹的角度。

4.5.3 电磁铁 - 键盘控制

运行代码

```
cd /home/pi/piarm/examples
sudo python3 keyboard_control1.py
```

运行代码之后，按照提示，按下键盘上的按键来控制 PiArm 的手臂和电磁铁。
但你需要先将电磁铁 安装到 PiArm 上。

注解：

- w, s, a, d, i 和 k 用来控制手臂的转动。
- j 和 l 用来控制电磁铁的开关。

代码

```
from piarm import PiArm
from robot_hat import Pin, PWM, Servo, ADC
from time import time, sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1, 2, 3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0, 0, 0])
controllable = 0

manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: on
    l: off
    ESC: Quit
'''

def readchar():
    fd = sys.stdin_FILENO()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin_FILENO())
        ch = sys.stdin.read(1)
    finally:
```

(下页继续)

(续上页)

```

        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    status = ""

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True

    if key == 'j':
        arm.set_electromagnet('on')
    elif key == 'l':
        arm.set_electromagnet('off')

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        print('servo angles: %s , electromagnet status: %s'%(arm.servo_positions,
↪status))

if __name__ == "__main__":

    print(manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break

```

在这个代码中，创建了 control() 函数来通过读取键盘上的键值来控制 PiArm。

- alpha,beta 和 gamma 分别指的是手臂上的 3 个舵机的角度，参考：角度模式。
- 按下键盘上的 w 键，alpha 增加，让手臂向前伸。
- 按下键盘上的 s 键，alpha 减小，让手臂向里缩。
- 按下键盘上的 a 键，gamma 增加，让手臂向左转动。

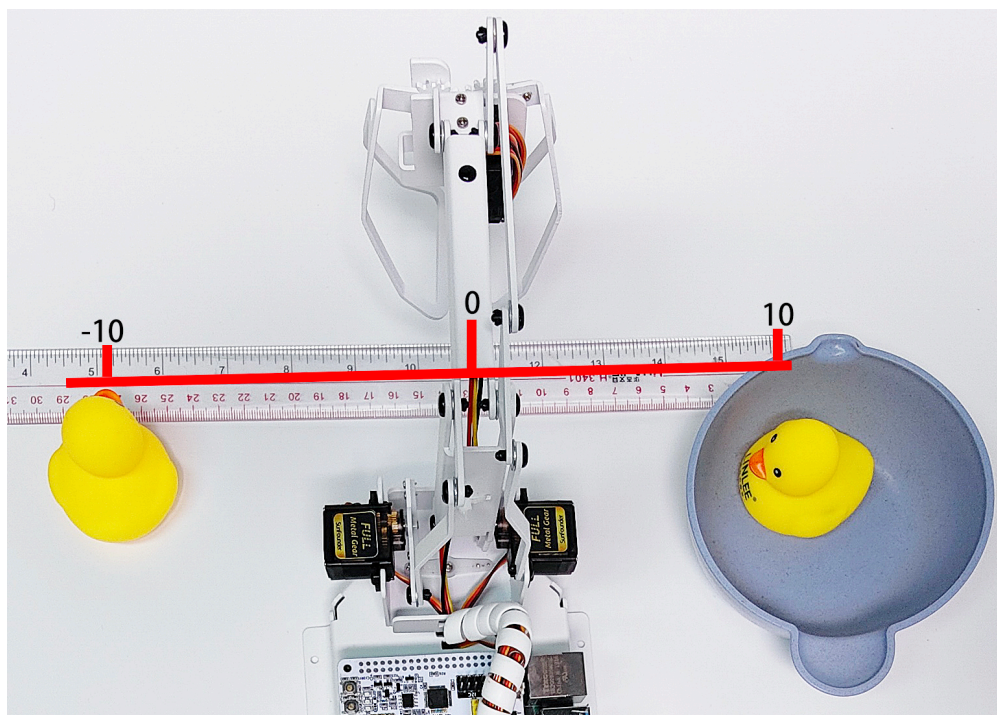
- 按下键盘上的 d 键，gamma 减小，让手臂向右转动。
- 按下键盘上的 i 键，beta 增加，让手臂向上。
- 按下键盘上的 k 键，beta 减小，让手臂向下。
- 最后，分别用 k 和 l 按键来控制电磁铁的开关。

4.6 坐标模式

PiArm 的手臂有 2 种控制模式: 角度和 坐标。

- **角度模式:** 向 PiArm 手臂上的 3 个舵机写入一定的角度，使其到达特定位置。
- **坐标模式:** 为 PiArm 建立空间坐标系并设置控制点，将空间坐标写入该控制点，使其达到特定位置。

本项目通过坐标模式，设定 2 个坐标点，让机械臂将左边的橡皮鸭夹到右边的碗里。但你需要先将**竖直夹**装到 PiArm 上。



运行代码

```
cd /home/pi/piarm/examples
sudo python3 coordinate_mode.py
```

代码运行后，你就能看到 PiArm 将左边的橡皮鸭夹到右边的碗里。

但你需要先将**竖直夹**安装到 PiArm 上。

代码

```
from re import M
from robot_hat import PWM
from robot_hat.utils import reset_mcu
from time import sleep
```

(下页继续)

(续上页)

```

from piarm import PiArm

reset_mcu()
sleep(0.01)

" Grab an object from one coordinate to another coordinate"

arm = PiArm([1,2,3])
arm.set_offset([0,0,0])
arm.hanging_clip_init(PWM('P3'))

if __name__ == "__main__":

    start_coord = [-100, 40, 20] # x,y,z
    end_coord = [100, 40, 30] # x,y,z

    arm.set_speed(60)
    arm.set_hanging_clip(20)
    arm.do_by_coord(start_coord)
    arm.set_hanging_clip(90)

    start_coord_up = [start_coord[0], start_coord[1], 80]
    arm.do_by_coord(start_coord_up)

    end_coord_up = [end_coord[0], end_coord[1], 80]
    arm.do_by_coord(end_coord_up)

    arm.do_by_coord(end_coord)
    arm.set_hanging_clip(20)
    arm.do_by_coord(end_coord_up)

```

它是如何工作的？

```

start_coord = [-100, 40, 20] # x,y,z
end_coord = [100, 40, 30] # x,y,z

```

- start_coord: 左边橡皮鸭的坐标。
- end_coord: 右边碗的坐标。

注解:

- 这里的坐标都是指的控制点的坐标，但是装好臂端工具后，X 和 Y 坐标的实际距离大一点。
- 不同的臂端工具，误差距离不一样。比如竖直夹和电磁铁为 3-4cm, 铲斗为 6-7cm。
- 比如在这里 X 坐标写的是 100，但实际距离是 13-14cm。
- 一般建议 X 的坐标是-80~80，但由于这里 Y 坐标值较小（建议范围是 30~130），所以设置为 100 也是能到的。但如果你增大了 Y 坐标值，由于连杆作用，X 坐标值需要根据实际情况调小一点。

```

arm.set_speed(60)
arm.set_hanging_clip(20)
arm.do_by_coord(start_coord)

```

(下页继续)

(续上页)

```

arm.set_hanging_clip(90)

start_coord_up = [start_coord[0], start_coord[1], 80]
arm.do_by_coord(start_coord_up)

end_coord_up = [end_coord[0], end_coord[1], 80]
arm.do_by_coord(end_coord_up)

arm.do_by_coord(end_coord)
arm.set_hanging_clip(20)
arm.do_by_coord(end_coord_up)

```

- PiArm 先张开竖直夹 (20°), 然后转动到左边橡皮鸭的位置 (start_coord), 再合拢竖直夹 (90°)。
- PiArm 抬起头 (start_coord_up), 再转动到右边的碗的正上方 (end_coord_up)。
- PiArm 低头 (end_coord), 再张开竖直夹 (20°) 让橡皮鸭掉落到碗里, 最后再抬起头 (end_coord_up)。

4.7 记忆功能

PiArm 提供了一个记录动作的功能, 可以让 PiArm 自动做一些重复性的动作。

在这个项目中, 我们来看下如何实现这个功能。

运行代码

```

cd /home/pi/piarm/examples
sudo python3 memory_function.py

```

代码运行之后, 你可以用左右摇杆来控制 PiArm 的转动和铲斗 (但你需要先将铲斗安装到 PiArm 上), 按下左摇杆来记录 PiArm 的一次移动, 记录了几组动作之后, 你可以按下右摇杆来让 PiArm 来复现这些动作。

只记录点与点之间的变化, 如果起点和终点是一样的, 中间做了很多次移动, 但只按下一次来记录, 它会直接从起点到终点, 不会记录中间过程。

代码

```

from robot_hat import Servo, PWM, Joystick, ADC, Pin
from robot_hat.utils import reset_mcu
from robot_hat import TTS
from time import sleep
from piarm import PiArm

t = TTS()
reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'), ADC('A1'), Pin('D0'))
rightJoystick = Joystick(ADC('A2'), ADC('A3'), Pin('D1'))

arm = PiArm([1, 2, 3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0, 0, 0])

def _angles_control():
    arm.speed = 100

```

(下页继续)

(续上页)

```

flag = False
alpha,beta,gamma = arm.servo_positions
bucket = arm.component_staus
global i

if leftJoystick.read_status() == "up":
    alpha += 1
    flag = True
elif leftJoystick.read_status() == "down":
    alpha -= 1
    flag = True
if leftJoystick.read_status() == "left":
    gamma += 1
    flag = True
elif leftJoystick.read_status() == "right":
    gamma -= 1
    flag = True
if rightJoystick.read_status() == "up":
    beta += 1
    flag = True
elif rightJoystick.read_status() == "down":
    beta -= 1
    flag = True

if rightJoystick.read_status() == "left":
    bucket += 2
    flag = True
elif rightJoystick.read_status() == "right":
    bucket -= 2
    flag = True

if leftJoystick.read_status() == "pressed":
    arm.record()
    t.say("record")
    print('step %s : %s'%(i,arm.steps_buff[i*2]))
    i += 1
    sleep(0.05)
elif rightJoystick.read_status() == "pressed":

    t.say("action")
    arm.set_speed(80)
    arm.record_reproduce(0.05)
    arm.set_speed(100)

if flag == True:
    arm.set_angle([alpha,beta,gamma])
    arm.set_bucket(bucket)
    print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.
↪component_staus))

if __name__ == "__main__":
    print(arm.servo_positions)
    i = 0
    while True:
        _angles_control()
        sleep(0.01)

```

它是如何工作的？

在这个代码中，我们来重点看下 `_angles_control()` 函数，它是用来读取双摇杆的值之后，进行不同的操作。

1. 控制手臂的移动

```
if leftJoystick.read_status() == "up":
    alpha += 1
    flag = True
elif leftJoystick.read_status() == "down":
    alpha -= 1
    flag = True
if leftJoystick.read_status() == "left":
    gamma += 1
    flag = True
elif leftJoystick.read_status() == "right":
    gamma -= 1
    flag = True
if rightJoystick.read_status() == "up":
    beta += 1
    flag = True
elif rightJoystick.read_status() == "down":
    beta -= 1
    flag = True
```

- alpha, beta 和 gamma 分别指的是手臂上的 3 个舵机的角度，参考：[角度模式](#)。
- 左摇杆向上拨动，alpha 增加，让手臂向前伸。
- 左摇杆向下拨动，alpha 减小，让手臂向里缩。
- 左摇杆向左拨动，gamma 增加，让手臂向左转动。
- 左摇杆向右拨动，gamma 减小，让手臂向右转动。
- 右摇杆向上拨动，beta 增加，让手臂向上。
- 右摇杆向下拨动，beta 减小，让手臂向下。

2. 控制铲斗的角度

```
if rightJoystick.read_status() == "left":
    bucket += 2
    flag = True
elif rightJoystick.read_status() == "right":
    bucket -= 2
    flag = True
```

- 右摇杆向左拨动，让铲斗回卷
- 右摇杆向右拨动，让铲斗向外延伸。

3. 记录动作和复现动作

```
if leftJoystick.read_status() == "pressed":
    arm.record()
    t.say("record")
    print('step %s : %s'%(i, arm.steps_buff[i*2]))
    i += 1
    sleep(0.05)
elif rightJoystick.read_status() == "pressed":
```

(下页继续)

(续上页)

```
t.say("action")
arm.set_speed(80)
arm.record_reproduce(0.05)
arm.set_speed(100)
```

- 如果左摇杆按下，调用 `record()` 函数来记录动作，PiArm 会提示已记录。在终端会显示此时的角度及记录的动作数。
- 如果右摇杆按下，调用 `record_reproduce()` 函数来复现记录的动作，PiArm 会提示开始做动作。

4. 将角度写给 PiArm

```
if flag == True:
    arm.set_angle([alpha,beta,gamma])
    arm.set_bucket(bucket)
    print('servo angles: %s , bucket angle: %s'%(arm.servo_positions,arm.component_
↪staus))
```

将手臂的角度和铲斗的角度写给 PiArm，让它转动到这些角度。

如果你的臂端工具接的竖直夹或者是电磁铁，你可以参考以下链接来修改上面的代码：

- 竖直夹 - 摇杆控制
- 电磁铁 - 摇杆控制

4.8 游戏 - 抓娃娃

现在我们来玩一个抓娃娃的游戏，看看谁能在规定的时间内用 PiArm 抓到更多的娃娃。为了玩这个游戏，我们需要实现两个功能，第一个功能是用双摇杆模块控制 PiArm，第二个功能是计时，当倒计时结束后，我们就不能再控制 PiArm。这两部分必须同时执行。



运行代码

```
cd /home/pi/piarm/examples
sudo python3 game_catching_dolls.py
```

代码运行后，同时按下左右摇杆来开始游戏，此时你就可以用双摇杆模块来控制 PiArm 来抓取娃娃，请注意时间，60 秒之后，PiArm 将提示游戏结束，你将无法再继续操作 PiArm。

代码

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep
from robot_hat import TTS
import threading

from piarm import PiArm

reset_mcu()
sleep(0.01)
t = TTS()

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])
arm.speed = 100
game_flag = 0
```

(下页继续)

(续上页)

```

def control():

    alpha,beta,gamma = arm.servo_positions
    clip = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
    elif leftJoystick.read_status() == "down":
        alpha -= 1
    if leftJoystick.read_status() == "left":
        gamma += 1
    elif leftJoystick.read_status() == "right":
        gamma -= 1
    if rightJoystick.read_status() == "up":
        beta += 1
    elif rightJoystick.read_status() == "down":
        beta -= 1
    if leftJoystick.read_status() == "pressed":
        clip += 1
    elif rightJoystick.read_status() == "pressed":
        clip -= 1

    # if key_flag == True:
    arm.set_angle([alpha,beta,gamma])
    arm.set_hanging_clip(clip)
    # print('coord: %s , servo angles: %s , clip angle: %s'%(arm.current_coord,
    ↪arm.servo_positions,arm.component_staus))

def timing():
    sleep(60)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("game over")
    global game_flag
    game_flag = 0

if __name__ == "__main__":

    thread1 = threading.Thread(target = timing)
    thread1.start()
    print("Press two joysticks at the same time to start the game")

    while True:
        if leftJoystick.read_status() == "pressed" and rightJoystick.read_status() ↪
        ↪== "pressed":
            t.say("timing begins")
            game_flag = 1
            if game_flag == 1:
                control()

```

它是如何工作的？

这个代码是在竖直夹 - 摇杆控制 项目的基础上加上了计时。

```
def timing():
    sleep(60)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("game over")
    global game_flag
    game_flag = 0
```

使用 sleep() 函数进行 60 秒的计时，随后就让 PiArm 进行 3, 2, 1 倒计时报数，时间到了之后，让 game_flag 为 0，此时将无法再控制 PiArm。

```
if __name__ == "__main__":

    thread1 = threading.Thread(target = timing)
    thread1.start()
    print("Press two joysticks at the same time to start the game")
```

让 timing() 函数以另外一个线程运行，这样就可以在控制 PiArm 的同时，进行计时。

```
while True:
    if leftJoystick.read_status() == "pressed" and rightJoystick.read_status() ==
    ↪ "pressed":
        t.say("timing begins")
        game_flag = 1
    if game_flag == 1:
        control()
```

这是代码的主要流程，当左右摇杆同时按下时，PiArm 说计时开始，让 game_flag 为 1，此时就可以调用 control() 函数来控制 PiArm。

4.9 游戏 - 收集铁片

在这个项目中，准备三角形，圆形，正方形 3 种形状的铁片，PiArm 会随机说一种形状，你需要控制 PiArm 将相应形状的铁片放到相应的盒子里面。

运行代码

```
cd /home/pi/piarm/examples
sudo python3 game_iron_collection.py
```

代码运行后，先按键盘上的 p 键来启动游戏，PiArm 将提示游戏开始，然后随机说出一种形状 (Round, Triangle 和 Square)，此时你可以用键盘上的 w, s, a, d, i, k, j 和 l 来控制 PiArm 来吸取对应形状的铁片。60 秒后，将提示游戏结束，你将无法再控制 PiArm。

如果你想停止代码运行，你需要先按下 Esc 键，然后再按 Ctrl+C。

注解:

- w, s, a, d, i 和 k 用来控制手臂的转动。
- j 和 l 用来控制电磁铁的开关。

代码

```

from piarm import PiArm
from robot_hat import Pin, PWM, Servo, ADC
from time import time, sleep
from robot_hat.utils import reset_mcu
from robot_hat import TTS

import threading
import sys
import tty
import termios
import random

reset_mcu()
sleep(0.01)
t = TTS()

arm = PiArm([1, 2, 3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0, 0, 0])
arm.speed = 100
flag = False

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

manual1 = '''
Press keys on keyboard
    p: Game Start
    ESC: Stop
'''

manual2 = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: on
    l: off
'''

# def control():

```

(下页继续)

(续上页)

```
#     while flag == True:
#         arm.speed = 100
#         flag = False
#         alpha,beta,gamma = arm.servo_positions

def control(key):
    alpha,beta,gamma = arm.servo_positions

    if key == 'a':
        gamma += 3
    elif key == 'd':
        gamma -= 3
    if key == 's':
        alpha -= 3
    elif key == 'w':
        alpha += 3
    if key == 'i':
        beta += 3
    elif key == 'k':
        beta -= 3
    if key == 'j':
        arm.set_electromagnet('on')
    elif key == 'l':
        arm.set_electromagnet('off')
    arm.set_angle([alpha,beta,gamma])

def timing():
    global flag
    while True:
        if flag == True:
            t.say("game start")
            sleep(60)
            t.say("three")
            sleep(1)
            t.say("two")
            sleep(1)
            t.say("one")
            sleep(1)
            t.say("game over")
            flag = False

def say_shape():
    k = random.randint(1,3)
    if k == 1:
        t.say("Round")
    if k == 2:
        t.say("Triangle")
    if k == 3:
        t.say("Square")

if __name__ == "__main__":

    print(manual1)

    thread1 = threading.Thread(target = timing)
```

(下页继续)

(续上页)

```

thread1.start()

while True:
    key = readchar().lower()
    if key == 'p':
        print(manual2)
        flag = True
        sleep(3)
        say_shape()
    if flag == True:
        control(key)
    if key == chr(27):
        print("press ctrl+c to quit")
        break

```

它是如何工作的？

这个代码是在项目[电磁铁 - 键盘控制](#)的基础上加上了计时和说出随机形状的部分。

```

def timing():
    global flag
    while True:
        if flag == True:
            t.say("game start")
            sleep(60)
            t.say("three")
            sleep(1)
            t.say("two")
            sleep(1)
            t.say("one")
            sleep(1)
            t.say("game over")
            flag = False

```

这个 timing() 函数是用来计时，提示游戏开始后，进行 60 秒的计时，然后进行 3, 2, 1 的倒计时报数，再提示游戏结束，让 flag 为 False。

```

def say_shape():
    k = random.randint(1,3)
    if k == 1:
        t.say("Round")
    if k == 2:
        t.say("Triangle")
    if k == 3:
        t.say("Square")

```

这个 say_shape() 函数是让 PiArm 随机说出一种形状。

```

if __name__ == "__main__":

    print(manual1)

    thread1 = threading.Thread(target = timing)
    thread1.start()

    while True:
        key = readchar().lower()

```

(下页继续)

(续上页)

```
if key == 'p':
    print(manual2)
    flag = True
    sleep(3)
    say_shape()
if flag == True:
    control(key)
if key == chr(27):
    break
print("press ctrl+c to quit")
```

这是代码的主要流程：

- 在终端打印出按键提示，按下 p 来开始游戏，让 timing() 以单独的线程运行。
- 调用 readchar() 函数来读取键值。
- 如果读取到按键 p 被按下，就打印出按键提示，让 flag 为 True，此时 timing() 函数开始计时，3 秒后，调用 say_shape() 函数来让 PiArm 随机说一个形状。
- 如果 flag 为 True，调用 control() 函数来让 PiArm 根据按键值转动。
- chr(27) 为 Esc 按键，如果 Esc 按键被按下，退出主循环。这一步是因为使用了 readchar() 函数一直读取键盘，所以无法直接通过 Ctrl+C 来停止代码运行。
- 此时就能通过 Ctrl+C 来停止代码运行。

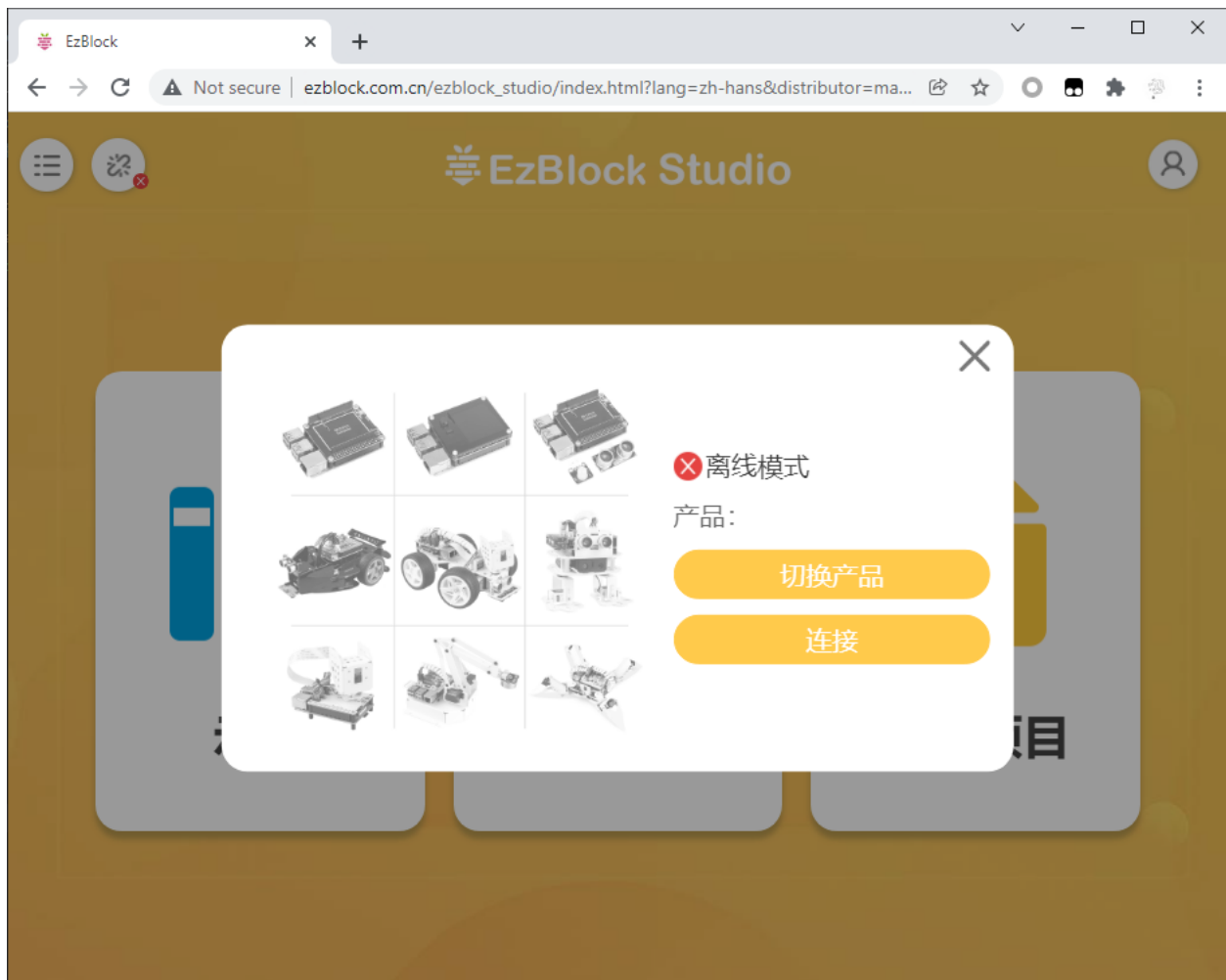
5.1 网页版 EzBlock

进入 [网页版 EzBlock](#)。

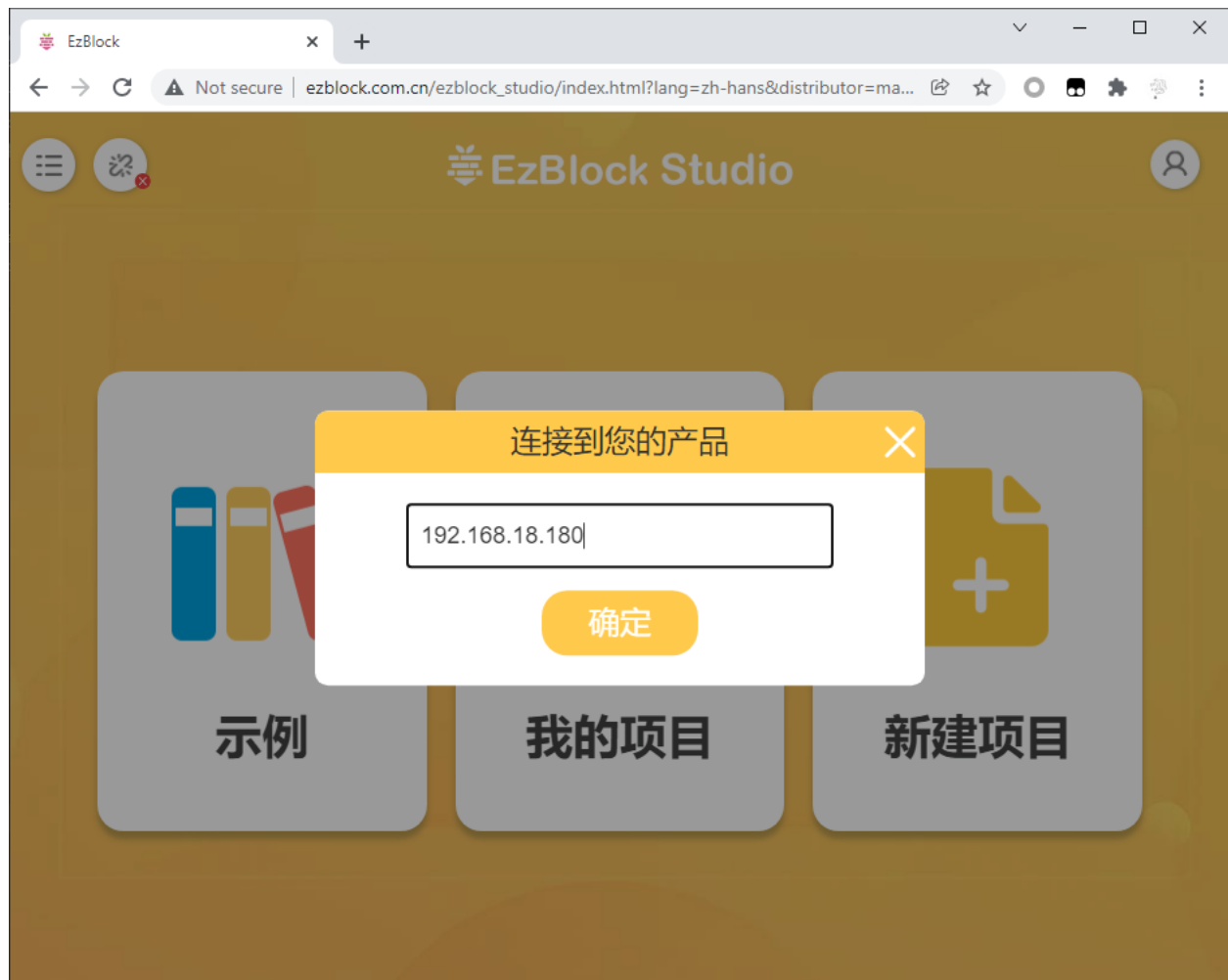
在首页点击左上角的产品按钮。



点击连接。



输入 IP 地址。



随后将弹出“连接成功”的提示，此时你可以回到玩转 *EzBlock*。



5.2 apt 和 pip 更换国内源

树莓派系统默认的 apt 源和 pip 源都是国外的服务器，使用国内网络访问可能会发生超时（ReadTimeoutError），或被拒绝访问的情况，如果是这样我们可以将 apt 和 pip 更改为国内的源，步骤如下所示：

1. apt 更换国内源

- 1) 访问链接：<https://mirrors.tuna.tsinghua.edu.cn/help/raspbian/> 以了解配置文件修改详情。
- 2) 不同的树莓派系统版本修改不同，先选择对应的版本，比如我的是 Debian 10（buster），如果你的树莓派系统是 bullseye 则选择对应版本，若没有符合的版本则请重新安装 buster 及以下版本的系统。

使用说明

选择你的 Raspbian 对应的 Debian 版本:

Debian 10 (buster) ▼

Debian 9 (stretch)

Debian 10 (buster)

Debian 11 (bullseye)

```
# 编辑 `/etc/apt/sources.list` 文件，用以下内容取代：
deb [arch=armhf] http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main non-free contrib rpi
deb-src http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main non-free contrib rpi

# 如果需要 arm64 软件源，在 `/etc/apt/sources.list` 中加上
deb [arch=arm64] http://mirrors.tuna.tsinghua.edu.cn/raspbian/multiarch/ buster main

# 编辑 `/etc/apt/sources.list.d/raspi.list` 文件，删除原文件所有内容，用以下内容取代：
deb http://mirrors.tuna.tsinghua.edu.cn/raspberrypi/ buster main
```

注解：一般不删除原内容，可以将其用 # 注释掉。

3) 我们这里以 raspbian buster 版本为例，用 nano 命令打开 /etc/apt/sources.list 文件。

```
sudo nano /etc/apt/sources.list
```

4) 然后用 # 将原本的内容注释掉，在最后面附上下面的代码。

```
# deb http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free rpi
# Uncomment line below then 'apt-get update' to enable 'apt-get source'
#deb-src http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free rpi

deb [arch=armhf] http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main_
↪non-free contrib rpi
deb-src http://mirrors.tuna.tsinghua.edu.cn/raspbian/raspbian/ buster main non-free_
↪contrib rpi
```

5) 按下 Ctrl+O 保存，按下 Ctrl+X 和 Y 退出。

6) 用 nano 命令打开 etc/apt/sources.list.d/raspi.list 文件。

```
sudo nano /etc/apt/sources.list.d/raspi.list
```

7) 然后用 # 将原本的内容注释掉，在最后面附上 deb... 代码。

```
# deb http://archive.raspberrypi.org/debian/ buster main
# Uncomment line below then 'apt-get update' to enable 'apt-get source'
#deb-src http://archive.raspberrypi.org/debian/ buster main

deb http://mirrors.tuna.tsinghua.edu.cn/raspberrypi/ buster main ui
```

8) 按下 Ctrl+O 保存，按下 Ctrl+X 和 Y 退出。

9) 用以下命令更新软件列表：

```
sudo apt update
```

2.(Pypi) pip 更换国内源

可以参考链接 <https://mirrors.tuna.tsinghua.edu.cn/help/pypi/> 了解详情。

有两种方法修改配置文件。

方法一：使用 pip 指令设置

```
pip3 config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
```

方法二：手动编辑文件

Linux/Mac os 环境中，配置文件位置在 `~/.pip/pip.conf`（如果不存在则手动创建该目录和文件）。

```
sudo mkdir -p ~/.pip
sudo nano ~/.pip/pip.conf
```

然后按如下编辑文件内容

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
[install]
trusted-host = https://pypi.tuna.tsinghua.edu.cn
```

最后 apt 和 pip 都已经更换了国内下载源，这样下载速度就会提高很快不会导致下载失败了。

5.3 Filezilla 软件



文件传输协议 (FTP) 是一种标准通信协议，用于在计算机网络上将计算机文件从服务器传输到客户端。

Filezilla 是一款开源软件，不仅支持 FTP，还支持 FTP over TLS (FTPS) 和 SFTP。我们可以使用 Filezilla 将本地文件（如图片和音频等）上传到树莓派，或者从树莓派下载文件到本地。

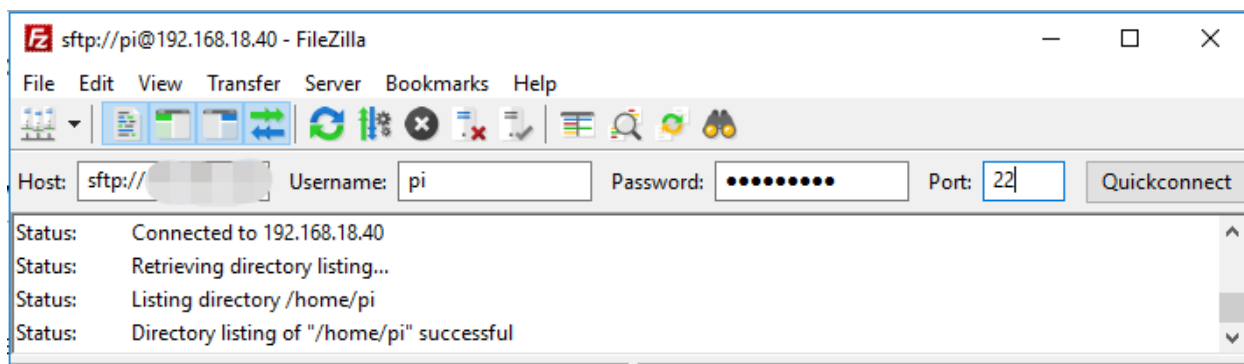
第 1 步: 下载 Filezilla。

在 [Filezilla 官方网站](#) 下载客户端，Filezilla 有一个很好的教程，请参考: [Filezilla 文档](#)。

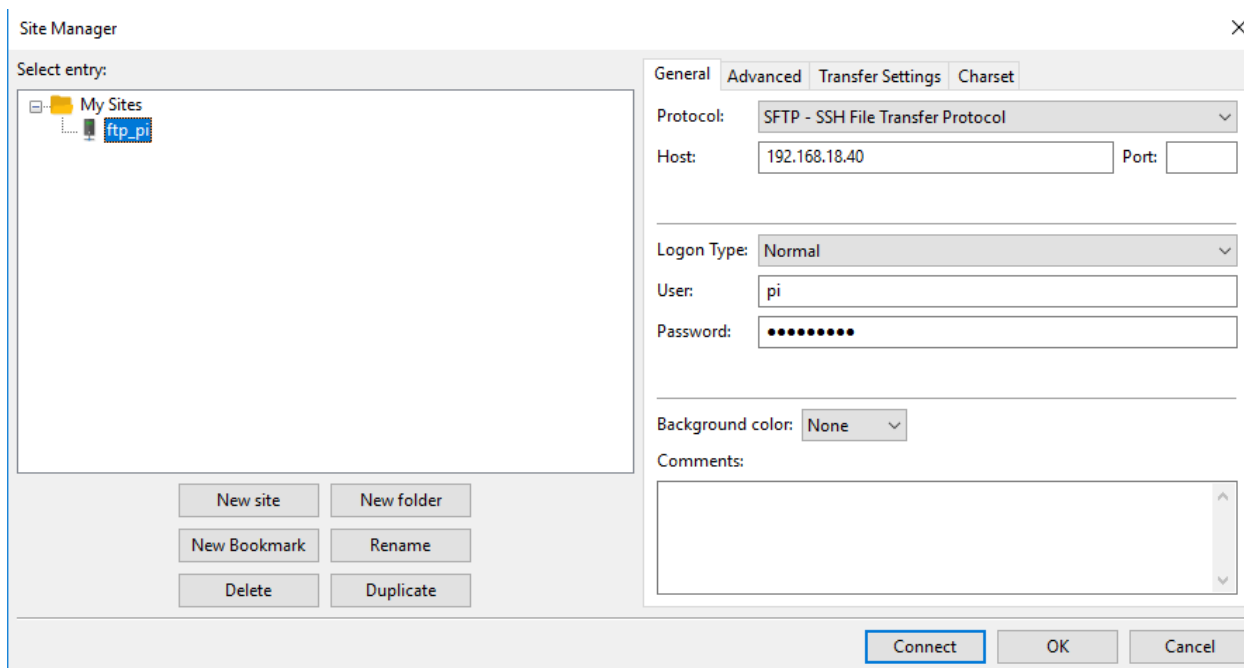
第 2 步: 连接树莓派

快速安装后打开它，现在 [连接到 FTP 服务器](#)。

它有 3 种连接方式，这里我们使用 [快速连接](#) 栏。输入 [主机名/IP](#)、[用户名](#)、[密码](#) 和 [端口 \(22\)](#)，然后点击 [快速连接](#) 或按 [回车](#) 连接到服务器。

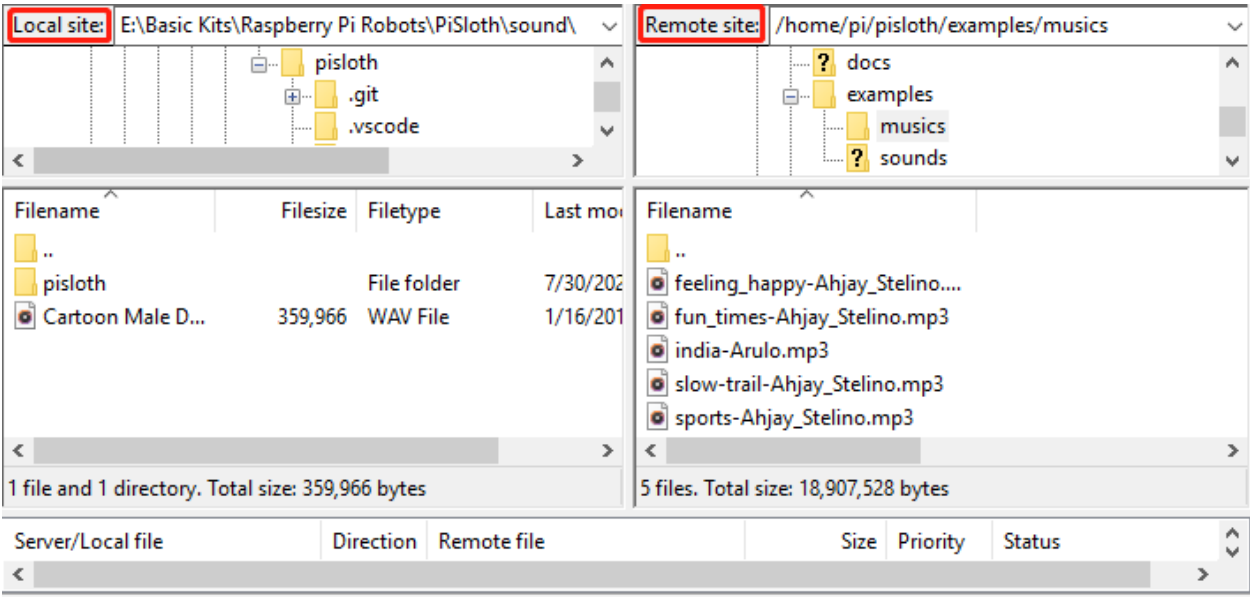


注解：快速连接是测试您的登录信息的好方法。如果要创建永久条目，可以在快速连接成功后选择 **File->Copy Current Connection to Site Manager**（文件->将当前连接复制到站点管理器），输入名称并单击 **确定**。下次您将能够通过 **File -> Site Manager**（文件->站点管理器。）中选择先前保存的站点进行连接。



第3步：上传/下载文件。

您可以通过拖放将本地文件上传到树莓派，也可以将树莓派文件中的文件下载到本地。

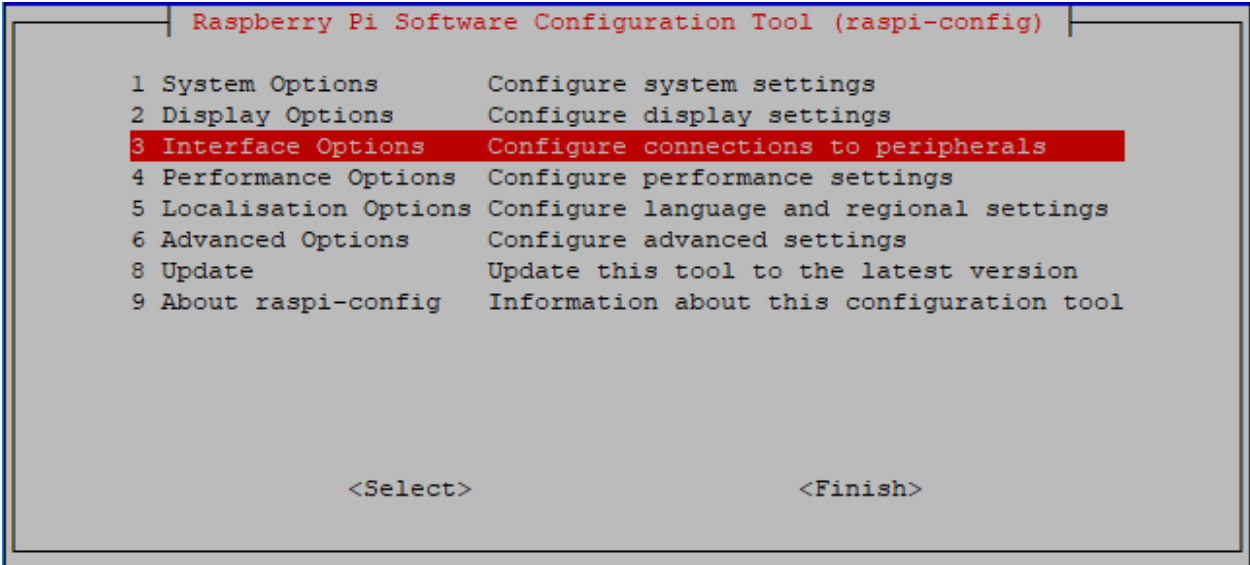


5.4 I2C 配置

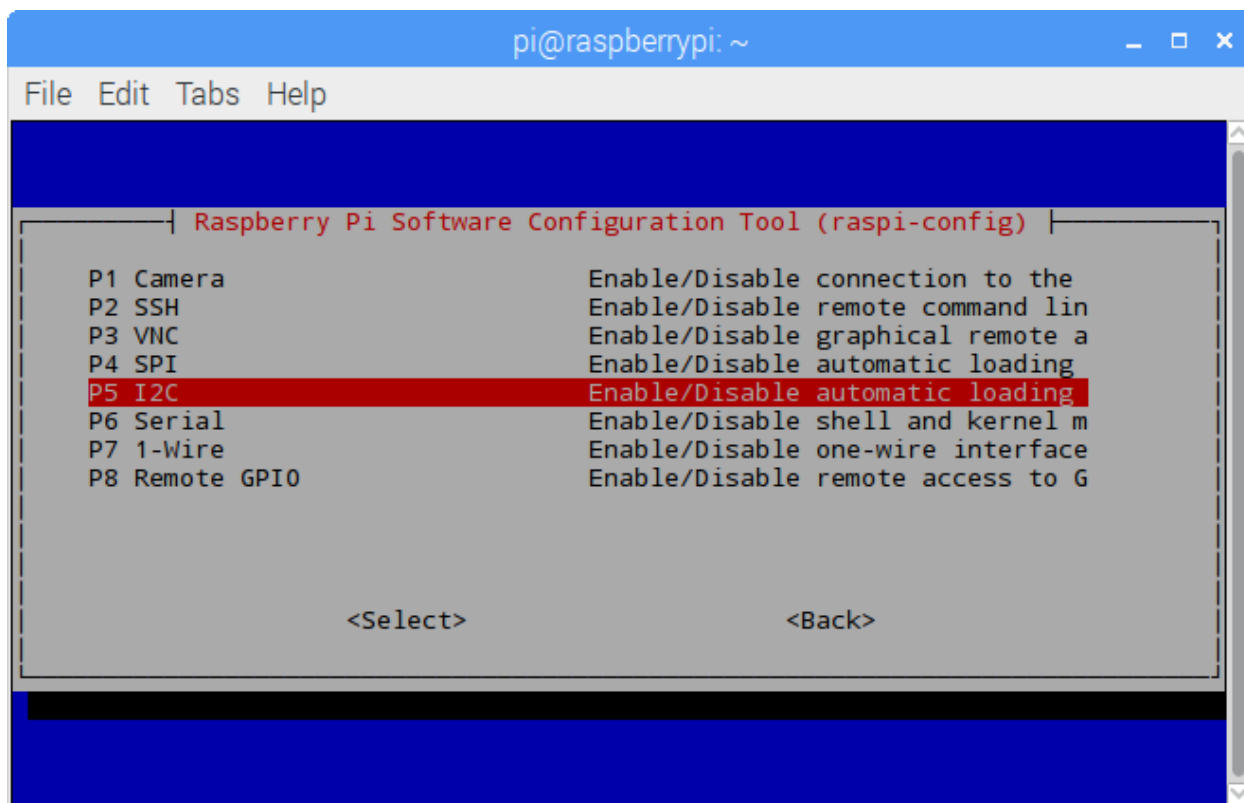
启用你的树莓派的 I2C 端口。如果你已经启用了它，请跳过这一步；如果你不知道你是否已经做了，请继续。

```
sudo raspi-config
```

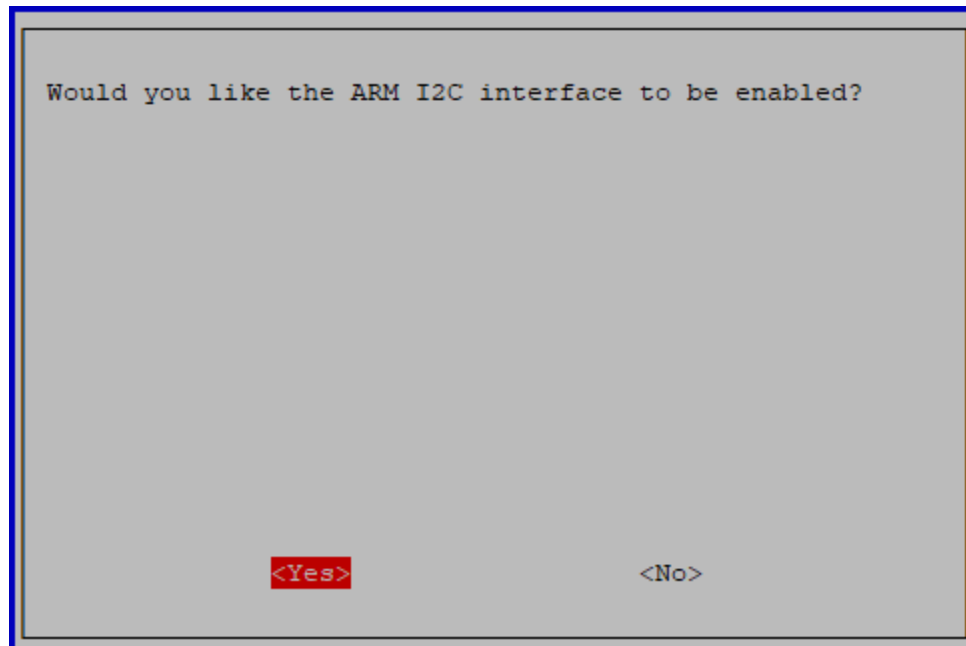
3 Interfacing options



P5 I2C



<Yes>, 然后 <Ok> -> <Finish>.



5.5 远程桌面

有两种方法可以远程控制树莓派的桌面。

VNC 和 XRDP，你可以使用它们中的任何一种。

5.5.1 VNC

你可以通过 VNC 使用远程桌面的功能。

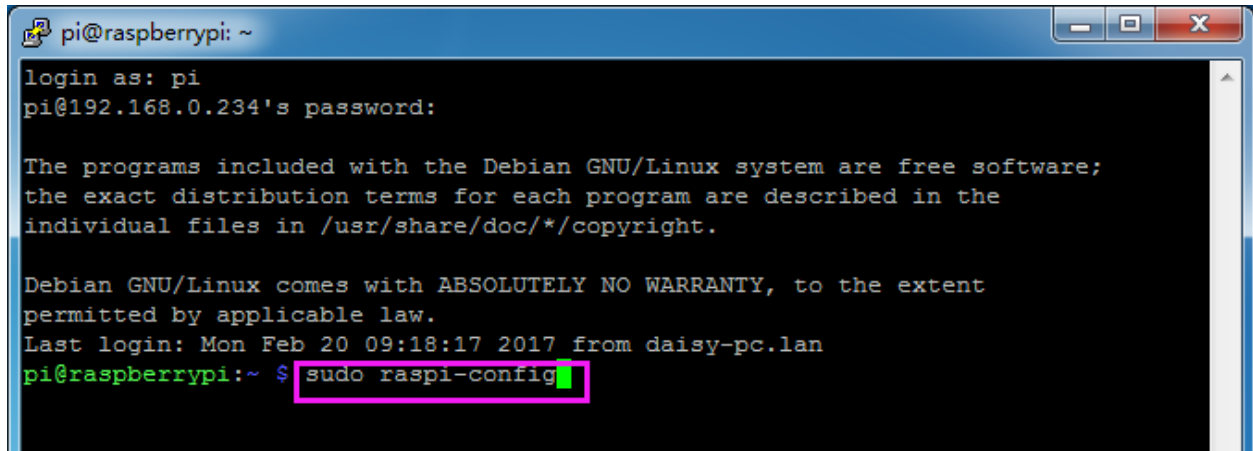
启用 VNC 服务

系统中已经安装了 VNC 服务。默认情况下，VNC 被禁用。默认情况下，VNC 是禁用的。你需要在配置中启用它。

第 1 步

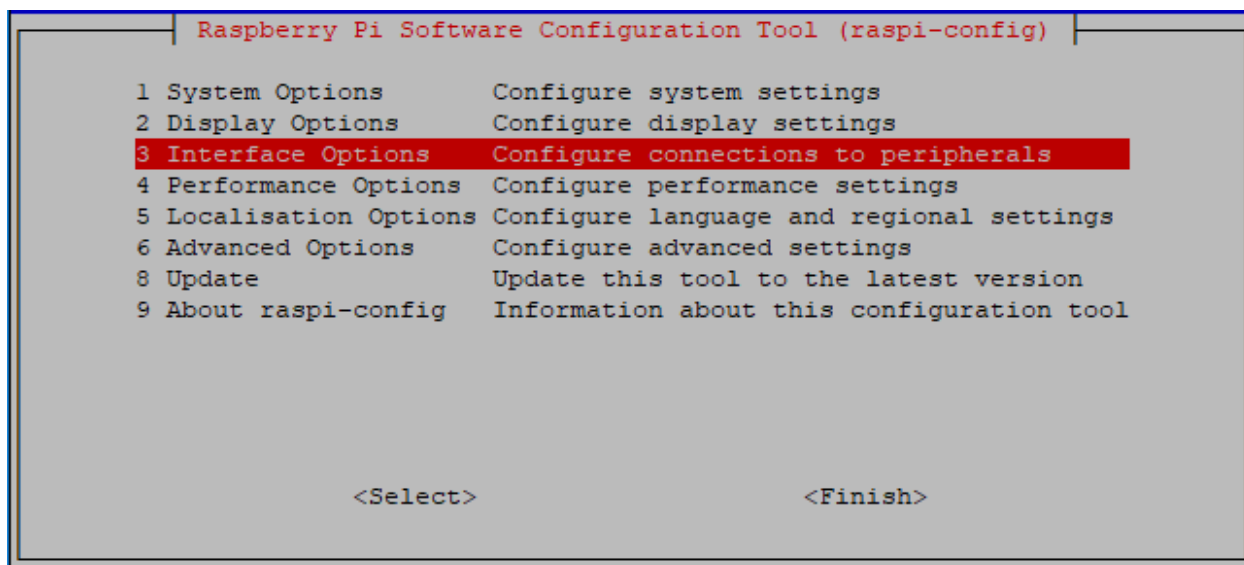
输入以下命令。

```
sudo raspi-config
```



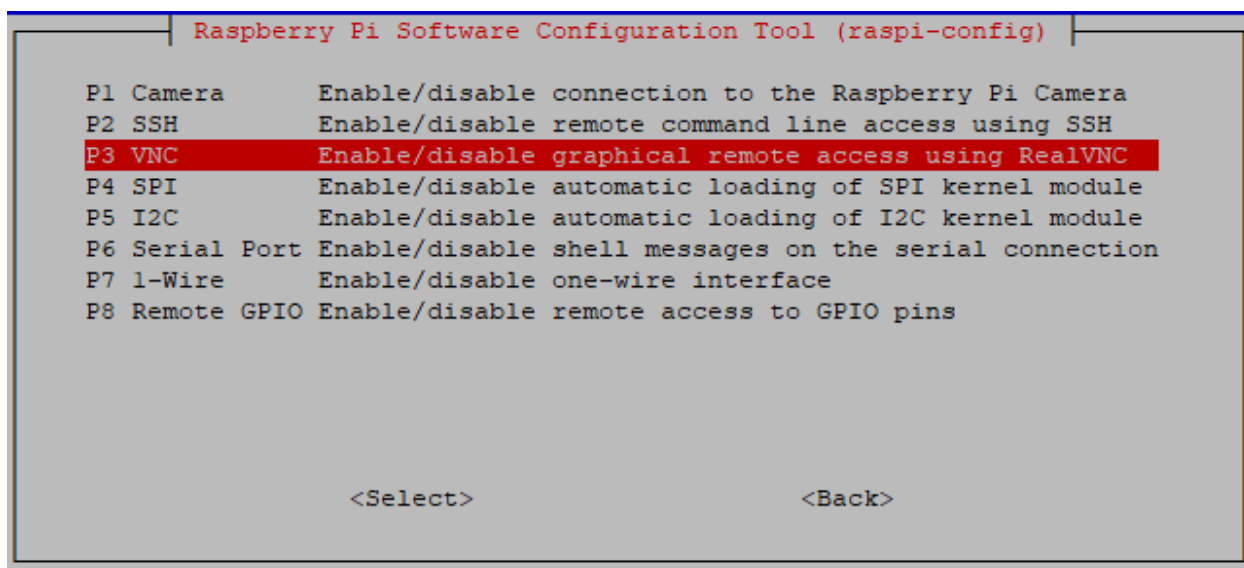
第 2 步

选择 **3 Interfacing Option**，按你键盘上的向下箭头键。键盘上的向下箭头键，然后按 **Enter** 键。



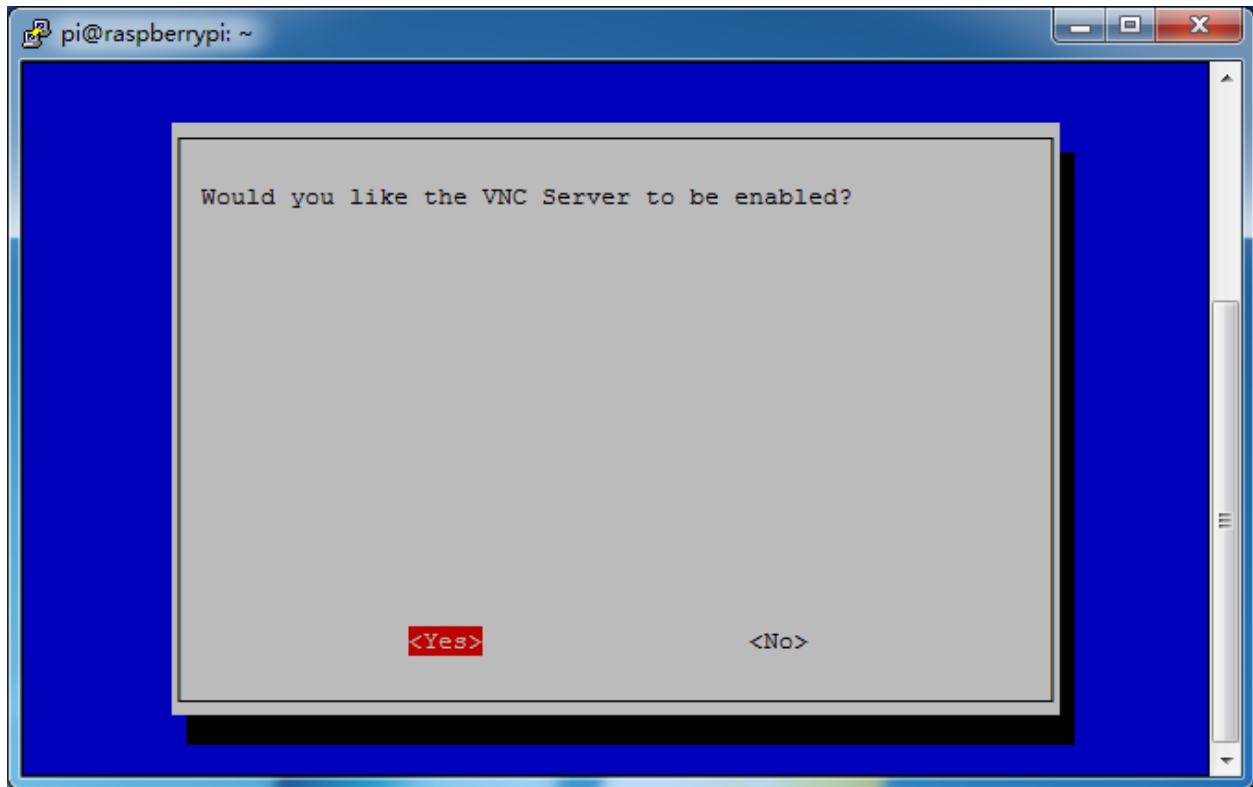
第3步

P3 VNC



第4步

选择 Yes -> OK -> Finish 来退出配置。



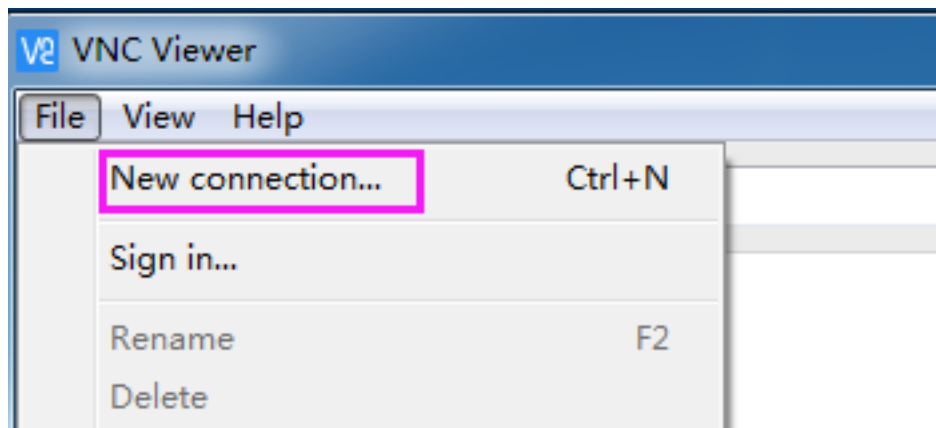
登录 VNC

第 1 步

你需要在个人电脑上下载并安装 VNC Viewer。安装完成后，打开它。

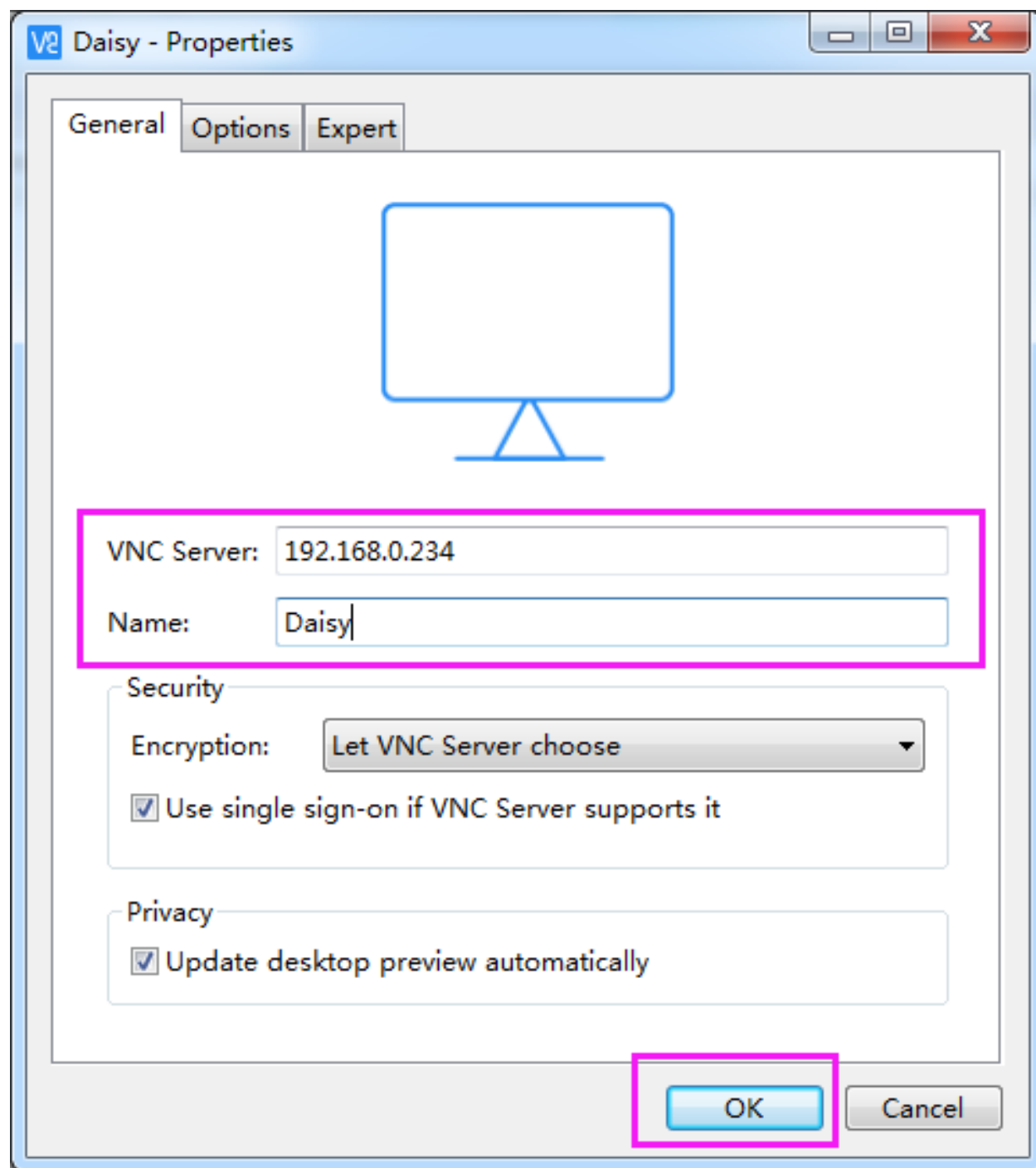
第 2 步

然后选择“New connection”。



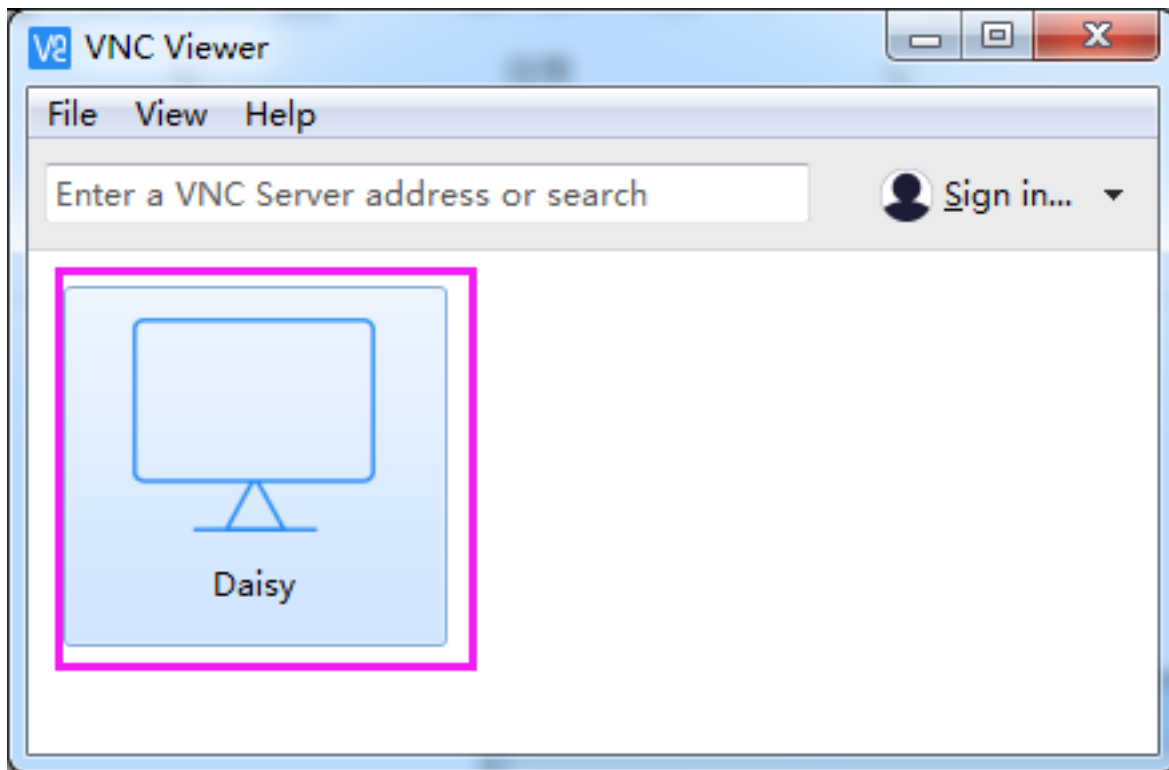
第 3 步

输入树莓派的 IP 地址和任意 名称。



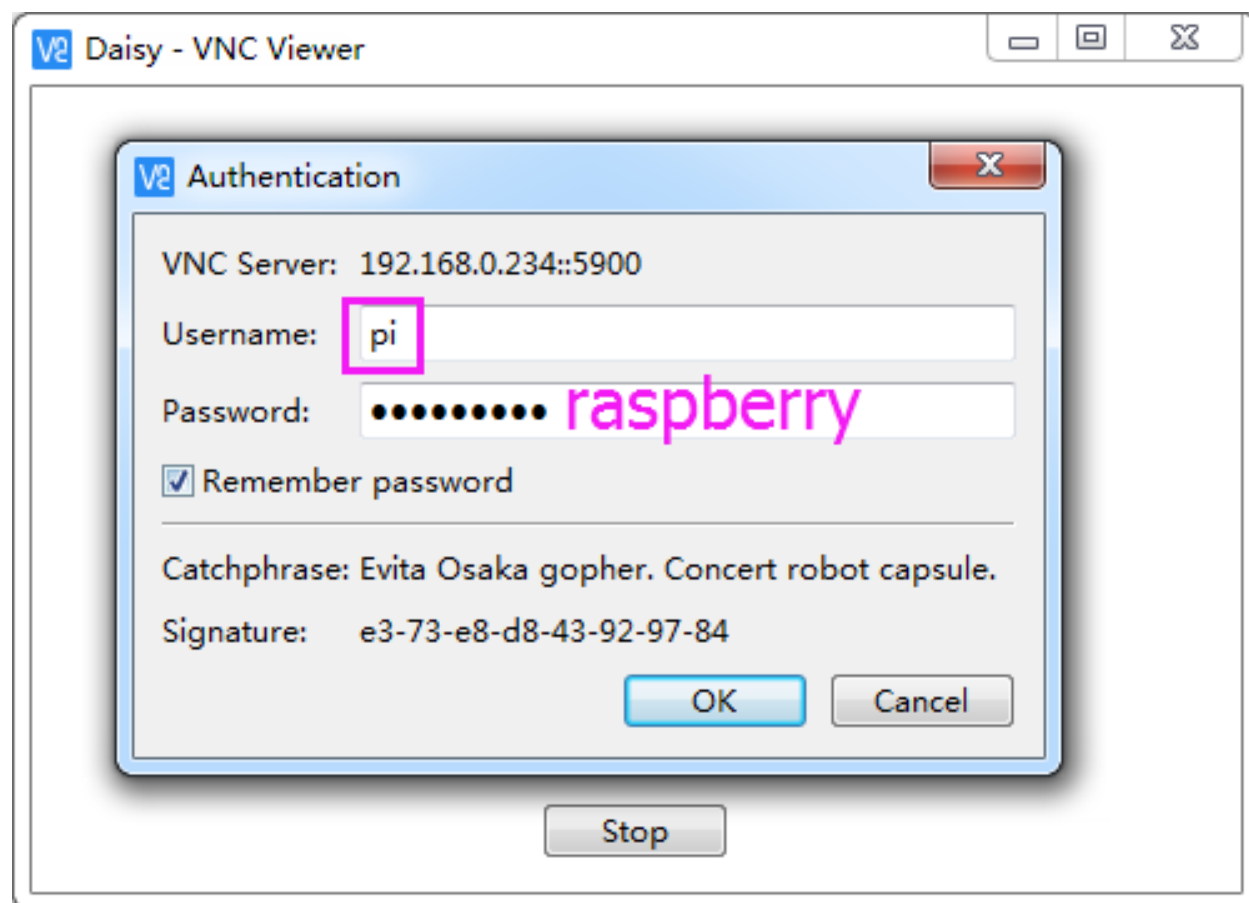
第 4 步

双击刚刚创建的 连接。



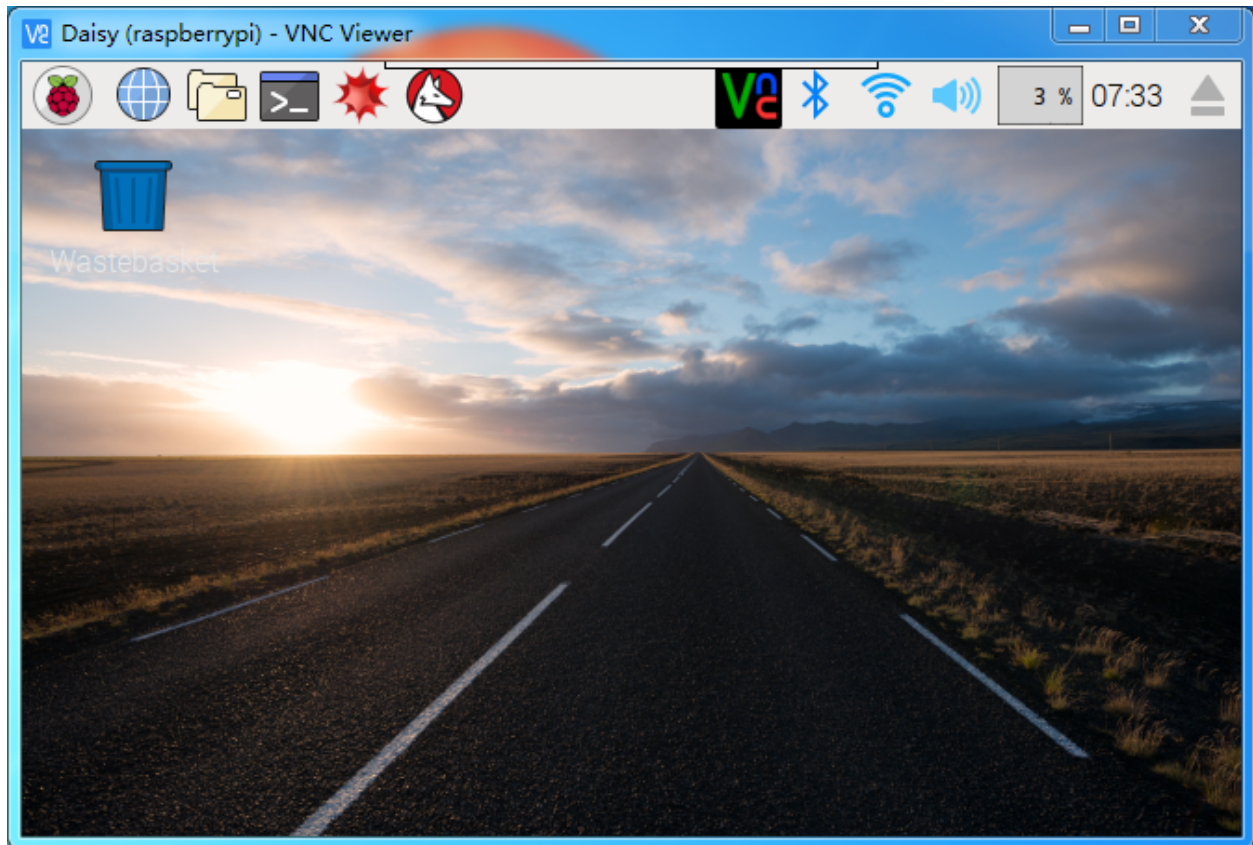
第 5 步

输入用户名 (**pi**) 和密码 (默认为 **raspberry**)。



第6步

现在你可以看到树莓派的桌面了。



VNC 部分就到此为止了。

5.5.2 XRDP

另一种远程桌面的方法是 XRDP，它使用 RDP（微软远程桌面协议）提供图形化登录到远程机器。远程桌面协议）。

安装 XRDP

第 1 步

通过使用 SSH 登录到树莓派。

第 2 步

输入以下说明来安装 XRDP。

```
sudo apt-get update
sudo apt-get install xrdp
```

第 3 步

后来，安装开始了。

按（“Y”），然后按（“Enter”）来确认。


```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo apt-get install xrdp
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
Suggested packages:
  vnc-java mesa-utils x11-xfs-utils
The following NEW packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
  xrdp
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,468 kB of archives.
After this operation, 17.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y

```

第 4 步

安装完成后，你应该使用 Windows 远程桌面应用程序登录到你的树莓派。使用 Windows 远程桌面应用程序。

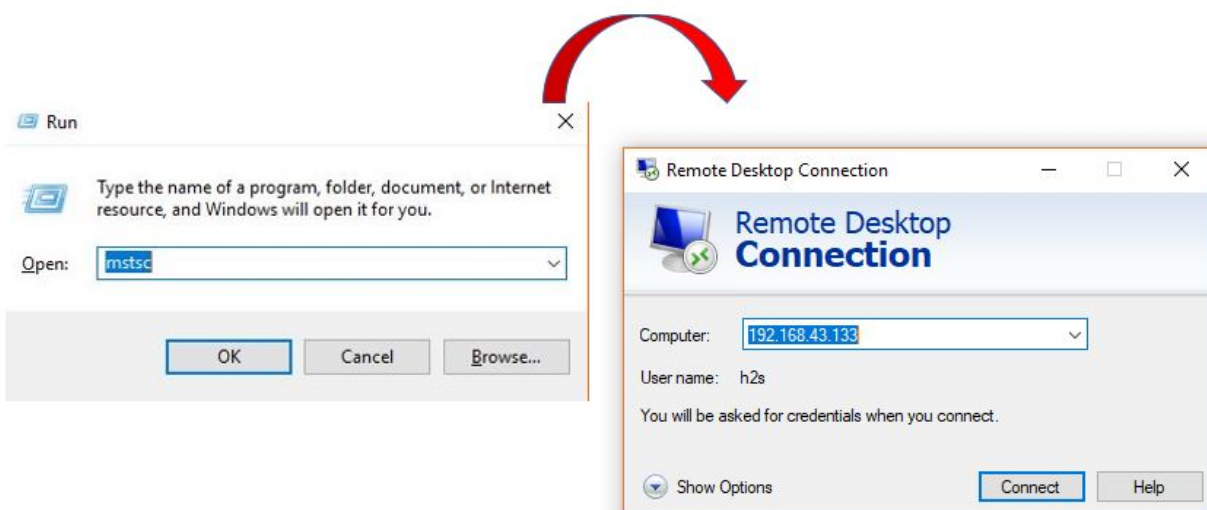
登录到 XRDP

第 1 步

如果你是一个 Windows 用户，你可以使用 Windows 自带的远程桌面功能。自带的远程桌面功能。如果你是 Mac 用户，你可以从 APP Store 下载并使用微软远程桌面，两者之间没有太大区别。两者之间没有什么区别。接下来的例子是 Windows 远程桌面。

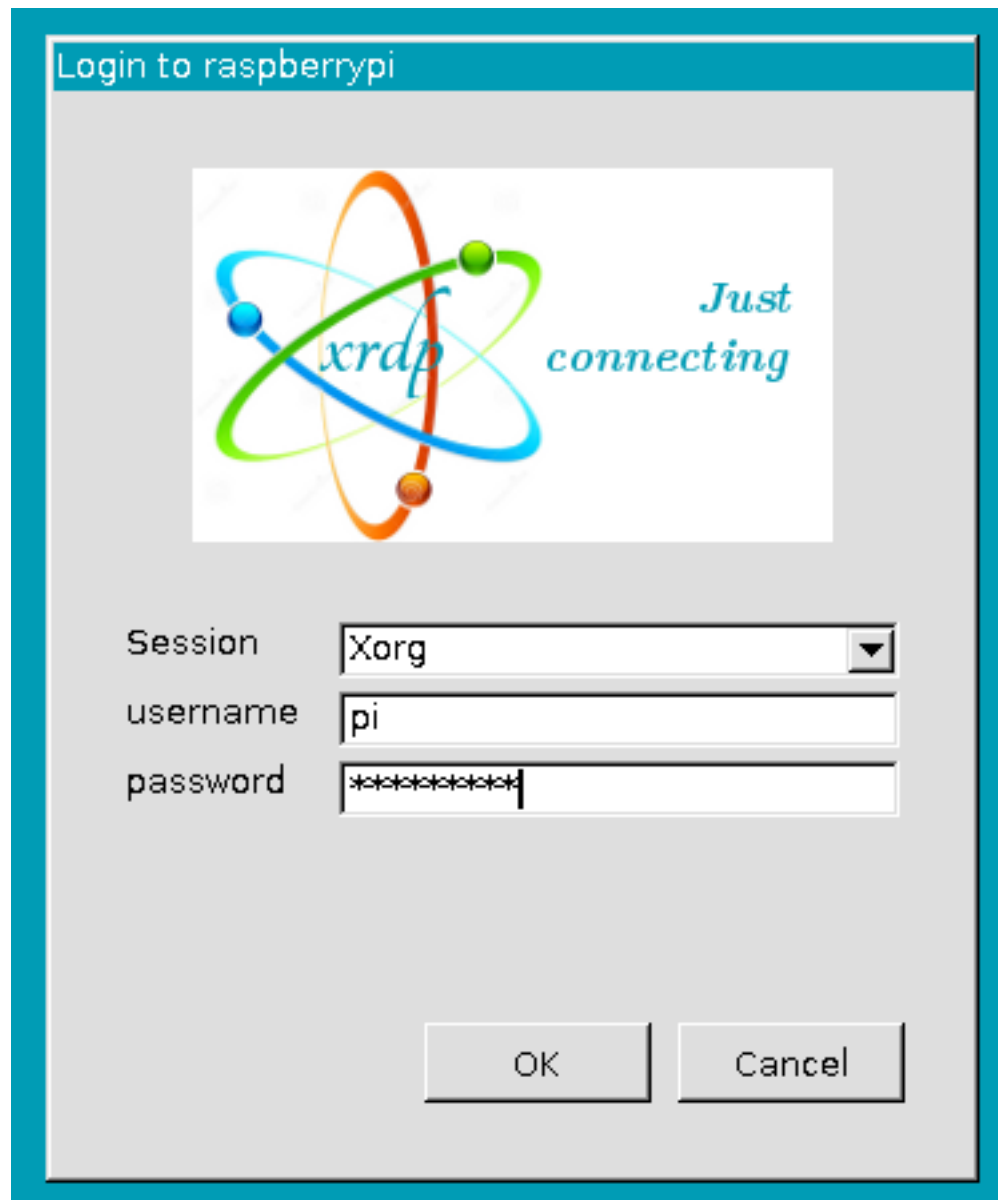
第 2 步

在运行（WIN+R）中键入“mstsc”，打开远程桌面连接，并输入树莓派的 IP 地址，然后点击 Connect”。



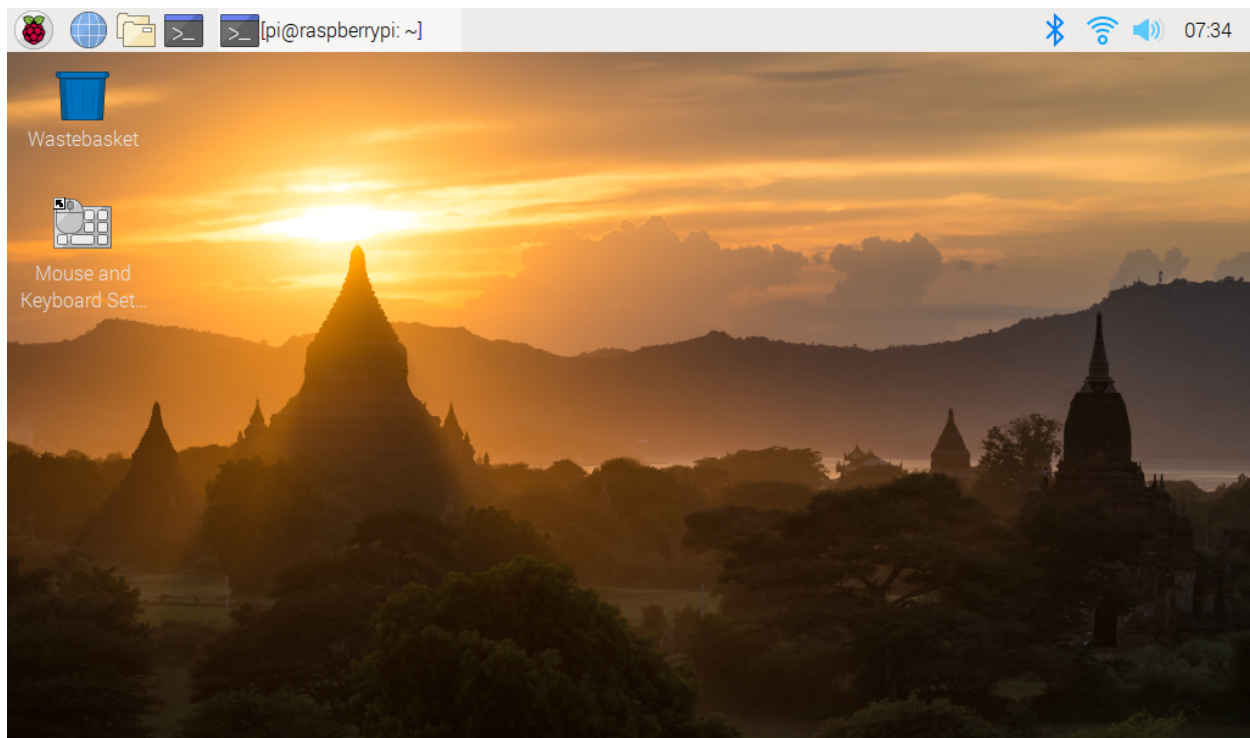
第 3 步

然后弹出 xrdp 登录页面。请键入您的用户名和密码。之后，请点击”OK”。在你第一次登录的时候。你的用户名是”pi”，密码是”raspberrypi”。



第4步

在这里，你通过使用远程桌面成功登录到 RPi。



5.6 关于电池

适用参数

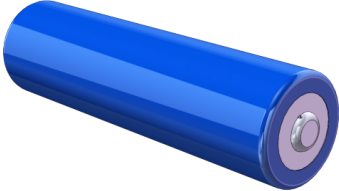
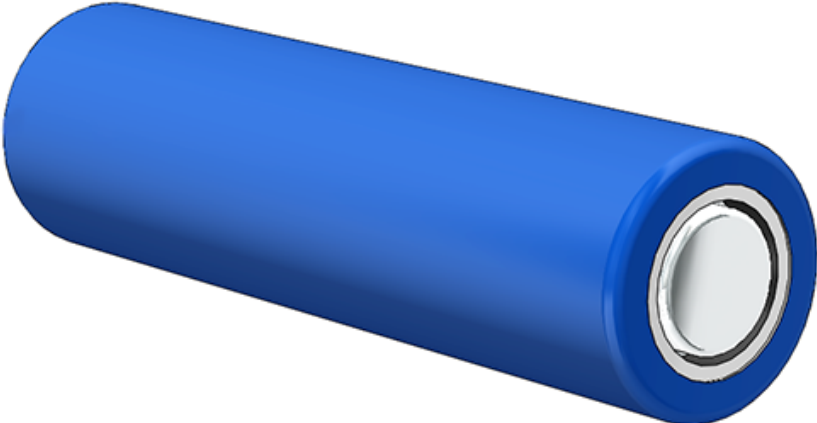
- 3.7V
- 18650
- 可充电的
- 锂离子电池
- 尖头电池
- 无保护板

注解:

- Robot HAT 无法给电池充电，需要购买电池充电器。
- 当 Robot HAT 上的两个电量指示灯熄灭时，表示电量过低，需要给电池充电。

尖头 vs 平头？

请选择尖头电池，以确保电池与电池座之间的连接良好。

尖头	平头
	

没有保护板？

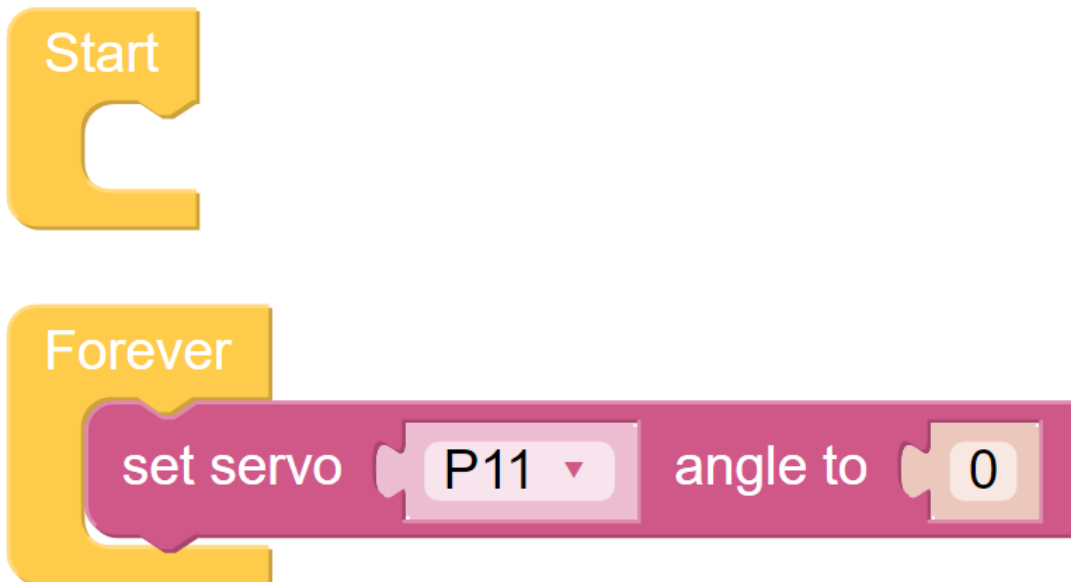
建议使用没有保护板的 18650 电池。否则可能会因为保护板的过流保护而导致机器人断电停止运行。

电池容量？

为了让机器人长时间工作，尽量使用大容量电池。建议购买容量为 3000mAh 及以上的电池。

6.1 安装 EzBlock 操作系统后，舵机不能转到 0° ？

- 检查伺服电缆是否正确连接，Robot HAT 的电源是否打开。
- 按下复位按钮。
- 如果您已经在 EzBlock Studio 中运行了程序，那么 P11 的自定义程序就不再可用。您可以参考下图，在 EzBlock Studio 中手动编写程序，将伺服角度设置为 0。

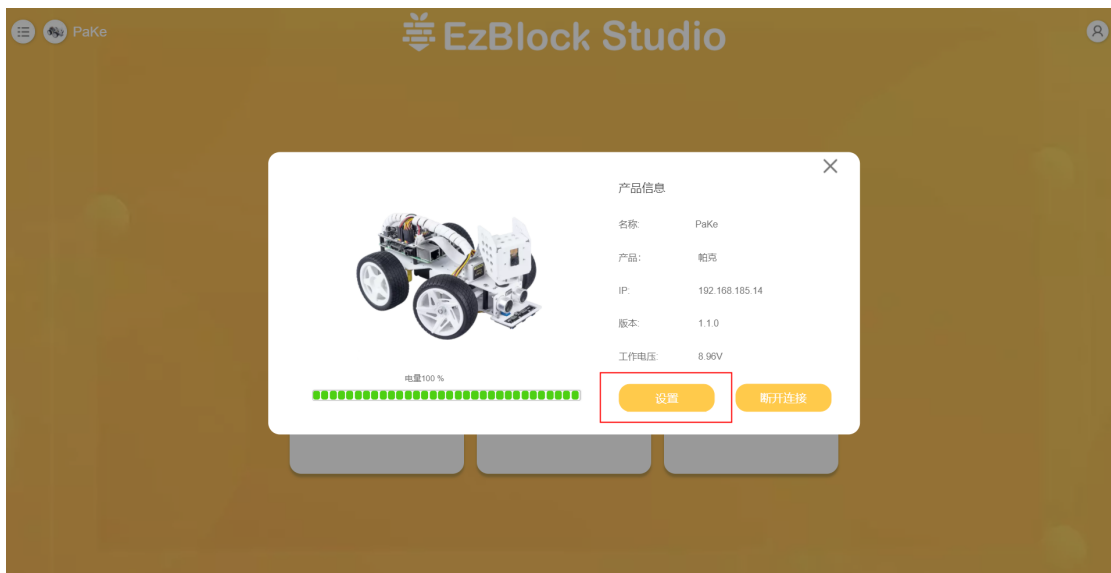


6.2 如何在 EzBlock 重新校准机器人

在首页，点击左上角的产品连接图标。



进入到产品页面之后，点击 **设置**按钮。



进入设置页面后，点击 **校准**按钮就能进入到校准页面，然后按照提示来校准。



6.3 EzBlock 无法连接蓝牙？

- 先检查是否有给你的 SD 卡烧录安装 [EzBlock](#) 镜像。
- 打开产品的电源开关后，等蓝牙指示灯变得更亮并且出现“zi~”声后，代表树莓派成功启动，此时再去连接。
- 检查您的移动设备的蓝牙是否打开。
- 检查 EzBlock 是否被允许访问设备的位置。
- 有些移动设备还需要打开位置服务。
- 检查电池电量。如果两个电源指示灯都关闭，或者只有一个指示灯在闪烁；电源电量低，请给电池充电。
- 如果以上方法都试过了，请尝试按下 RST 按钮，或重新启动产品和 APP。

6.4 APP 搜索到蓝牙，但无法连接

- 先检查是否有给你的 SD 卡烧录安装 [EzBlock](#) 镜像。
- 打开产品的电源开关后，等蓝牙指示灯变得更亮并且出现“zi~”声后，代表树莓派成功启动，此时再去连接。
- 检查 ROBOT HAT 上的 BLE 或 USR 灯是否一直亮着（这意味着产品被其他设备连接），如果是，请断开其他设备的连接或重新启动产品。
- 如果以上方法都试过了，请尝试按下 RST 按钮，或者重新启动产品和 APP。

6.5 配置 WIFI 后 APP 无法连接

- 检查国家、账号和密码是否正确。
- 检查该 WIFI 的网络状态。
- 检查电源电量。如果两个电源指示灯都熄灭或只有一个电源指示灯闪烁，则说明电源电量不足，请给电池充电。
- 检查配置的 WiFi 和移动设备连接的 WiFi 是否相同。

6.6 为什么伺服机有时会无缘无故地返回到中间位置？

当舵机被结构物或其他物体挡住，无法到达预定位置时，舵机会进入断电保护模式，以防止舵机被过大的电流烧坏。

断电一段时间后，如果没有给伺服机提供 PWM 信号，伺服机将自动恢复到原来的位置。

版权声明

本手册中包括但不限于文字、图片、代码等所有内容均归 SunFounder 公司所有。根据相关规定和版权法，你只能将其用于个人学习、调查、欣赏或其他非商业或非营利目的，不得侵犯作者和相关权利人的合法权利。对于任何个人或组织未经许可将其用于商业利益，本公司保留采取法律行动的权利。