Botball Link 控制器使用手册

文件名	Botball Link 控制器使用手册	版本号	LC_2014_1.2
更新日期	2014/1/11	版权所有	Botball 中国组委会

目　录

1. KIPR Link 机器人控制器

1.1 关于 KIPR Link

KIPR Link 是一个基于 Linux 的机器人控制器。它由美国 KISS Institute for Practical Robotics（KIPR）设计。用户可以通过在电脑上安装 KISS IDE(同样由 KIPR 设计和维护的集成开发环境)来编程使用 Link。

Link 的硬件拥有显著的特色和超过前版本的易用性能。Link 不仅对于机器人的初学者来说是一个易于上手的选择，同时对于机器人专家也是一种强大的、富有特色的设备。

1.2 Link 的基本特色

- 基于 GNU/Linux 的操作系统
- 开源的机器人控制软件
- 集成了颜色视觉系统
- 800M Hz ARMv5te 处理器
- Spartan-6 FPGA
- 集成锂电池和充电系统
- 内置扬声器
- 320 x 240 像素的彩色触摸屏

1.3 输入和输出

- 1 个三轴 10 位的加速度计（可软件选择 2/4/8g）
- 8 个数字 I/O 口（硬件选择 3.3V 或 5V）
- 8 个 3.3V（最高 5V）10 位精度的模拟输入端口
- 4 个舵机端口
- 4 个带 10 位反电动势和 PID 电机控制的电机端口
- 1 个 3.3V（最高 5V）的 TTL 串口
- 2 个用于连接设备的 USB 2.0(A 型)主机端口
- 1 个硬按键
- 1 个红外发射器

- 1 个红外接收器
- 1 个 HDMI 视频输出端口

1.4 其它特色

- 所有传感器的输入均有软件使能的上拉电阻（数字传感器 47k,模拟传感器 15k）
- 电机端口的最高输出电流达 1A
- 舵机端口输出 6V 电压
- I2C 接口（需要附加硬件）
- Arm 7 的调试端口（需要附加硬件）
- JTAG 接口
- V_{CC} 的最大电流 500mA(3.3V),1A(5V)
- 7.4V,2000mAH 的锂离子电池
- 1GB 容量 SD 卡
- 内置 802.11b/g wifi

1.5 Link 所含硬件

KIPR Link 的基本硬件包括 Link 控制器、USB 线、电源适配器和 USB 摄像头。在充电时，应有成年人的照看，再次充满电需要约 90 分钟。

注意：只能使用提供的充电器或相同供应商产的充电器来为 Link 充电，使用不适当的充电器可能损害 Link，因此造成的后果不属保修范围。

KIPR Link

下载线

（micro USB）

USB 摄像头

电源适配器

13.5v 1000mA
REGULATED (switching)

1.6 Link 图解说明（英文）

彩色触摸屏 (320x240 dpi), WiFi, 2000 mAh 锂聚合物电池, 800mhZ ARMv5te CPU, FPGA, Linux 操作系统

俯视图

8 个模拟端口, 8 数字 I/O 端口, 4 个马达端口, 4 舵机端口

HDMI 视频接口

左视图

前视图

电源开关 红外 发射/接收

右视图

定位孔与乐高兼容

仰视图

TTL 串口

micro USB 接口
(用于连接电脑)

USB2 端口(U 盘, 摄像头,
鼠标,键盘)

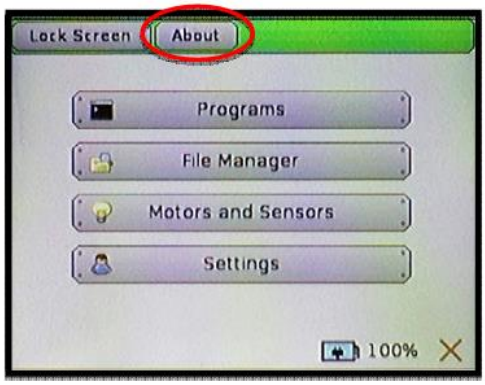
电源输入

后视图

2. 快速启动

2.1 开启 Link

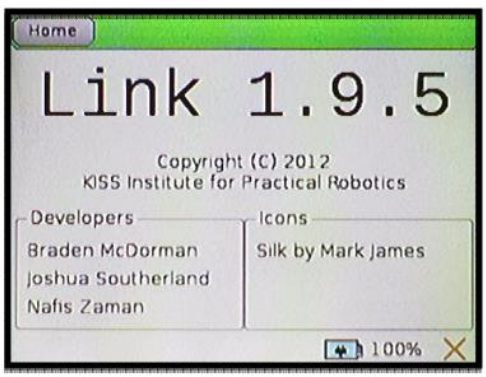
将电源适配器插入 Link，在电源插口附近将出现一个绿色的电源适配器连接指示灯和一个彩色的充电状态指示灯。将电源开关打到 ON 位置以引导 Link 开机。Link 将花费 40~45 秒以进入用户界面。当 Link 启动完毕， 将出现如下的屏幕界面。



2.2 检查 Link 的固件版本

Link 由一个叫做“Link Firmware”（固件）的软件进行控制。在 http://www.kipr.org/kiss-platform-link-firmware 网页处发布了此软件的最新版本，并提供下载。

用户可以使用 U 盘更新 Link 的固件程序，方法为：开启你的 Link，点击主界面上的“About”（关于）按钮，可以检查当前在 Link 上运行的固件版本。如果固件版本不是最新，可以去上述网站下载固件镜像压缩文件到你的电脑上，然后按照网站上的教程或本手册附录 6.1 中的方法更新你的 Link 固件。



2.3 安装 KISS 平台——KISS IDE

请安装下述指导完成系统安装。

Mac（操作系统版本 10.4 及以上——64 位处理器）

兼容 Mac 系统的 KISS 平台可以通过如下网站下载：

http://www.kipr.org/kiss-platform-mac-os-x.

首先，**为你的操作系统安装最新版本的 Xcode**。你可以在 OS X 的安装光盘里找到 Xcode 的安装程序，或者从苹果官网上进行下载 http://www.kipr.org/kiss-platform-mac-os-x.（你必须先注册一个免费账号用于登录并下载所需软件）。

如果你使用的版本是 OS X 10.8(Mountain Lion)，你需要自行改变系统设置，令系统允许 KISS IDE 作为一种不需核对的应用程序来运行。请参看苹果网站上的说明 http://support.apple.com/kb/HT5290.

然后，双击 KISS-x.x.x.dmg 文件来挂载磁盘映像，并**将磁盘映像里的 KISS-C 文件夹复制到苹果里的应用文件夹里**。你需要将 KISS-C 应用程序和库文件夹放置于同一个 KISS-C 文件夹里（你自己编写的程序可以保存于任何位置）。

无需安装 USB 驱动；OS X 自带相应的驱动程序。

Windows(XP,Vista,7)

兼容 Windows 系统的 KISS 平台可以通过如下网站下载：

http://www.kipr.org/kiss-platform-windows.

如果你使用的是 XP 系统，安装时需要使用配套的 USB 线将 KIPR Link 连接到电脑上。对于 Windows Vista 和 Windows 7，不需将 Link 连接到电脑上即可安装。

双击 KISS-C 安装文件(KISS-x.x.x.exe)，打开 KISS-C 安装向导。选择 Next 开始进行安装。Windows 8 需要先对系统进行一下设置更改，详见

http://www.kipr.org//kiss-platform-windows.

在”*Choose Components*（选择组件）”界面，如果你以前从未进行过软件和驱动的安装，确保将 KISS-C 与 Link drive 选项同时选上。然后单击“下一步”以选择安装位置。推荐安装在默认位置处，如 Program 文件夹里。单击”*Install*(安装)”进行安装。

现在驱动已开始安装，并提示你驱动未经认证。选择 *install anyway*(继续安装)选项。在 XP 系统里，你可能会被提示需要安装驱动，这时需要单击“*Next(下一步)*”，接着再次选择“*Next（下一步）*”以使系统搜索并安装驱动。

KISS-C 现已出现在你的程序列表里。桌面也将出现 KISS-C 的快捷方式。

Windows 8

请按下述步骤安装系统，在未被要求连接 LINK 之前不得将 LINK 连接上电脑！

相应版本的 KISS 平台可以通过如下网站下载：

http://www.kipr.org/kiss-platform-windows.

按以下步骤在 Windows 8 下安装 KISS-C:

1. 正常启动 Windows 8，进入开机引导
2. 在右上角单击搜索
   a) 搜索“Settings”
   b) 单击”*Settings*（设置）”
   c) 搜索“General”
   d) 单击”*General Settings*（一般设置）”
3. 选择”*Advanced Startup*（高级启动）”,单击”*Restart Now*（现在重启）”
4. 单击”*Troubleshoot*（故障排除）”
5. 单击”*Advanced Options*（高级选项）”

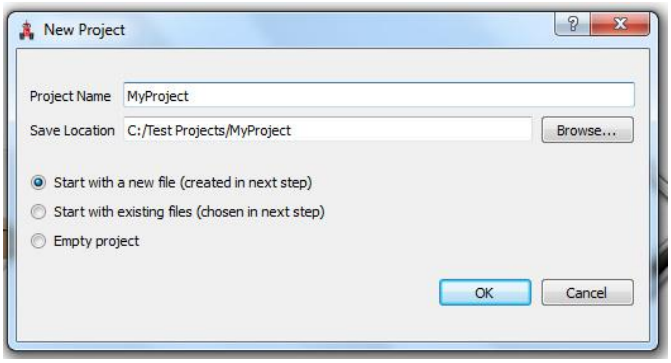
6. 单击”*Startup Settings*（开机设置）”
7. 单击”*Restart*（重启）”
8. 选择选项 7
9. 登陆 Windows 8
10. **此时，使用 USB 线，将 KIPR Link 连接到电脑**
11. 打开 Link
12. 双击 KISS-C 安装文件，同上述其它 Windows 版本的安装方法一样进行 KISS IDE 的安装
13. 搜索“设备管理器”
    a) 你将发现“Gadget Serial v2.4”
    b) 右击”*Gadget Serial v2.4*"，选择*更新驱动*
    c) 选择自动搜索
    d) 出现 COM 端口号意味安装完成(端口号小于 100 均能使用)

2.4 下载并运行一个 Link 程序

1. 连接：使用套件提供的 USB 线将 Link 同电脑连接。
2. 运行 KISS IDE：点击 IDE 启动图标，出现启动界面

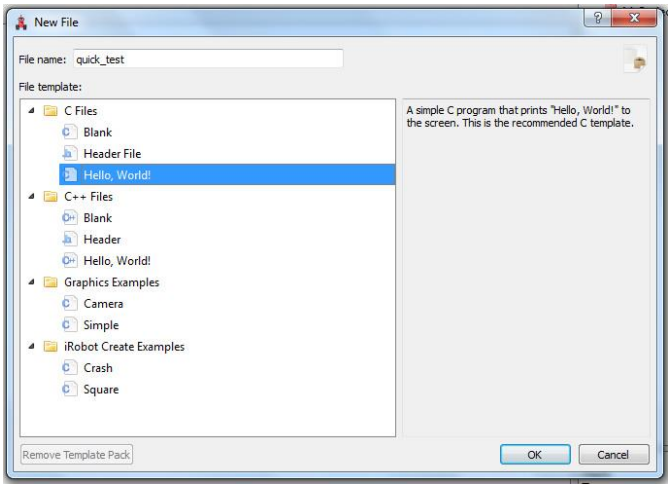
3. 创建新的工程

*“New Project ”(新工程)*对话框出现后，键入工程名称并设置保存位置。



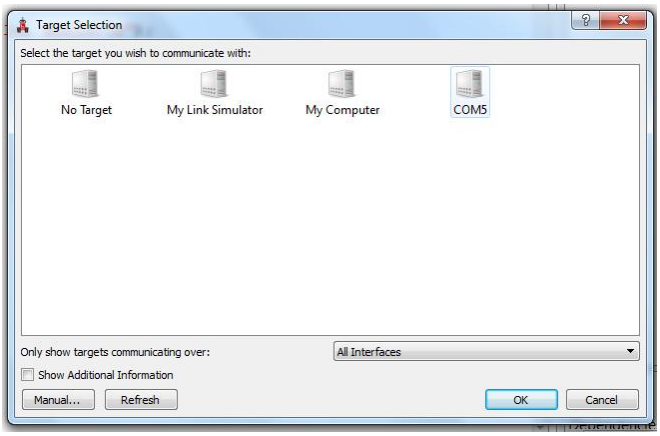
4. 为新工程添加一个文件

New File(新文件)对话框出现后，键入文件名，并选择”Hello, World!”C 语言文件。如果你的文件名中没有.c 后缀，系统会自动添加。注意：不要在文件名中包含其它的”.”符号。



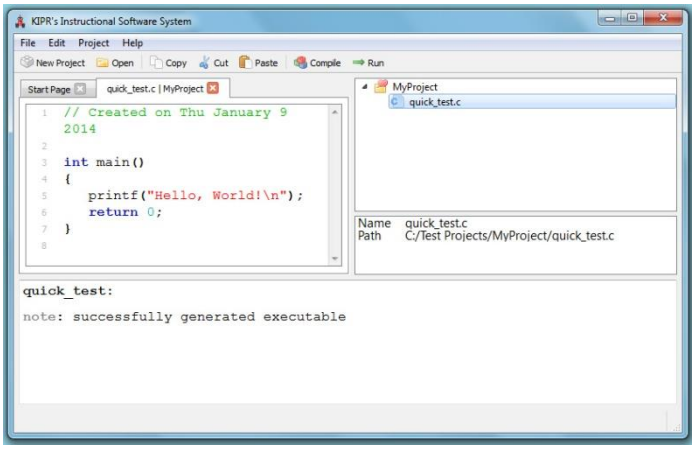
5. 选择通信目标

出现“*Target Selection*”（*目标选择*）对话框，为你的 Link 选择与电脑连接的目标的通信端口（对于 Windows 操作系统通常都是最高端口号）。下图中包括了 *No Target(无目标)*，*My Link Simulator(仿真器)*，*My Computer(我的电脑)*和 *COM5(5 号串口端口)*等选项，可能的选项还有连接到无线网的 Link 设备名。

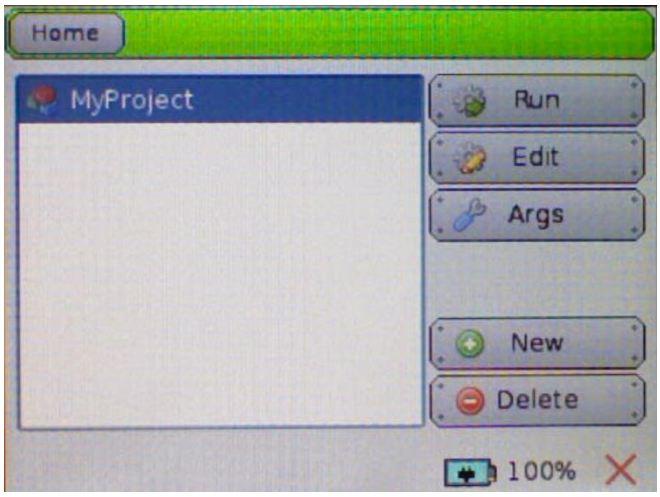


6. 编辑/保存/下载程序

文件在 IDE 中显示为一标签窗口，并可被编辑、保存、编译、下载到 Link 中。使用 *File…Save As…*将当前程序保存到电脑磁盘中。“*Compile*”*（编译）*或“*Run*”*（运行）*时将自动保存文件，同时要求用户为文件命名。如果目前没有目标，目标选择对话框将出现。其中，如果选择了 Link 设备作为目标，*编译*和*运行*都将程序发送到 Link 中并编译。

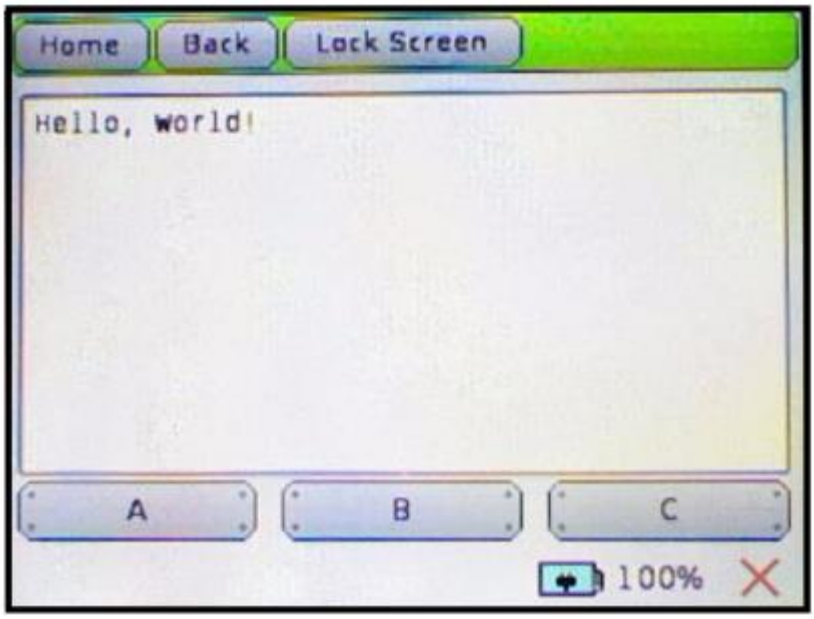


7. 打开 Link，并在其中找到程序文件：在 Link 的主界面中单击“*Programs*”*(程序)*，程序将以和电脑 KISS IDE 中一样的工程名出现。单击文件，将其高亮显示。



8. 在 Link 中编译并运行程序

单击 *“Run”（运行）*，程序将被编译并运行。如果编译出错，编译产生的错误将出现在结果界面。如果成功，将直接得到运行。



如果你顺利完成了上述步骤，恭喜你！你已经验证了 Link 和 IDE 是完好的，并能够用它们来开发自己的程序了。

2.5 本手册后续章节内容

本手册接下来的部分将包括以下内容：

- 讲述如何使用 Link 库函数和 Link 驱动器和传感器的用户接口。
- 讲解关于怎样调用 Link 库函数来使用摄像头。
- 故障解决手册包括了一些用户使用 Link 中常见的问题及其解决方法。

- 附录提供了对部分用户有用的信息，包括 Link 固件的更新、驱动器和传感器的特殊函数、USB 摄像头的使用、以及 iRobot Creat 的使用。

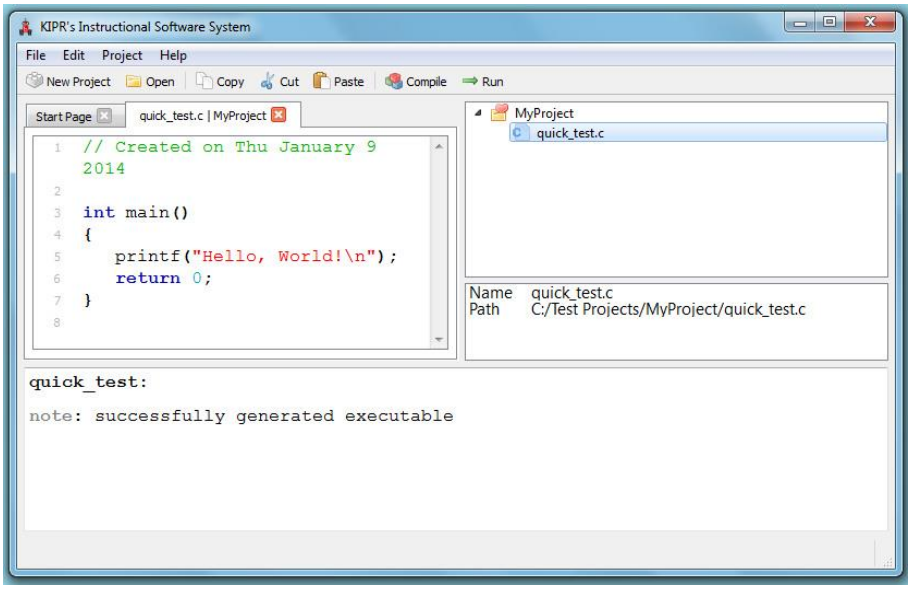
3. KIPR Link 编程

3.1 在 Link 和 KISS IDE 中使用 C 程序语言

C 语言是最被广泛使用的高级语言。Link 提供了一组库函数供用户调用，以使用其硬件的各种功能。Link 和 KIPR IDE 中所使用的 C 编译器是内置的 Linux ANSI C 编译器（本文档中我们不讨论 Link 支持的其它高级语言的编译和使用方法）。IDE 中的帮助文档提供了使用 C 语言编程的在线指导，也包括对特殊函数库的说明文档，以及 KIPR 的驱动器和传感器的图片信息。IDE 中还包含一个仿真器，此处也不做讨论。

更多的关于 ANSI C 编程语言资料可以参阅相关书籍。我们推荐给初学者的书是 Greg Perry 写的《Absolute Beginner's Guide to C》（第二版）和 Brian W.Kernighan,Dennis M.Ritchie 写的《C Programming Language》（中文版为《C 程序设计语言》）。IDE 在线文档中也包括一个简单的 C 教程，它非常适合于那些已经熟悉另一种编程语言但并不熟悉 C 语言的用户。

请采取第 2 章中介绍的步骤，当进行到“编辑/保存/下载程序”这一步时，即可开始为 Link 编程。选择使用一个 C 语言模板，你的工程将会出现于 IDE 的用户界面中。点击旁边的小三角标志，所有处于此工程的文件都会被列出。点击这些文件，它们会在旁边的编辑窗口中被以一个标签窗口的形式打开。每次重复上述步骤都将在 IDE 中得到一个新的标签窗口。你可以通过点击标签在几个程序文件之间进行切换，以及在工程间切换。当前显示的文件就是正在被编辑的程序文件，它所在的工程就是当前正在活动状态的工程。



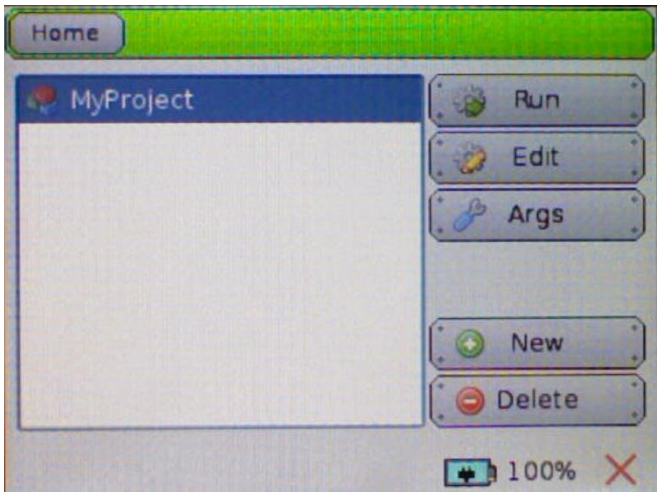
在模板的基础上，用户可按自己编写程序的需要进行各种编辑修改。当你认为编程工作告一段落时，就可以对程序进行测试了。单击 IDE 上的 *“Compile”（编译）* 按钮以检查程序是否有语法错误。你可以先选择 *“Simulator(仿真器)”* 作为对象来对程序进行测试，然后再实际下载到 Link 中。在此之前，需要先对程序文件进行保存，并对之命名。注意：每次编译时，工程都会自动进行保存。

在 IDE 窗口的底部面板将显示编译发现的错误，应从第一条错误开始检查程序。双击错误，光标将跳转到错误所在的程序行，错误发生在该行或该行之前（如在该行以上缺少一个分号“;”）。很可能发生的是，虽然报错很多，但当你修改了第一个错误并重新编译时程序就没有其它错误了，因为前面的错误可能会引起编译器对后续程序的报错。

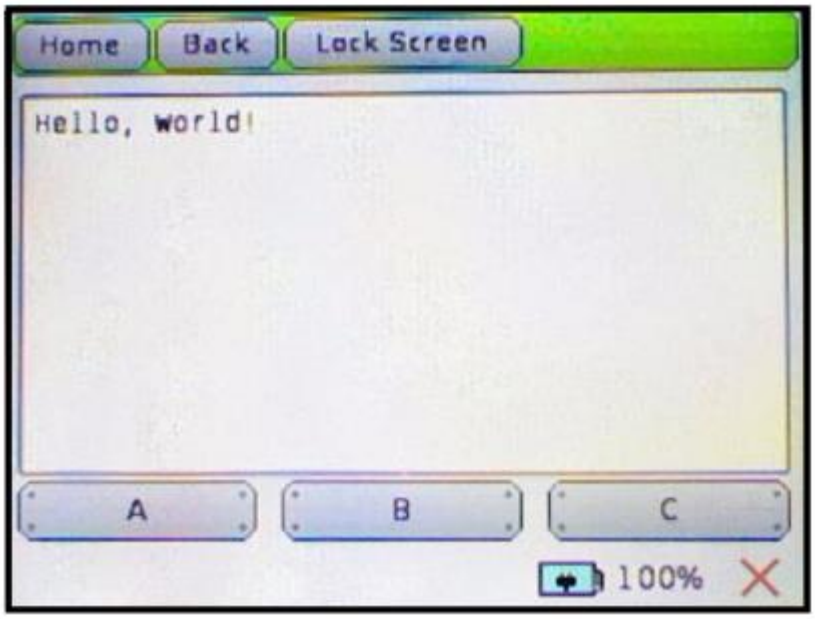
如果希望不编译而直接将程序下载到 Link，请右击工程名，并选择 *“Download（下载）”*。因为程序需要在 Link 中进行编辑，所以这样做只是将工程拷贝到 Link 目前的步骤只是将程序文件下载到 Link 里的程序路径下。

在 Link 控制器中查找和运行程序的方法在第 2 章中已经介绍了，这里再简单重复一下：

1. 在 Link 中找到程序文件。在 Link 的主界面中单击 *“Programs”（程序）*，程序将以和 IDE 中一样的工程名出现。单击文件，将其高亮显示。



2. 在 Link 中编译并运行程序。单击*“Run”（运行）*，程序将被编译并运行。如果编译出错，编译产生的错误将出现在结果界面。如果成功，将直接得到运行。

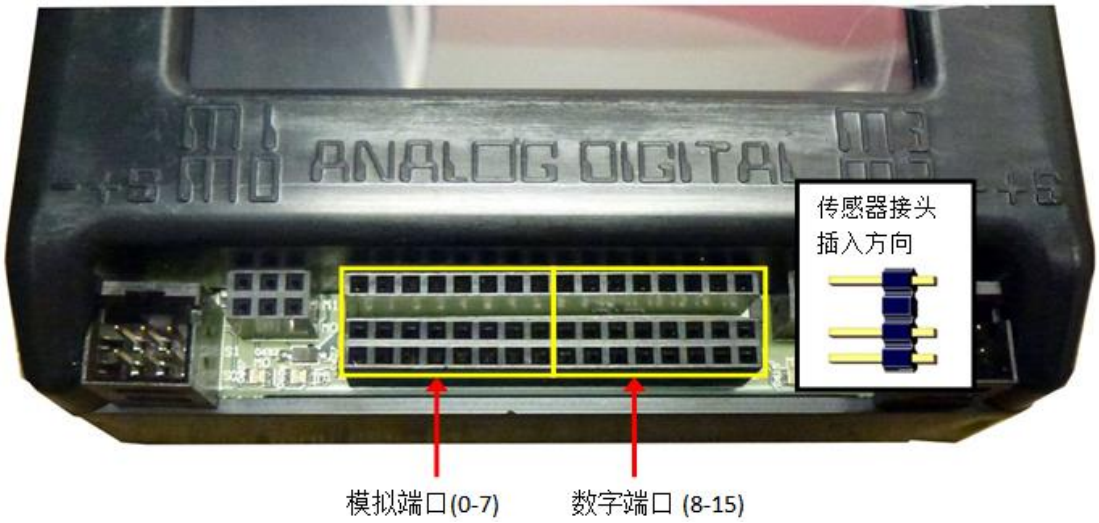


3.

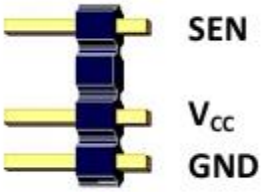
3.2 Link 函数库

函数库由 Link 固件提供，通过调用函数库以使用 Link 的各种外设。一些更常用的函数下面将讲述，更全的函数列表将在附录中给出。

3.2.1 使用传感器



Link 中有 8 个模拟传感器端口(analog ports)和 8 个数字传感器端口（digital ports）。BotBall 官方套件中的任何传感器都能用于 Link 上。它们的连接都具有方



向性：

数字传感器通常只有两根线，这样当传感器被触发时，*SEN（信号）*和 *GND（地）*构成回路。模拟传感器有两个或三个线。对于一个模拟传感器来说，位于 *SEN* 和 *GND* 之间的电阻是变化的。第三根线用于连接 *Vcc（电源电压）*来为需要电源的传感器提供电源（如反射式红外传感器有一个红外发射器需要提供电源才能工作）。

在 Link 中，*Vcc（电源电压）*与 *GND（地）*间的电压是+5V(无论数字端口还是模拟端口)。*SEN* 和 *GND* 间的最高电压也是 5V(无论数字端口还是模拟端口)。+5V 的设置是通过内部跳线来完成的。

注意：如果打开 Link 来更改跳线，以此造成的后果将不在保修之内。如果确实需要更改，请联系 KIPR 技术支持。通过更改跳线帽来使用自购的传感器的方法，请参阅附录里的相关内容。

3.2.2 Link 传感器库函数

使用传感器的两个最基本函数：analog10, digital：

- **analog10(<port#(端口号)>)**

模拟传感器根据 *SEN* 和 *GND* 之间的电阻值变化而产生一个变化的电压值。analog10 函数即返回该端口的模拟电压值（范围从 0~1023）。模拟端口标号为 0~7。光传感器和测距传感器是使用模拟传感器的两个例子。下例是显示接入模拟端口 3 的光传感器的读数的用法：

```
printf("Light sensor reading is %d\n", analog10(3)); //  显示 3 号模拟端口读数
```

- **digital(<port#(端口号)>)**

数字传感器如同一个开关一样，快速的在 SEN 和 GND 之间产生两种电阻：0（开关闭合）和∞（开关断开）。如果开关闭合，digital 函数返回 1；如果开关断开，digital 函数返回 0。在 Link 中，数字端口标号为 8~15。触觉传感器是典型的数字传感器，通常用于碰撞或极限位置的转换。下例是插入数字端口 8 的一个按键传感器的用法：

```
if (digital(8)==1)    //  如果端口 8 上的读数是 1
    printf("button is being pressed\n");    //  显示按键被按下
else
    printf("button is not being pressed\n"); //  否则显示按键没有被按下
```

上述两种函数的函数原型分别是：

- **int analog10(int port_no);**
- **int digital(int port_no);**

传感器通常需要一个上拉电阻来使其正常工作。每一个 Link 传感器端口都可以用软件选择使用上拉电阻（这是由 Link 固件决定的默认设置，对于一般传感器不需更改）。

但对于如 ET 距离传感器，由于其本身已经有内置的上拉电阻，因此若要使用该传感器，该端口便不能再设置上拉电阻，必须关闭该端口的上拉电阻。使用函数 **set_ analog_pullup** 来进行上拉电阻的端口设置。

- **set_analog_pullup(<port#(端口号)>, 0(浮动)/1(上拉))**

如模拟端口 4 接有 ET 传感器，应在使用前进行如下设置：**set_analog_pullup (4, 0)**，即关闭端口 4 的上拉电阻。术语“浮动”用于描述一个端口没有使用上拉电阻，“上拉”即使用了上拉电阻。由于“上拉”是 Link 的默认设置，因此未经设置为“浮动”就通过 analog10 函数读取的 ET 传感器值是无意义的。

在附录里还有一些关于 Link 其他传感器的使用函数。包括读取边侧按钮和软按钮 A, B, C, X, Y, Z(出现在 Link 的控制面板屏幕上)的按钮函数。

3.2.3 使用舵机

如上图所示，舵机接在 Link 的舵机端口。上图中的箭头表示最常见的舵机各线的含义（黑线接地，红线接电源，黄线为信号线。当然，有舵机线的颜色不同于此。）。在插入舵机之前应先检查舵机与端口是否兼容。舵机端口的工作电压为 7.2V。由于舵机驱动函数的功能是转动舵机到指定位置并保持不动，因此舵机需要不断的提供电压。随着 Link 电池输出电压的下降，若已经低于舵机正常工作

需要的最低电压，舵机将无法正常工作，这时需要给电池充电。

3.2.4 关于舵机的 Link 库函数

当 Link 不使用舵机时，应关闭舵机端口以节约电源。默认情况下，舵机端口是处于关闭的。有四个舵机端口，标号为 0~3。下例函数管理舵机端口及操作舵机：

- **enable_servo(<servo_port#（舵机端口号）>)**

指定舵机端口使能。当舵机端口使能后，如将舵机接入该端口，舵机将运转到该端口上次设置的角度位置，默认是舵机角度的中间值为 1024。

- **disable_servo(<servo_port#（舵机端口号）>)**

指定舵机端口关闭。当你想保持电池电力的时候，可以使用该函数关闭舵机端口。当舵机端口关闭，舵机的角度将由连接到它的外部结构（如重力）支配，且无法维持自己的角度位置。

- **set_servo_position(<port#（舵机端口号）>, <position（位置）>)**

令指定端口的舵机转动到指定位置上，位置范围是 0~2047。注意：位置范围在极限值 200 范围内（即数值小于 200 或大于 1847）的角度位置可能某些舵机是无法达到的。一旦使能（enable）舵机，如果没有外力阻挡或位置不可达到（此时舵机仍在提高电压，试图达到极限位置），舵机将迅速到达指定角度。如果此函数在调用舵机端口使能之前即被调用，则当端口使能后，舵机仍会立即转动到指定的角度位置。下例用舵机端口 2 连接舵机：

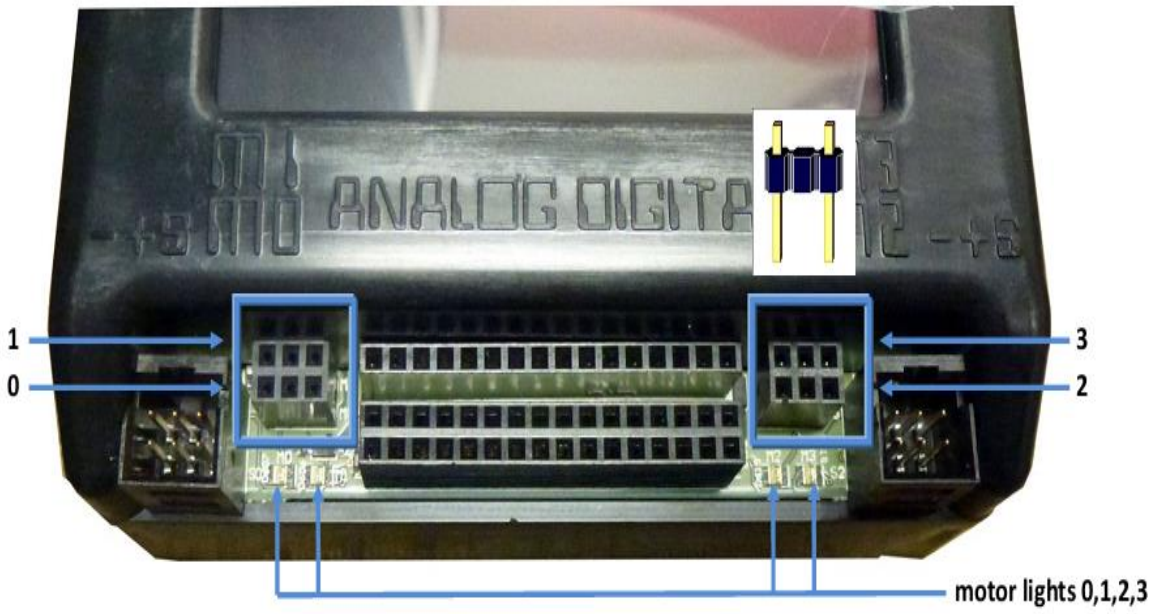
```
set_servo_position(2, 670);   // 为舵机指定位置为 670
enable_servo(2);   // 打开舵机(为其供电)
... 做其它工作的语句...
disable_servo(2);   // 关闭舵机
```

C 语言函数原型：

- **void enable_servo(int servo);**
- **void disable_servo(int servo);**
- **int digital(int servo, int position);**

在附录中有关于舵机的，其它使用频率较少的函数说明。

3.2.5 使用直流电机



BotBall 套件中的直流驱动电机有两针管脚，可以以任意两个方向插入 Link 的电机端口，电机端口的中间孔是不需使用的。改变电机的插入方向，会引起电机转向方向的改变。

直流电机转动产生的反电动势被外接到电机指示 LED 灯（motor light）上。因此，可以通过哪一个电机指示 LED 灯亮来判断该端口电机的转向：一个 LED 发绿光，另一个发红光。一般地，我们可以规定绿灯亮的方向是电机向前正转方向；如果此时将电机反向插入，则电机反转，红灯亮。

使用电机函数就可使电机转动。电机端口的最大电流是 1A，Link 通过改变 PWM（脉冲宽度调制）波的占空比来改变电机的平均电压，以此来获得电机的不同转速。最高电压为 5V。

每对端口（0 和 1、2 和 3）均由一个独立的 H 桥芯片控制。如果你的两个电机的电流输出都接近 1 安培，最好将两个电机插入不同组的端口里（如 0 和 2）。这样能减小芯片的发热，延长芯片使用寿命。

注意电机端口的中间孔是不使用的。Link 中的电机库函数为端口设置电压极性，如果电机向前转动，将点亮绿色 LED 灯；反之，电压极性相反，电机反转，

红色 LED 亮。若使用自制电机插头，请参阅附录中相关内容。

3.2.6 关于电机的 Link 库函数

- **motor(<motor#（电机端口）>,<power（驱动力大小）>)**

Link 以一定占空比的 PWM 波为电机提供一定的驱动电压，直到下一个电机命令到来才终止。**power** 的范围为 100（满速正转）至-100（满速反转）。PWM 代表“pulse-width-modulation”(脉冲宽度调制)是一种通过改变电机的平均驱动电压来控制电机转速的有效方法。

- **mav(<motor#（电机端口）>,<velocity（速度值）>)**

mav 代表 Move At Velocity，即以指定速度来驱动电机，直到下一个电机命令到来才终止。速度值从-1000 到 1000 ticks/秒。mav 函数（包括 Link 中其他的一些电机命令）使用比 PWM 波控制电机更复杂的控制方法来保证电机以指定速度运行。它通过获取电机运转时产生的反电动势作为反馈，通过 PID（比例微分积分）控制方法，动态改变 PWM 占空比，从而实现控制电机以指定速度运行。

- **mrp(<motor#（电机端口）>,<velocity（速度值）>,<ticks（相对位置）>)**

mrp 代表 Move Relative Position。电机以指定速度从到当前位置转动到当前位置加上指定 ticks 值的目标位置。速度范围是 0~1000ticks/秒。mav,mrp 均以 PID 算法控制电机。当指定位置达到或另一个电机命令施行时，则该电机命令终止（最终电机的位置可能由于电机运动的惯性而比指定的位置略微向前）。

- **bmd(<motor#（电机端口）>)**

如果电机正在执行一个位置命令（如 mrp），调用 bmd(block motor done)将令当前程序暂停等待，直到电机到达指定位置后才返回。注意，如果电机由于外部原因失速，该函数将暂停程序的执行，直到电机恢复可控状态。

- **ao()**

ao 代表 all off，关闭所有电机端口。

- **off(<motor#（电机端口）>)**

关闭指定的电机端口。

其它 Link 提供的电机函数的说明，请详见附录。

上述函数的原型为：

- void motor(int m,int p);
- void mav(int m,int vel);
- void mrp(int m,int vel,int ticks);
- void bmd(int m);
- void ao();
- void off(int m);

3.2.7 其他常用的 Link 库函数

- **msleep(<milliseconds(毫秒)>)**

该函数使程序暂停，当暂停时间达到指定的时间或稍比指定时间长后，程序继续往下执行。

- **printf(<string（字符串）>, . . .)**

标准 C 库函数用来格式化输出（即输出在 Link 控制器的控制面板屏幕上）。在 string 里出现%的地方，显示时均会被指定的数据以相对应的格式来代替，并显示在控制面板上。更多关于格式化输出的内容，可以参阅附录或有关 C 语言的书籍。

- **beep()**

产生短暂的蜂鸣声。

上述函数的函数原型为：

- void msleep(int msecs);
- int printf(char s[], ...); (返回值为 0 或输出写入到 s%的字符)
- void beep();

函数使用举例：

```
int t=1000, m=2;
mav(m, 500, 500); //驱动电机速度为 500ticks/秒
printf("Running m%d %d mseconds\n",m, t);  //输出信息，int 型数据使用 %d
msleep(t); //程序暂停 t 毫秒
off(m);//关闭 m 端口电机
beep(); //执行此语句会发出蜂鸣声响
```

电机将在一秒内移动 500ticks。与 bmd 函数不同，使用 msleep 将在 1 秒以后关闭该电机，即使该电机没有达到指定位置。printf 函数的输出将为：

Running m2 1000 mseconds

更多关于 Link 的库函数的内容，可参阅帮助里的在线文档或本手册附录。

4. Link 视觉系统

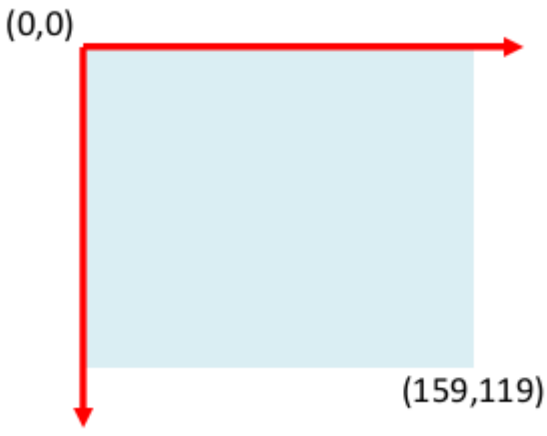
4.1 关于颜色视觉追踪和二维码

Link 的视觉系统包括颜色视觉追踪和二维码的辨识。摄像头通过 USB 与 Link 连接，根据光线条件，为 Link 提供不低于 6 帧/秒速率的图像。可以通过 Link 的屏幕界面，来为摄像头配置用于颜色视觉追踪或二维码辨识的颜色通道信息。

对于颜色视觉追踪，Link 处理摄像头传回的图像，并寻找与颜色通道里的指定颜色接近的“颜色块(blobs)”。颜色块是图像里能和指定颜色匹配上的连续像素点的集合。

对于每一个配置（可以设置多个颜色配置文件）里的颜色通道（每个配置有多个颜色通道），预期颜色值的取值是通过光谱图里的颜色选择框交互选择来确定的。摄像头的不断实时图像传送简化了预期颜色值的选取（如在触摸屏中选择了略微带红色的区域，这样程序就差不多能辨别红色物体）。为通道配置的光谱值在更改或删除前将保持不变。

摄像头获取的图片大小为 160 x120，左上角的坐标为（0,0），右下角坐标为（159,119）。Link 上显示的摄像头图片大小比实际图片大小略小。



Link 视觉系统的库函数用于选择一组配置以及获得用于辨识的颜色块的信息，如边界框的坐标，像素数量等。

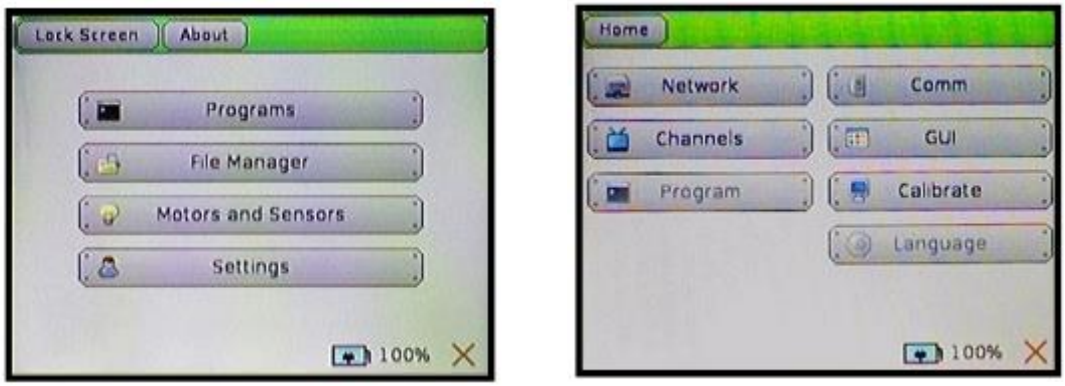
颜色通道除了用于颜色追踪，也可以通过配置来进行二维码的辨识。一个二

维码，本质上来说是一个用于简洁表示数据的二维条码。Link 库函数提供了对图像中的二维码的解码。

4.2 设置 Link 颜色追踪通道

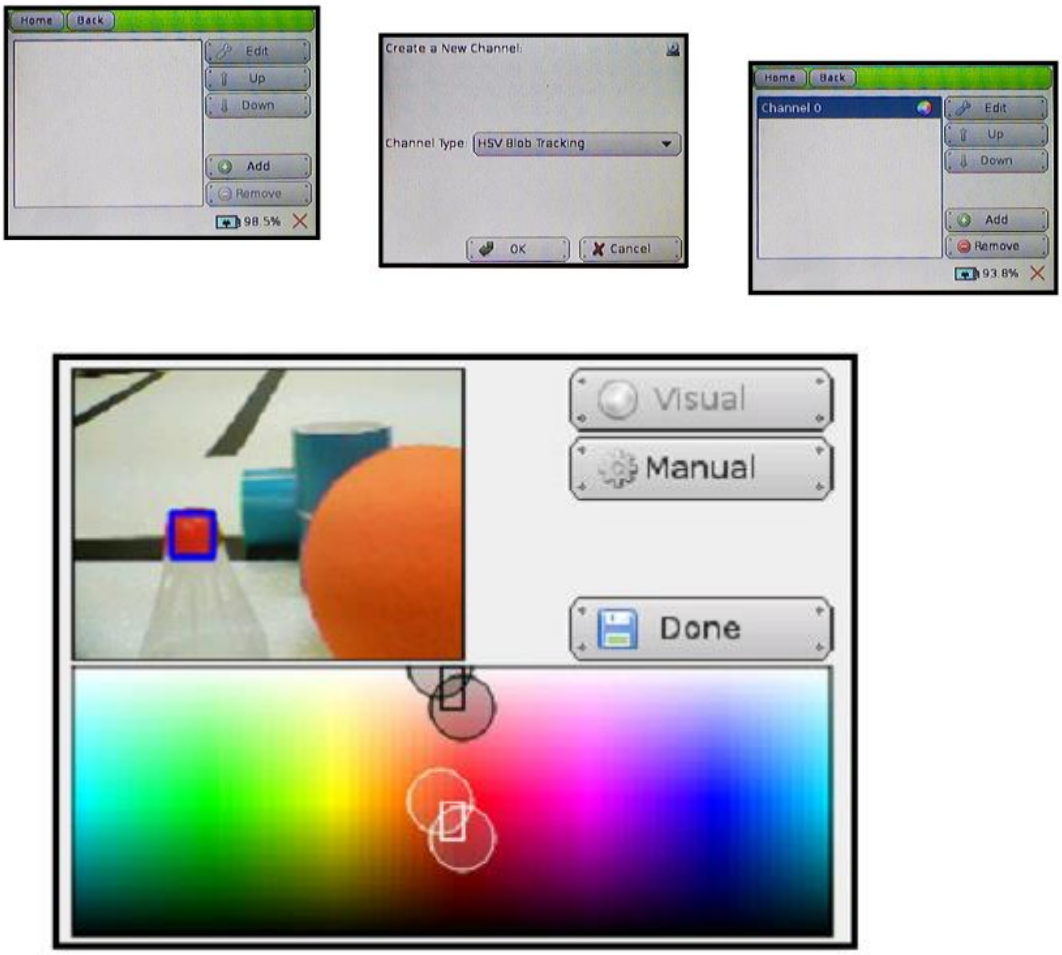
摄像头通过 USB 线插入 Link 底部的 USB（A 型，Link 上有 2 种 USB 端口，A 型是较大的）端口。注意：在读取摄像头图像的时候如果拔出摄像头，通常会使 Link 死机，这时需要重新启动 Link。

在 Link 主界面单击“*Settings（设置）*”进入 Link 设置界面。单击“*Channels（通道）*”进入通道设置界面，用来设置颜色追踪通道。

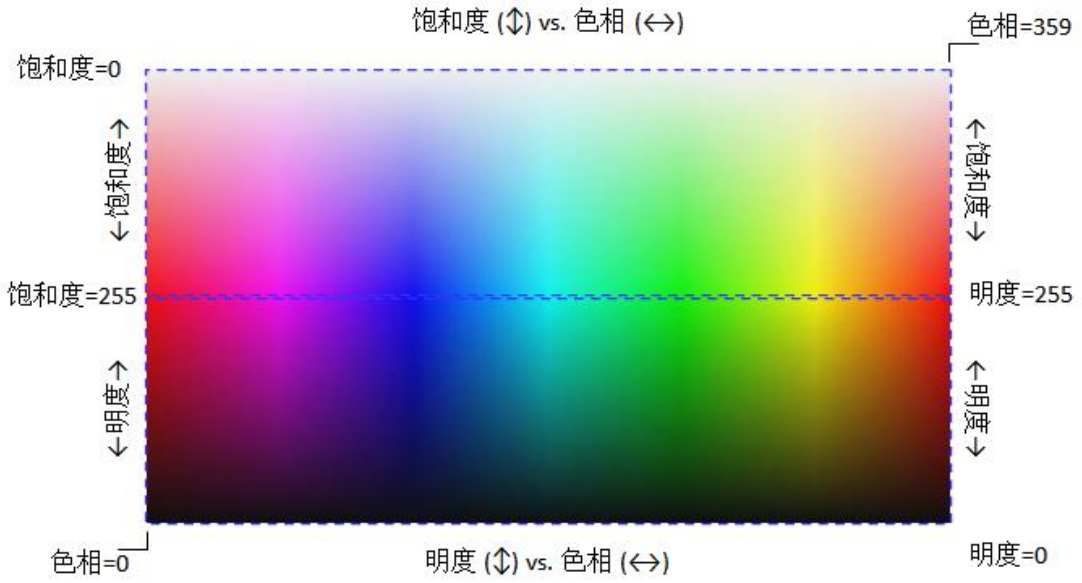


单击“*Add（添加）*”按钮来创建一个新的摄像头配置，输入一个配置的名称，然后界面将返回到通道设置界面。这时，列表里会出现刚命名的配置。如果用一个五角星标记了某个配置，那么这个配置就是当前的默认配置。选择你想要的配置，单击“*edit（编辑）*”.

定义好的通道会显示在列表里。单击添加，为该配置添加添加一个通道，然后选择“*HSV Blob Tracking（颜色追踪）*”，这样，该通道就用于颜色追踪了。选择该通道，单击“*edit（编辑）*”，进入 HSV 配置界面。通道号从 0 开始，1,2，……



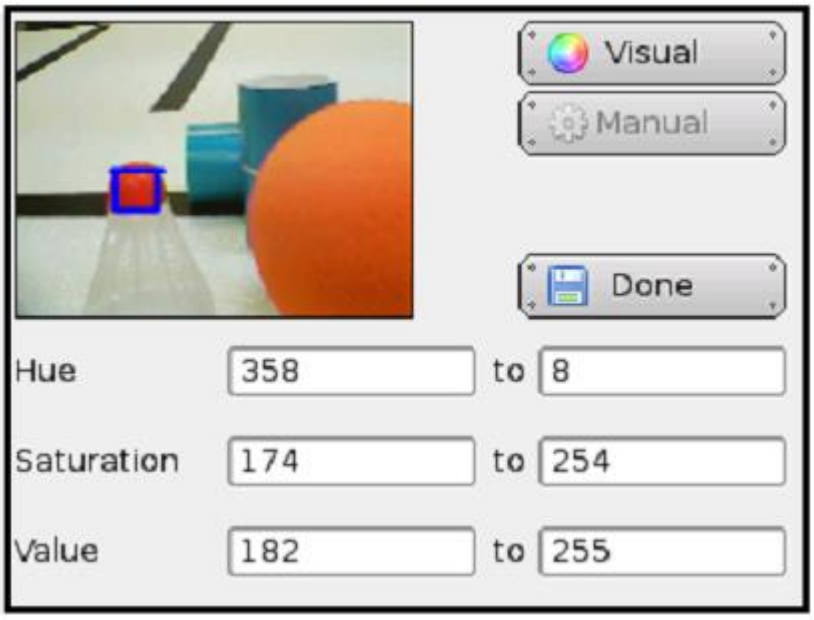
颜色的定义是根据 HSV (Hue-Saturation-Value，色相、饱和度和亮度)颜色模型进行的。HSV 颜色选择面板是通过两个堆叠面板来组织整个色相谱的:



可以通过拖动像素选择框里的圆形角来近似选择所期望的 HSV 值范围。举个

例子，如果通道准备用于查找红色颜色块，那么只需拖动选择框至光谱的红色部分，在图像中将在近红色物体边缘产生蓝色边框。当松开边框选择角，为了更好的显示颜色匹配, Link 将整个光谱图平行移动以使选择区域光谱位于屏幕中间(色相值为 0~359, 所以色相值的范围可能是从高到低，也可能是从低到高)。选择区域的每一个像素值都是和颜色相匹配的。在上图中，就有识别出来的近红色目标物体边缘有一个蓝色边框。

在获得一个颜色近似的 HSV 范围值后，可以选择 *“Manual（手动）”* 来设置更加准确的 HSV 值。单击手动，光谱面板将出现当前通道 HSV 值的数字框，这时就可以通过数字按键手动输入 HSV 值了。



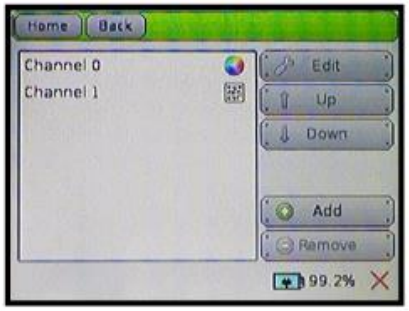
对于那些颜色相近但又能有效区分的颜色，如上面图片中的橘黄色和红色，手动输入 HSV 值是非常有用的。

单击“*Done（完成）*”按钮，界面返回到显示通道列表的配置界面，就可以继续添加颜色通道了。

4.3 设置 Link 二维码扫描通道

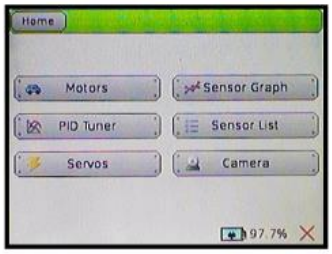
当为配置添加一个通道时，它的类型只有两种： HSV 颜色块追踪（HSV Blob Tracking or）或二维码扫描（ QR Code Scanner）。对于二维码扫描，就不再需要进行其他设置了。在通道列表上，二维码和颜色追踪是两种不同的图标来表示的。

后面将给出例程讲解如何进行二维码解码的。

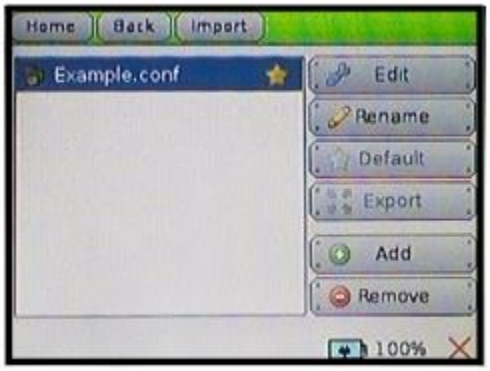


4.4 验证通道是否设置正确

进入 Link 主界面，单击“*Motors and Sensors（驱动器与传感器）*”，有一个选项是*“Camera（摄像头）*”，单击*此选项*，将出现默认配置下的各个通道的边界框，且每个通道的边界框的颜色各不一样（二维码边界框为黑色，通道 0 边界框为蓝色，等等）。



默认使用的配置将被标记以一个五角星。可以选择某个配置，单击“*Default（默认）*”即可将该配置设为默认情况下使用。



4.5 Link 视觉系统的库函数（含常用部分，其它详见附录）

- **camera_open(<resolution（分辨率）>)**

使用当前配置，摄像头开始工作。对于大部分用户，Link 使用的是默认配置。对于想使用其他配置的，应先加载配置后才能使用（见 camera_load_config 函数）。分辨率（resolution）可选择低分辨率（LOW_RES）,中分辨率（MED_RES）,高分辨率（HIGH_RES）。一般应用下，低分辨率能满足使用要求，且对系统动态性能的影响也最小。

- **camera_close()**

关闭当前摄像头，使其不再占用系统资源。

- **camera_update()**

通过 camera_open 打开摄像头后，每次调用 camera_updata 将读取和处理最新的摄像头图像信息。一般在使用其它摄像头函数之前调用该函数，以确保其它摄像头函数处理的是最新的图像。

- **get_object_count(<channel（通道号）>)**

返回给定颜色通道上识别出的物体的数目。识别出的最大的物体编号为 0，以此类推。

- **get_object_bbox(<channel（通道号）>,<number（编号）>)**

返回给定颜色通道上指定编号的目标物体边界框：以一个复合数据类型——矩形数据类型表示。该矩形数据有 4 个元素：

ulx——目标边界框的左上角的 x 方向坐标值

uly——目标边界框左上角的 y 方向坐标值

width——目标边界框的宽度

height——目标边界框的高度

- **get_object_center(<channel（通道号）>,<number（编号）>)**

返回给定颜色通道上指定编号的目标物体的中心坐标 ：

x——目标物中心点的 x 方向坐标值

y——目标物中心点的 y 方向坐标值

下面的函数用于二维码解码：

- **get_object_data(<channel（通道号）>,<number（编号）>)**

函数返回指针，该指针指向二维码所表示的有序数据。如果指定的通道号不存在，或没有发现目标，或没有数据，则函数返回 0。同时，返回的数据并不能保证是以 null 结尾的，但可以用数组的方法来读取，如：

get_object_data(0,0)[0], get_object_data(0,0)[1]…

函数返回的指针直到下一次 camera_update 调用时才失效，这时 **get_object_data** 将返回一个新的指针。

- **get_object_data_length(<channel(通道号)>,<number（编号）>)**

函数返回通道辨识的 QR 二维码所表示的字符数据的长度。如果通道不存在，或没有发现目标，或没有数据，则函数返回 0.

4.6 程序举例：控制舵机实现颜色追踪

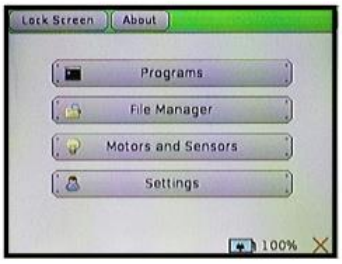
本示例程序是使用摄像头来控制舵机的一个演示程序。如果你手头没有舵机，请跳转到第二个示例程序中。在舵机上安装一个舵盘，程序的目的是通过摄像头 0 通道感知到目标物体在镜头前的左右运动，作为反应，舵机将不断调整角度使之瞄准目标物。

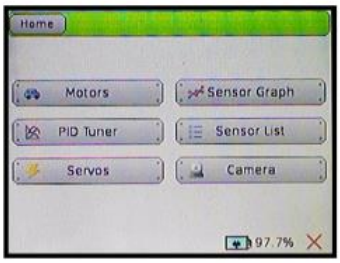
1. 设置

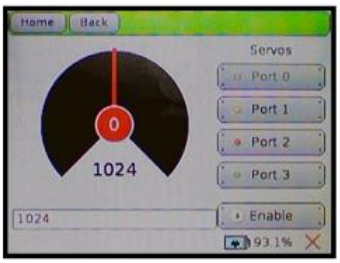
将摄像头连接到 Link，将舵机连接到舵机端口 0。同时将颜色通道 0 设置为颜色追踪模式，被追踪的物体颜色请尽量采用深色。另外，请进行调试，令舵机处于中间位置 1024 时，此时它应指向摄像头视域的中间位置。

从 Link 界面可以直接操控调试驱动器。这对于舵机来说，也是一种确定位置

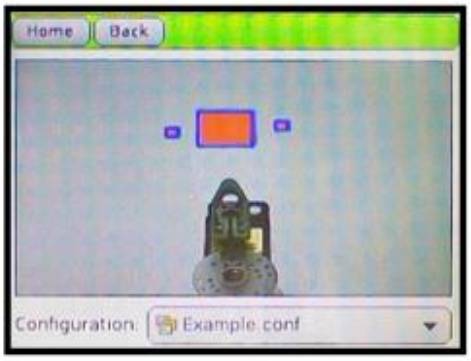
值的方法。在这个例子里，我们只需确定舵机的方向。单击“主界面→Motors and Sensors(驱动器与传感器)→Servos(舵机)”,进入舵机测试界面。选择端口 0，输入 1024，单击“*Enable（打开）*”，这样就能立即对舵机进行定位调试了。







单击“Back（*返回*）”，回到设备选择界面（如果希望关闭舵机在返回之前先按“*Disable（关闭）*”按钮）。选择摄像头，将显示摄像头图像。这样，你就可调整舵机位置来对准摄像头视域中心区域。为了获得更好的实验效果，可以在舵机输出轴上安装一个如下图所示的指示器 (颜色应与用于追踪的模型的颜色不一样)。

下面就可以运行程序了。记住：在下载文件到 Link 前先在 IDE 中进行编译确认程序无误，然后就可以下载到 Link 并且编译和运行程序了。

```
/* 将舵机插入 0 号端口，并将其中点与摄像头视野的中点对应，这个程序是用来保持摄像头指向 0 号通道中最大的一个目标。*/
int main() {
    int offset, x, y;
    enable_servo(0);      // 打开舵机
    camera_open(LOW_RES); // 激活摄像头
    camera_update();      // 读取摄像头所捕捉的图像并进行处理
    while(side_button() == 0) {
      x = get_object_center(0,0).x;  // 返回目标在图像中的横坐标
      y = get_object_center(0,0).y;  // 返回目标在图像中的纵坐标
      if(get_object_count(0) > 0) {  // 如果当前图像中能够捕捉到目标
        display_printf(0,1,"Center of largest blod: (%d,%d)   ",x,y);
        offset=5*(x-80); // 根据坐标计算舵机的偏移角度
        set_servo_position(0,1024+offset);
      }
      else
        display_printf(0,1,"No object in sight              ");
      msleep(200);       // 防止屏幕刷新过快
      camera_update();   // 在进入下一次循环前读取摄像头所捕捉的图像并进行处理
    }
    disable_servos();
    camera_close();
    printf("All done\n");
}
```

2. 代码

为了提高程序的可读性，同时因为 printf 函数在达到显示屏幕最后一行后进行滚动输出，Link 中使用了 display_printf 函数代替 printf。

- **display_printf(<col>, <row>, <string>, . . .)**

在屏幕的指定行列处按 printf 函数进行输出显示。列数（col）从 0~41，行数（row）从 0~9(如果其他软按键显示情况下，行数达不到 9)。使用“\n”来为显示换行。每行的字符数如果超过边界，将被删去不显。

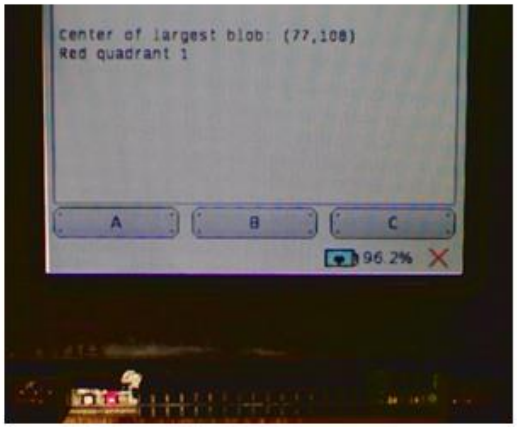
4.7 程序举例：控制电机指示灯亮灭来表示颜色追踪

本程序追踪 0 通道的目标物体,并根据物体的位置来点亮电机端口的 LED 灯,以此来表示目标位置。每一个电机端口的 LED 灯表示摄像头图像的四分之一大小。即如果目标物在图像的下半部，点亮红灯；如果目标物在上半部，点亮绿灯。

1. 设置

将摄像头与 Link 安装好，将颜色模型的通道 0 设置为颜色追踪，准备好目标物（颜色越深越好）。为演示效果更好，可以制作一张网格纸，并将摄像头视域的中心水平地对准网格。最后，将编译过的程序下载到 Link 中。

绿色 0	绿色 1	绿色 2	绿色 3
红色 0	红色 1	红色 2	红色 3



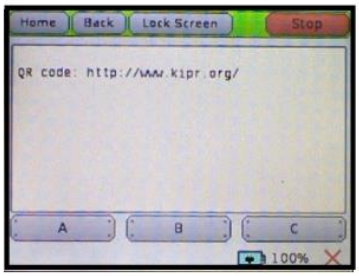
2. 代码

```
/*在这个程序中，我们用 4 个电机的指示灯代表 4 列格子，分别用绿色和红色代表上下两行，将目标物放到对应的格子里，会亮起相应的指示灯，例如将目标物放到第二行第二个格子里，则第二个指示灯亮红色，如果将目标物放到第一行第四个格子里，则第四个灯亮绿色*/
int main()
  {
    int x, y, quad, xMax = 160, yMax = 120;
    camera_open(LOW_RES);   // 激活摄像头
    camera_update();        // 读取摄像头所捕捉的图像并进行处理
    while (side_button()==0) {
      x = get_object_center(0,0).x;  // 返回目标在图像中的横坐标
      y = get_object_center(0,0).y;  // 返回目标在图像中的总坐标
      if (get_object_count(0) > 0) { // 如果当前图像中能够捕捉到目标
        display_printf(0,1,"Center of largest blob: (%d,%d)   ",x,y);
        ao();  // 关闭所有马达的指示灯
        // 确定目标所在的行
        if (x < xMax/4) {
          quad = 0;}
        else if (x >= xMax/4 && x < xMax/2) {
          quad = 1;}
        else if (x >= xMax/2 && x < 3*xMax/4) {
          quad = 2;}
        else if (x >= 3*xMax/4) {
          quad = 3;}
        if (y < yMax/2) { // 第一行（绿色）
          fd(quad);       // 将灯点亮为绿色
          display_printf(0,2,"Green quadrant %d    ", quad);
        }
        else {            // 第二行（红色）
          bk(quad);       // 将灯点亮为红色
          display_printf(0,2,"Red quadrant %d  ", quad);
        }
      }
      else {
        display_printf(0,1,"No object in sight              ");
        ao();             // 关闭所有马达的指示灯
      }
      msleep(200);        // 防止屏幕刷新过快
      camera_update();    // 在进入下一次循环前读取摄像头所捕捉的图像并进行处理
    }
    ao();                 // 关闭所有马达指示灯
    camera_close();
}
```

程序举例：解码二维码

本程序将演示通过将通道 1 设置为二维码模式，从而使用摄像头对图像中的二维码进行扫描并解码。

单击“主界面→Motors and Sensors(驱动器与传感器)→Camera(摄像头)”，进入摄像头图像界面，将摄像头对准二维码图标，确保通道 1 能识别出二维码。



编译并运行下面程序，破解二维码内容。

代码：

```
// 定义一号通道为二维码通道
// 二维码的内容会在其被锁定后自动翻译
int main() {
    int i, lngth;
    camera_open(LOW_RES); // 激活摄像头
    camera_update();      //  读取摄像头所捕捉的图像并进行处理
    while(side_button()==0) {
      if (get_object_count(1) > 0) { // 如果在视野范围内有二维码
        display_printf(0,1,"QR code:                              ");
        lngth = get_object_data_length(1,0);
        // 将所读取的二维码的内容以字符形式输出
        for(i=0; i < lngth; i++) {
          display_printf(9+i,1,"%c", get_object_data(1,0)[i]);
        }
      }
      else {
        display_printf(0,1,"No QR code detected                  ");
      }
      msleep(200);        // 防止屏幕刷新过快
      camera_update();    // 在进入下一次循环前读取摄像头所捕捉的图像并进行处理
    }
}
```

5. 故障排除

对于任何疑难点或难以解决的问题，都可以在上午 9 点至下午 5 点来电咨询 Botball 中国组委会技术支持 010- 66537227 或者发邮件至 support@itccc.org.cn。

问题	解决
Link 无法开机。	将 Link 连接上充电器，进行充电。此时红色 LED 亮，表明充电器正确插入到位；绿色充电状态指示 LED 灯也亮，表明 Link 正在充电。当最多 90 分钟后，Link 充满电，绿色 LED 灯由亮转灭。充电时，Link 应处于开机状态。
开机时，所有电机端口的红色 LED 灯亮。	表明 Link 的固件已不可用，应立即更新固件。详见附录中关于更新固件的说明。
Link 开机了，但是只加载到启动画面，且不断地自动重启。	表明 Link 的固件已不可用，应立即更新固件。详见附录中关于更新固件的说明。
Link 触摸屏不响应或不灵敏。	应重新校准触摸屏。单击“主界面→*Settings(设置)*→*Calibrate(校准)*”。依次触碰四次“十”型点（最好用触针）即完成校准。如果主界面按钮仍不灵敏，请重新校准，在触碰“十”的时候尽量靠近中心点。
Link 对于 IDE 中编译通过的程序不能成功编译。	表明你的 Link 固件与 IDE 版本不对应。请更新固件，并安装最新版的 IDE。详见附录。
摄像头传回的图像迟滞，变黑，模	表明 Link 与摄像头失去连接(如拔掉摄

糊。	像头后又重新插入）。应重启 Link。
Link 无法被苹果机 10.4 操作系统识别，但可以被 10.5 或更高版本操作系统或 Windows 操作系统的电脑识别。	表明你的 Link 固件与 IDE 版本不对应。请更新固件，并安装最新版的 IDE。详见附录。
Link 无法被苹果机 10.5（或更高）操作系统识别。	启动 Link，检查并确保 Link 在苹果系统列表里。如果不在，检查并确保 USB 连接线正确插入并被识别。如果问题还没解决，换一个 USB 端口试试。
Link 无法被使用 Windows XP 操作系统的 PC 识别。	启动 Link。在 Windows“控制面→设备管理器”下查看 Link 是否在 COM 端口列表里。它可能显示为“*Gadget Serial v2.4*”。如果系统安装设备驱动不成功，将产生设备告警，可单击端口，或从 IDE 程序包里重新安装驱动。如果设备没有出现在 COM 端口列表里，检查并确保 USB 线连接正确到位，因为电脑通过 USB 线给 Link 充电可使电脑连接上 Link。如果还不行，换一个 USB 端口试试。

Link 无法被安装了 64 位 Windows7 操作系统的 PC 识别。	启动 Link，进入操作系统的“设备管理器”查看 COM 端口列表里是否有“*KIPR Link*”或“*Gadget Serial*”。如果有但端口号大于 12，应清理 COM 端口，以确保 IDE 有足够时间去发现 Link 的 COM 端口，因为端口号是从小到大去识别的。如果列表里没显示 Link COM 端口，在“*其它设备*”里寻找，如果还是没有，试着从 IDE 安装包里重装设备驱动。如设备管理器中发现了“*Gadget Serial*”，右击并选择更新驱动程序。如果能接入因特网，“*自动搜索*”即能解决该问题。或者选择“浏览我的电脑”，选择 *C:Windows/KIPRLinkDriver* 作为更新文件的位置，从而进行更新。如若没有发现驱动文件，在控制面板里选择“文件夹选项→查看”，选择”显示隐藏文件，文件夹和驱动”。这样就可在 C*:Windows/KIPRLinkDriver 中*发现驱动文件夹了。最后返回设备管理器，选择“浏览”选项。 在 Windows 操作系统下，有时先前使用的串口设备或在系统关闭而操作系统未关闭的情况下连接了串口设备都可能使电脑出现额外的、隐藏的 COM 端口。设备管理器能设置显示隐藏的 COM 端口并移除所选择的端口： 步骤 1：打开“程序→附件”，右击

	命令提示符，选择以管理员身份运行 步骤 2：回车，设置 devmgr_show_nonpresent_devices=1 步骤 3：回车，运行 *devmgmt.msc*，打开设备管理器 步骤 4：在设备管理器菜单栏下，选择“查看→显示隐藏文档” 步骤 5：展开 COM 端口列表，右击每一个额外的 COM 端口，选择“卸载”，即可清除该端口。

我们已经在各个操作系统上实验过 KISS IDE 并确认能够正常安装、使用，但是可能存在某些系统设置不一样，导致 Link 不能正确被电脑识别。如果，采取以上方法也不能解决问题时，请致电 Botball 中国组委会技术支持以获取帮助。

6. 附录

6.1 更新 Link 固件

Link 的固件是为 Link 在启动时提供用户界面和其它系统服务的一种基于 Linux 操作系统的镜像文件。可从 http://files.kipr.org/Link 获得固件下载。从 http://www.kipr.org/kiss-platform-Link-firmware 则可获得安装指导，也可获得最新的官方版本下载。

所需设备： Link，Link 电源适配器，U 盘

步骤 1： 下载固件的最新版本。无论是什么版本号，下载的固件镜像文件都是一个叫做 kovan_update.img.gz 的压缩文件。该文件不需用户解压缩，一切都将在安装过程中由系统自动处理。

步骤 2： 将 kovan_update.img.gz.文件复制到 U 盘的根目录下。

步骤 3： 单击弹出 U 盘，并等待系统显示已弹出后拔掉 U 盘。从苹果机上弹出 U 盘需右击存储器图标并选择弹出。对于 Windows 机器，在我的电脑下右击存储器图标，选择弹出。这样做可以保护存储器不遭破坏，也能防止在复制未完成的情况下拔掉 U 盘而造成镜像文件损坏的情况。

步骤 4： 拔掉 Link 上的所有驱动器与传感器，**接上电源适配器**。关闭 Link，并将 U 盘插入 Link。

步骤 5： 按下边侧白色按钮不放，同时开启 Link。

步骤 6： 保持按下边侧白色按钮不放，直到出现更新进度条。这就表示更新已经开始启动。这时可以松开按钮。

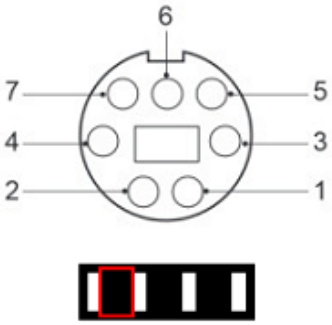
步骤 7： 更新完成后即可拔去 USB 存储器。重新启动 Link，并确认固件版本。如果还不是最新版本（有可能是 Link 使用了先前版本的固件进行的更新），重复这个过程。

步骤 8： 进入“主界面→Settings(设置)→Calibrate(校准)”，重新校准触摸屏

6.2 使用 Link 控制 iRobot Create

Link 通过 TTL 串口与 Create 连接并控制它。可以通过官方购买或自己制作连接线：Create 使用 7 针连接线，其第 3 针是 Create TTL 信号输入，4 针是信号输

出，6、7 针接地，1、2 针接电源正极。Link 使用三个母孔来对应连接线，即（红线）对应 Create 串口信号输出线，地，串口信号输入线。



可通过下面步骤来使 Link 和 Create 建立通信连接：

步骤 1：将连接线一端接入 Create（此端口默认是被一个盖子盖住的）

步骤 2：将连接线另一端接入 Link 串口端口（标配线只能在红色标记线对准端口的方向单向插入）。电源连接是可选项（如果有的话），在 Create 开机后能给 Link 缓慢充电。如果不使用标配的连接线，应注意线序的正确性（Create 串口信号输出，地，信号输入）才能使 Link 和 Create 正常连接。

步骤 3：编写一个两者连接的程序并下载到 Link。

步骤 4：将 Create 放置在地上。

步骤 5：将 Create 开机，**切勿忘记**。

步骤 6：执行 Link 上的程序。

Link 的串口一般用于通过 USB 线从 KISS IDE 下载程序到 Link，也用于和 Create 的通信。库函数 create_connect 的作用即是将 USB 连接转换到串口通信功能来。只有先调用此函数，后面的 Create 库函数才有效。反之，create_disconnect 将串口通信转换到 USB 连接。

6.3 Link 里控制 iRobot Create 的库函数

Link 里包含了大量用于控制 Create 的库函数。这些函数为 Link 里的程序与 Create Open Interface 之间提供了通信接口。(http://www.irobot.com/filelibrary/pdfs/hrd/Create/Create%20Open%20Interface_v2.pdf)。这些函数获取传感器数据，或返回需要的信息（如发生错误，则返回大于 100,000 的值）。如果发生了错误，返回错误信息为 100,000+<Create 串口包编

号>。如，读取碰撞传感器或轮降（指示 Create 的轮子是否悬空）传感器状态值的时候返回 100,007，则表示发生了错误。

驱动 Create 运动的函数是非阻塞(non-blocking)的，因此程序不会等待驱动命令的完成。运动命令（除了 create_stop）如果和先前的运动命令不一样，则立刻被发送到 Create 并执行。因此，运动命令可以放在循环里被不断调用而不用考虑对串口通信的影响。Create 的运动只有在下一个移动命令到来时才会得到改变。

Create 也可以用于播放 MIDI 音乐。通过全局变量数组 gc_song_array，Create 可以下载超过 16 首的音符歌曲，在 Create 的开放接口手册有更详细说明。

下面是常用函数，更多函数请参阅附录。

- **create_connect()**

在 Link 和 Create 间建立连接，需要先调用此函数。默认情况下连接后 Create 处于“安全模式”。

- **create_disconnect()**

将 Link 串口用于 USB 连接从而断开 Link 和 Create 的串口连接。应在程序的最后的调用该函数，否则 Link 将仍保持与 Create 的通信，导致 IDE 不能连接上 Link。该函数也会关闭 Create 的所有驱动器。重新打开电源也能使 Link 串口转换到 USB 上。

- **create_stop()**

关闭 Create 轮子的驱动电机。

- **create_drive(<speed（速度）>,<radius（半径）>)**

以指定速度转弯，速度范围是 20~500mm/s，半径单位是毫米。

- **create_drive_direct(<left motor speed（左电机速度）>,<right motor speed（右电机速度）>)**

令单独的左轮和右轮驱动器以速度 20~500 毫米/秒（可能有浮动）运动。

当 Create 运动时，其运动的距离和角度都会被累加记录下来，这些数据可以通过 Link 中的 set_create 和 get_create 函数来初始化和读取，更多说明见附录。两个常用函数：

- **set_create_distance(value 数值)**

将距离值初始化到指定值，单位为毫米。

- **get_create_distance()**

从上一次初始化距离值开始计算，返回之后 Create 行驶的总距离值。

- **set_create_total_angle(value 数值)**

将角度值初始化到指定值，单位为度。

- **get_create_angle()**

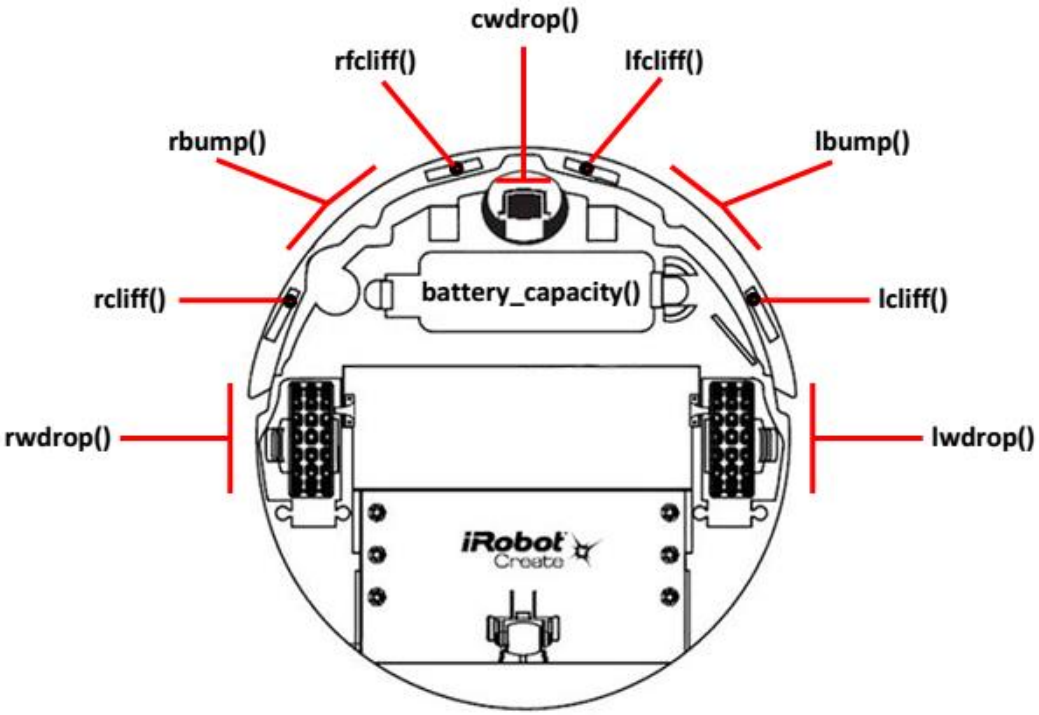
从最后一次角度值的初始化开始计算，返回 Create 转过的总角度值（如果）。

“get_create”系列函数也用于查询 Create 自带的传感器的状态值：

- **get_create_lbump()**

返回左侧碰撞传感器的值（被按下为 1，反之为 0）

下面的图中显示了所有“get_create”系列函数的后缀，它们可以用于查询相应的传感器值（如 get_create_rbump 或 get_create_lcliff）。



完整的 get_create 函数列表请参见 IDE 帮助文档或附录。

使用 Create 函数库时，用户应该明白 Link 和 Create 是相互独立的，它们之

间是通过串口来连接的。如果失去连接，Create 仍会按照最后收到的命令来自主动作。另外，除非你的程序考虑到了自动识别传感器数据丢失，两者的通信延时也可能导致传感器数据的丢失，（如你的程序可以在 0.1 秒内读取 3 次传感器值而只使用最适合的 2 个值）。

6.4 用 Link 控制 Create 的编程示例

程序演示：Create 以 200mm/s 的速度绕 0.25 米半径的圆行驶 10 秒，然后显示行驶的总距离和转过的总角度。

```
/* Create 以 200mm/s 的速度绕 0.25 米半径的圆行驶 10 秒，然后显示行驶的总距离和转过的总角度。*/
int main()
  {
    printf("connecting to Create\n");
    create_connect();
    set_create_distance(0);
    set_create_total_angle(0);
    create_drive(200, 250);
    msleep(10000);
    create_stop();
    printf("\nResults:\n");
    printf("  distance = %d mm\n", get_create_distance());
    printf("  angle = %d degrees\n", get_create_total_angle());
    printf("\ndisconnecting from Create\n");
    create_disconnect();
}
```

准备

将 Link 和 Create 充满电。将 Link 和 Create 用连接线接好。将 Link 放入 Create 的“小仓”里。将 Create 放置在空阔干净的地上。注意，如果 Create 探测到区域边界或车轮失去传感器（如被悬空），程序将停止，Create 将发出怪声。

代码

6.5 编写 Create 脚本（*高级内容，一般情况下不需要使用）

关于 Create Open Intreface(Create 开放接口)可以参阅

(http://www.irobot.com/filelibrary/pdfs/hrd/Create/Create%20Open%20Interface_v2.pdf)上的指导，该手册也包含用于控制 Create 的串口字节编码的说明。

Link 通过串口线与 Create 连接通信。Link 提供了进行通信的函数库，通过调用相应函数 Link 向 Create 发送串口字节码序列以执行相应的功能。这种方式简化了用户的工作，就不需要用户自己去查阅 Open Interface 手册并编写串口字节码从而操控 Create 了。这些函数包括了 Create 组件大部分的常用动作（如以指定速度前进，获得 Create 行驶总距离等）。

但是 Link 也有直接操控 Create 的方法，即使用开放接口提供的内置脚本功能。所谓的脚本就是对应 Create 动作的一系列字符码命令，且它是独立于外部控制的。和其它命令不同的是，开启脚本的命令将中断其它串口通信直到脚本执行完毕。

Create 有几个内置的脚本，主要用于控制 Create 的兄弟机器人 Roomba（吸尘器机器人）。开放接口提供字节码命令用于运行这些脚本。它也提供一些字节码命令，用于将用户自己编写的脚本下载到 Create 里，以及启动这些脚本的字节码命令。这些脚本将一直有效，直到 Create 电力不足。

与高级语言不同，脚本没有诸如 if,while 之类的控制流命令，但是还是有它自己形式的命令，如：等待一定时间，行驶指定距离或角度，或等待诸如碰撞之类的事情发生（*等待*命令只能在脚本里才能使用）。

用于存储用户自定义的脚本的存储器容量不超过 100 字节。

create_write_byte 函数用于脚本的定义并发送脚本至 Create（以字节为单位发送）。Link 函数库提供 3 个命令用于两者之间的串口通信。

- **create_read_block(<data string 数据流>, <count>)**

Create 返回指定字节数的数据流。

- **create_write_byte(<byte 字节>)**

Link 将指定的字节发送到 Create。

- **create_clear_serial_buffer()**

清空内部的串口缓冲寄存器中任何尚未读取或发送的数据。

举例

下面的程序将包含这样一个函数：使用 create_write_byte 写一段脚本并一个字节一个字节地传送到 Create。该脚本将使 Create 以指定速度行驶指定距离，以此来说明定义一个脚本所涉及到的低级编程方式。

代码

```
#define RUN_SCRIPT create_write_byte(153)
void make_drive_script(int speed, int dist) {
  create_write_byte(152); // specifies start of script definition
  create_write_byte(13);  // remaining number of bytes in script
  create_write_byte(137); // drive command, speed & radius follow
  create_write_byte(speed >> 8); // send bits 8-15 of speed
  create_write_byte(speed); // and then send bits 0-7
  create_write_byte(128); // send hex 80 (X8000 specs drive straight)
  create_write_byte(0);   //   and then send hex 00 (no turn radius)
  create_write_byte(156); // wait for distance command, dist follows
  create_write_byte(dist >> 8); // send the 2 bytes
  create_write_byte(dist);      //   for distance in mm
  create_write_byte(137); // stop by dropping speed and radius to 0
  create_write_byte(0);   // speed = 0
  create_write_byte(0);
  create_write_byte(0);   // turn radius = 0
  create_write_byte(0);
  // end of script (15 bytes)
}
int main() {
  create_connect();
  set_create_distance(0);
  set_create_total_angle(0);
  make_drive_script(500, 500); // script to move 0.5m at 500 mm/sec
  msleep(500); // give serial connection some time
  RUN_SCRIPT;
  msleep(1500); // allow time for the script to finish (+ some extra)
  printf("  distance traveled = %d mm\n", get_create_distance());
  printf("  angle turned = %d degrees\n", get_create_total_angle());
  create_disconnect();
}
```

当一个 16 位的 int 整型数据（范围是-32768~+32767）作为 Create_write_byte 的参数时，实际只有低八位被使用（0~7 位）。为传送高八位，先将该值右移（>>）

八位即可。

该脚本有三个主要的字节码命令：

1. 字节码 152

表示脚本定义开始，且必须下接脚本的字节总数（0~98）。

2. 字节码 137

字节码 137 是一个驱动命令，且必须立即接着发送 4 个字节来分别表示 2 个 16 位数据，分别用于指定速度和转弯半径（单位毫米）。数据的高八位先发送。当发送正的速度和正的半径值，则 Create 向左前行驶，则 Create 将向左转弯。转弯数据为十六进制的 8000 或 7FFF，是特殊数据表示 Create 直线行驶。也有特殊数据是用于 Create 顺时针或逆时针转动的。

3. 字节码 **156**

命令让 Create 保持等待直到行驶了指定的距离，后接 2 个字节来表示一个 16 位的指定距离，单位毫米。

- 字节码 **153**

用于令 Create 开始执行定义好的脚本。

- 字节码 **145**

用于单个驱动器的驱动命令(对比字节码 137 是对机器人整体的控制而不是针对单独的驱动器的)。其后接两个 16 位数据用以指定左右轮速度。

上例中的等待命令是用于行驶距离的，共有四个等待命令：

1.等待时间 **155**

该字节码后接一个字节的数据，表示 0~255 个十分之一秒，分辨率为 15ms。

2.等待行驶距离 **156**

该字节码后接二个字节的数据，来表示一个 16 位的行驶距离。

3.等待转过角度 **157**

该字节码后接二个字节的数据，来表示一个 16 位的转过角度值。

4.等待事件发生 **158**

该字节码后接一个表示事件号的字节（+或-，+表示事件发生，-表示事件不

发生）。举例：事件-9 表示“未探测到墙”（即当一个 Create 被墙上传感器探测到后又移出探测范围，事件才发生）。更多的 22 个等待命令可参阅 Create 开放接口的手册。

脚本的典型应用是移动 Create，上面程序可当做一个模板用于编写脚本。

脚本使用的其他事项：

1. 在脚本的最后用字节码 **153**（脚本运行命令）结尾，可使脚本持续运行。

2. 如果所等待的事件发生，则以“驱动 Create 运动，等待事件，结束”为架构的命令序列将立即停止，这样驱动 Create 运动的脚本就可能不会有机会运行。

3. 无论提供脚本的 Link 是否还处于与 Create 的连接状态，脚本都将继续控制 Create。举例：一个用户的脚本开始后立马等待“触发”事件发生（如碰撞），Create 将一动不动，直到事件发生。在等待期间，可以断开 Link。因此，脚本是独立操控 Create 的一种方法。

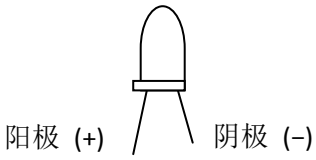
6.6 示例程序：使用 Link 数字输出来点亮 LED 灯

默认情况下，数字端口是配置为输入的。库函数 **set_digital_output** 用于配置数字端口的方向。举例：

```
set_digital_output(9,1);设置端口 9 为输出
set_digital_output(9,0);设置端口 9 为输入。
```

对于一个设置成输出的数字端口，Link 使用 set_digital_value 函数来设置输出值：

```
set_digital_value(9,1);端口 9 输出高电平。
```



如果你现有 5mm 的 LED 灯，你可以使用下面程序来操控它。当它的阳极比阴极电压高 1.9~3.2V（取决于灯的颜色），LED 灯将发光。如果电流过大，LED 将被击穿（典型值为 20~30mA）。对于 Link 的在 SEN 脚上的数字输出，是足够点亮 LED 而不把它烧坏。

LED 灯的较长引脚为阳极。另外，阴极侧通常有较厚的平的基底。

准备

将 LED 的阳极引脚接入 9 号端口的 SEN 孔,阴极接入 GND。

代码

```
/*测试这个程序需要将 LED 插入 9 号数字端口*/
int main()
  {
    printf("LED in port 9\n");
    printf("Press side button to quit\n");
    set_digital_output(9, 1);
     while (side_button() == 0) {
      set_digital_value(9, 1);
      msleep(500);
      set_digital_value(9, 0);
      msleep(500);
    }
    set_digital_output(9, 0);
    printf("\ndone\n");
}
```

6.7 示例程序：使用线程来监控传感器（*高级内容，一般情况下不需要使用）

在计算机科学中，“执行线程”简称线程。它表示的是由系统管理的一系列指令，操作系统一边执行进程一边安排进程时间。对于一个单处理器计算机来说，如 Link，各个线程的指令看上去是同时并行处理的。每一个线程一旦开始执行，直到程序结束或使用 thread_destroy 函数，才能迫使该线程结束。每一个线程轮流由操作系统分配处理时间（时间片），直到它的所有语句被执行完成或被强制结束。如果一个线程的程序还没执行完，则当处理器处理其它线程时，此线程会暂时停止（挂起）。所有的线程轮流得到它们的时间片，直到程序执行完毕。如果处理器运算速度足够快，从用户角度看上去就好像所有的程序在同时并行运行一样。

线程里的函数能够通过读取和修改全局变量来和其它函数通信。全局变量可

以作为程序间互通信息的标记。进程号可以被存储在全局变量中，如果程序逻辑确实需要，一个进程能够终止另外一个进程的线程（如一个无限循环监视传感器的进程，如果不这样，将一直不会结束）。

Link 中有四个控制线程的库函数，**thread_create, thread_destroy, thread_start,** 和 **thread_wait**。存储线程系统号的变量数据类型必须为 thread 类型。

下面的示例程序中将创建一个线程，该线程执行监视边侧按钮的函数，如果被按下，则将一个标志变量赋值为 1。因此，当主函数处于休眠模式下时，程序仍能捕捉到按钮被按下这一事件的发生。标志变量通过一个全局变量来实现。

代码

```
int flag = 0; // 用来记录按键状态的全局标记变量
void chksens() {
  while (1) {
    if (side_button()) flag = 1;  // 如果按下，标记为 1
    msleep(100); // 将次标记状态持续 100 毫米
  }
}
int main() {
  int cnt = 0;
  thread tid; // 创建一个名为 tid 数据类型为 thread 的变量存储线程 ID
  tid = thread_create(chksens); // 为 chksens 创建线程
  thread_start(tid);  // 运行线程
  while (flag == 0) { // 当主函数休眠（sleep）时，线程仍然运行
    display_printf(1,1,"elapsed time %d   ",++cnt);
    msleep(1000);
  }
  thread_destroy(tid); // 关闭线程
  printf("\ndone\n");
}
```

6.8 自制传感器（*高级内容，一般情况下不被 Botball 搭建规则允许）

所需工具

电烙铁、斜口钳、剥线钳、热熔胶枪、热缩管、美工刀、剪刀

所需材料

1 x 4 排针，2.54mm 间隔（一般是将成排的接头按需要硬剪）。

杜邦线、焊锡、传感器（3.3V 最好，5V 也能使用）、绝缘材料（热熔胶、热缩管）。

方法

Link 的传感器接口是标准的 2.54mm 母头插孔，数字和模拟传感器插孔均是三排。一二排之间的间隔为 2.54mm。一排是传感器输入 SEN，二排是 5V 的 VCC，三排接地。可以通过打开 Link 调整跳线帽来更改电压值。

3.3V 或 5V 的传感器均能连接到 Link 正常工作。对于一个 3.3V 的传感器，当电压达到 3.3V 或更高(第一排)，Link 通过 SEN 读进来的读数达到最大值。用户需要查阅传感器的使用手册来为你的传感器接线。

Link 自带的传感器使用的是标准的 28AWG 杜邦线缆，它比一般的线缆更加柔软。传感器连接线将根据传感器自身的不同而使用 2 线或 3 线。Link 传感器的连接头是 1x4 的排针，在排一和排二之间拔掉了一根插头（可防插反）。

传感器与线缆的连接方法：首先，在线缆的两端将每根线分开一定的距离，剥去足够长度的绝缘皮。一端和传感器端脚焊接在一起，一端和排针焊接在一起。注意两端的线序要一致，可以通过线的颜色来辨识，避免交叉接线。

线和传感器、排针接好后，为防止接线处可能的接触，需要将各个端脚隔离开来。原装的传感器使用热熔胶来隔离端脚，也起到了强化连接的作用（将热熔胶液包裹住连接点，尽量使表面平坦。变硬后再用小刀或剪刀修剪掉多余的胶）。

在传感器插入 Link 之前，还需要用万用表检查一下是否短路或是否有虚焊。

进入 Link 传感器界面（*“Motors and Sensors（驱动器和传感器）”→“Sensor List（传感器列表）”*），插入传感器并试着改变传感器的读数。如果为数字传感器，则返回值为 0 或 1；模拟传感器读数范围为 0~1023。

6.1 自制电机（*高级内容，一般情况下不被 Botball 搭建规则允许）

所需工具

电烙铁、斜口钳、剥线钳、热熔胶枪、热缩管、美工刀、剪刀

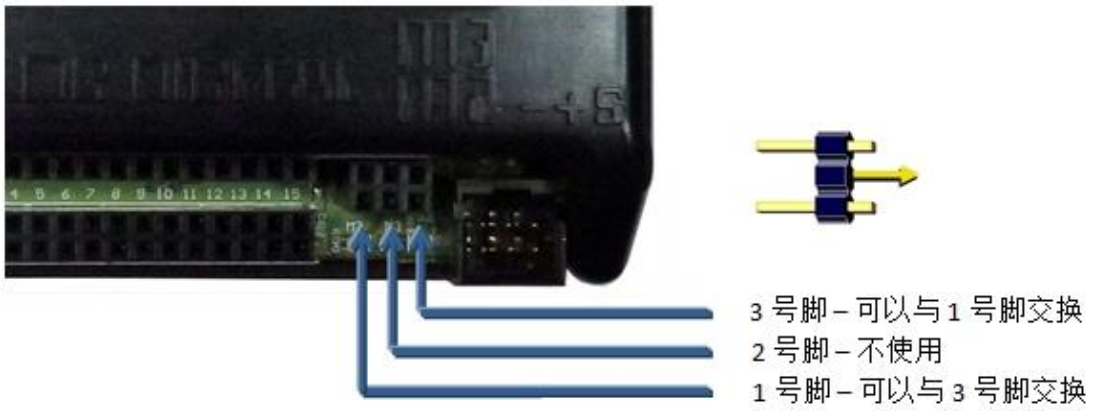
所需材料

1 x 3 排针，2.54mm 间隔（一般是将成排的排针按需要剪断所得）

杜邦线、焊锡、电机（5V 或更高，最大电流 1A）、绝缘材料（热熔胶、热缩管）

方法

和传感器端口一样，Link 直流电机接口也是使用间隔 2.53mm 的母头插孔。每一对电机端口由两排 1 x 3 母头插孔组成，内部是产生 PWM 波的两个 H 桥芯片。两侧的两脚（1 和 3 脚）如同电池一样为直流电机提供驱动电压。若 1 脚为负 2 脚为正，则电机插上后绿色 LED 灯点亮，反之红色 LED 灯点亮。我们约定，绿灯亮表示“向前”，红灯亮表示“后退”（如果不符合约定，将电机反接即可）。中间的 2 脚在内部是和 1 脚连接的，一般情况不要使用。



当选择一个电机时，注意电机的电压是由 PWM 波来提供的，但是它提供的有效的 0~5V 电压和电机的速度是非线性的。使用反电动势 PID（比例微分积分）方法能提供更精确的电机运动控制，但是需要合理设置 PID 值来匹配电机特性，而这需要大量实验才能确定。还需注意的是，电机端口最大输出电流为 1A，当超过该值容易烧坏 Link。

当使用特殊的电机时，还需要查阅电机手册才能确定电机的接线方式以及该电机是否能用 Link 进行控制和电机对 PWM 波的反应。

Link 自带的电机使用的是标准的 28AWG 杜邦线缆，它比一般线缆更加柔软。

你的电机连接线将使用 2 线，如同将电机直接连接到电池上一样。

电机插头是 1 x 3 的排针制作，中间的针脚被拔出。

电机与线缆的连接方法：首先，在线缆的两端将每根线分开一定的距离，剥去足够长度的绝缘皮。一端和电机端脚焊接在一起，一端和排针焊接在一起。注意两端的线序要一致，可以通过线的颜色来辨识，避免交叉接线。

线和电机、排针接好后，为防止接线处可能的接触，需要将各个端脚隔离开来。原装的传感器使用热熔胶来隔离端脚，也起到了强化连接的作用（将热熔胶液包裹住连接点，尽量使表面平坦。变硬后再用小刀或剪刀修剪掉多余的胶）。

在电机插入 Link 之前，还得用万用表检查一下是否短路或是否有虚焊。

进入 Link 驱动器界面（*“Motors and Sensors（驱动器和传感器）”→“Sensor List（传感器列表）”*），插入电机并在 Link 里开启该端口。单击“*Forward or Backward（前进和后退）*”，则 Link 以最大电压驱动电机。

6.9 传感器端口 5V 或 3.3V 电压的设置（*高级内容，一般情况下不需要使用）

注意！本项修改要求打开 Link 外壳，而这会使得保修失效。同时，KIPR 对任何可能的后果都不负责任。

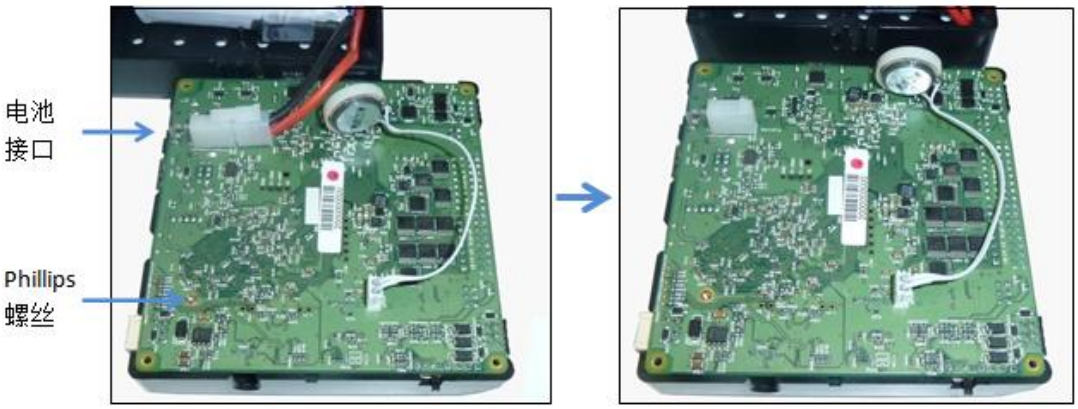
默认情况下，数字和模拟传感器端口的 Vcc 引脚的电压是 5V。而这也是信号引脚 SEN 的标准电压。可通过内部跳线帽来选择电压是 5V 还是 3.3V。编号为 0-7 的模拟端口和 8-15 的模拟端口都分别有一个对应的跳线帽。Botball 套件中的传感器在两种电压下都将正常工作，对于自行购买的传感器则需要查阅使用手册。

在打开 Link 外壳前，请确保工作场地是静电隔离的，并明白这将使得保修失

效。同时最好有有个接地线。另外，不要接上充电器。

步骤 1：用螺丝刀将底部的四个螺丝卸去。

步骤 2：移去底盖，拔掉电源连接线，卸下用于固定主板与面盖的 Phillips 螺丝。



步骤 3：慢慢地将主板取出，将面板拿走，这时就可以看到如下所示的跳线接头了。注意不要扭结或拉伸显示屏的排线。然后就可以取下跳线帽再按需要插上（电路板上有电压值的提示）。下图中，左边的模拟端口被设置为 3.3V，右边的数字端口被设置为 5V。

步骤 4：取回面板，放置好主板，安上 Phillips 螺丝。

步骤 5：重新接上电源线，将 Link 关机。必须确保电源线是整洁的且扬声器安放到位。最后再把四个螺丝将后盖合上。

步骤 6：开启 Link，检查是否正常工作。

6.10 Link 的主要库函数（按照字母顺序罗列，且自动包括来自 C 数学库的数学函数）

1. a_button [分类: 传感器]

格式: int a_button();

读取 a 按钮（Link 软按键）的值（0 或 1）。

2. a_button_clicked [分类: 传感器]

格式: int a_button_clicked();

获取 a 按钮的状态（按下或未按下）。如果按下按键则阻塞程序直到按键被释放。按下，返回 1；不按下则返回 0；按下不释放则程序暂停。语句

```
while (a_button()==0) {
while (a_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (a_button_clicked()==0) {. . .}
```

3. accel_x [分类: 传感器]

格式: int accel_x();

返回 Link 水平 X 方向的加速度值。

4. accel_y [分类: 传感器]

格式: int accel_y();

返回 Link 水平 Y 方向的加速度值。

5. accel_z [分类: 传感器]

格式: int accel_z();

返回 Link 铅垂面 Z 方向的加速度值。当 Link 水平放置时该值和重力加速度常量正相关（重力加速度使人呆在地面而不至于飘走）。

6. alloff [分类: 驱动器]

格式: void alloff();

关闭所有电机，ao 是 alloff 函数的简写形式。

7. analog [分类: 传感器]

格式: int analog(int p);

返回指定端口 p 的传感器值。该值范围是 0~255。此函数只可使用于端口 0~7。

8. analog10 [分类: 传感器]

格式: int analog10(int p);

10 位的 analog 函数。返回值范围为 0~1023，而不是 0~255。

9. any_button [分类: 传感器]

格式: int any_button();

任何一个按键被按下（边侧按键以及 6 个软按键）则返回 1。

10. ao　　[分类: 驱动器]

格式: void ao();

关闭所有电机，等价于 alloff()。

11. atan　　[分类: Math]

格式: double atan(double angle);

返回角度的正切值。角度的单位是弧度，结果的单位是“1”。

12. b_button　　[分类: 传感器]

格式: int b_button();

读取 b 按钮的值。

13. b_button_clicked　　[分类: 传感器]

格式: int b_button_clicked();

获取 b 按钮的状态（按下或未按下）。如果按下按键则阻塞程序直到按键被释放。按下，返回 1；不按下则返回 0；按下不释放则程序暂停。语句

```
while (b_button()==0) {
while (b_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (b_button_clicked()==0) {. . .}
```

14. beep　　[分类: Output]

格式: void beep();

产生蜂鸣后返回。

15. bk　　[分类: 驱动器]

格式: void bk(int m);

将 m 电机全速后退运行。

举例：

```
bk(1);
```

16. block_motor_done　　[分类: 驱动器]

格式: void block_motor_done(int m);

只有当指定电机完成了速度控制或位置控制运动，函数才返回。

举例：

```
mrp(0,500,20000L);
block_motor_done(1);
```

17. bmd [分类: 驱动器]

格式: void bmd(int m);

同 block_motor_done。

18. c_button [分类: 传感器]

格式: int c_button();

读取 c 按钮的值。

19. c_button_clicked [分类: 传感器]

格式: int c_button_clicked();

获取 c 按钮的状态（按下或未按下）。如果按下按键则阻塞程序直到按键被释放。按下，返回 1；不按下则返回 0；按下不释放则程序暂停。语句

```
while (c_button()==0) {
while (c_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (c_button_clicked()==0) {. . .}
```

20. console_clear [分类: 输出]

格式: void console_clear();

清除 Link 的屏幕输出。另见 display_clear。

21. display_clear [分类: 输出]

格式: void display_clear();

清除 Link 屏幕的输出以及缓存。另见 console_clear.

22. display_printf [分类: 输出]

格式: void display_printf(int col, int row, char s[], ...);

在屏幕的指定行列处开始进行输出。 (列 0~41, 行 0~9 ， 如果 d、e、f 按钮被显示则行数相应减少) 。

23. clear_motor_position_counter [分类: 驱动器]

格式: void clear_motor_position_counter(int motor_nbr);

将指定电机的位置计数器清零。

24. cos [分类: Math]

格式: double cos(double angle);

返回角度的余弦值。角度单位为弧度。

25. digital [分类: 传感器]

格式: int digital(int p);

返回端口 p 的数字传感器值，为真为 1，为假为 0.

传感器一般是工作态的电压为 0 ，即当他们处于工作时或条件真时，返回值为 1.因此传感器返回值与实际工作状态相反：如果电压为 0 或逻辑 0，函数返回 1. 数字端口为 8~15.

26. disable_servo [分类:舵机]

格式: void disable_servo(int p);

关闭指定舵机端口。

27. disable_servos [分类: 舵机]

格式: void disable_servos();

关闭所有舵机端口。

28. enable_servo [分类: 舵机]

格式: void enable_servo(int p);

开启指定舵机端口。

29. enable_servos [分类: 舵机]

格式: void enable_servos();

开启所有舵机端口。

30. exp10 [分类: 数学]

格式: double exp10(double num);

返回 10 的 num 数幂。

31. exp [分类: 数学]

格式: double exp(double num);

返回 e 的 num 数幂。

32. extra_buttons_show [分类: 舵机]

格式: void extra_buttons_show();

显示 X,Y,Z 三个软按钮。注意这将减少屏幕显示的行数。参见 extra_buttons_hide，get_extra_buttons_visible。

33. extra_buttons_hide [分类: 舵机]

格式: void extra_buttons_hide();

隐藏 X,Y,Z 三个软按钮。 注意这是默认状态。

参见 **extra_buttons_show, get_extra_buttons_visible**。

34. fd [分类: 驱动器]

格式: void fd(int m);

令指定电机 m 全速前进。

举例：

fd(3);

35. freeze [分类: 驱动器]

格式: void freeze(int m);

冻结指定电机(立即令电机停止转动，与 off 函数不同，off 对电机断电但允许电机继续以惯性转动一定角度)。

36. get_analog_pullup [分类: 传感器]

格式: int get_analog_pullup(int port);

如果指定模拟端口设置了上拉电阻（默认设置）则返回 1。参见 set_analog_pullup。

37. get_extra_buttons_visible [分类: 传感器]

格式: int get_extra_buttons_visible();

如果 X、Y、Z 按钮可见则返回 1，反之为 0。参见 extra_buttons_show, extra_buttons_hide。

38. get_motor_done [分类: 驱动器]

格式: int get_motor_done(int m);

无论电机是否以到达指定位置都立即返回。

39. get_motor_position_counter [分类: 驱动器]

格式: int get_motor_position_counter(int m);

返回指定电机的现在位置计数器。(该值根据每个电机的反电动势值自动动态更新，单位为 ticks，典型值可取 1100 ticks/转)。

40. get_pid_gains [分类: 驱动器]

格式: int get_pid_gains(int motor, int *p, int *i, int *d, int *pd, int *id, int *dd);

该函数用于获取电机使用的 PID 参数。 p, i 和 d 分别是比例(P)、积分(I)和 D 微分(I)控制参数的分子，pd、id、dd 分别是 PID 参数的分母。因此所有的参数均为整型，而实际使用的参数却成为了浮点型。如果某个电机过于灵敏，则应该降低 P、D 值；反之，如果电机的动态响应很慢，则 P、D 参数应增加。默认值在固件安装时便已经给定，参见 set_pid_gains。

41. get_servo_enabled [分类: 舵机]

格式: int get_servo_enabled(int srv);

指定舵机端口如果是打开状态，则返回 1，反之返回 0，舵机端口号为 0~3。参见 enable_servo，disable_servo。

42. get_servo_position [分类: 舵机]

格式: int get_servo_position(int srv);

返回指定端口的舵机位置值。范围是 0~2047，舵机端口号为 0~3。

43. log10 [分类: 数学]

格式: double log10(double num);

返回给定数字的以 10 为底的对数值。

44. log [分类: 数学]

格式: double log(double num);

返回给定数以 e 为底的对数值。

45. mav [分类: 驱动器]

格式: void mav(int m, int vel);

同 move_at_velocity。

46. motor [分类: 驱动器]

格式: void motor(int m, int p);

令指定端口 m 上的电机以指定占空比 p 的 PWM 波来驱动，p 取值范围为 -100~100。100 为全速前进，-100 为全速后退。

47. move_at_velocity [分类: 驱动器]

格式: void move_at_velocity(int m, int vel);

令端口 m 上的电机以指定速度 vel 运行。速度范围是-1000~1000ticks 每秒。

48. move_relative_position [分类: 驱动器]

格式: void move_relative_position(int m, int vel, int ticks);

令指定端口 m 上的的电机以指定速率 vel 从当前位置运动到当前位置值加 ticks 值所得的位置（如果 ticks 值为负值，则电机反转）。速率范围为 0~1000ticks 每秒。举例：

```
move_relative_position(1,275,-1100L);
```

49. move_to_position [分类: 驱动器]

格式: void move_to_position(int m, int vel, int pos);

将指定端口的电机以指定速率运行到指定位置。速率范围是 0~1000ticks 每秒，如果一开始电机就处于该位置，则电机将不转动。

50. mrp [分类: 驱动器]

格式: void mrp(int m, int vel, int pos);

同 move_relative_position.

51. mtp [分类: 驱动器]

格式: void mtp(int m, int vel, int pos);

同 move_to_position.

52. msleep [分类: Time]

格式: void msleep(int msec);

令程序暂停指定时间，时间单位是毫秒。举例：

```
msleep(1500); // 暂停 1.5 秒
```

53. off [分类: 驱动器]

格式: void off(int m);

关闭指定端口 m 上的电机。

举例：

off(1);

54. power_level [分类:传感器]

格式: double power_level();

返回电源电压值。

55. printf [分类: 输出]

格式: void printf(char s[], ...);

在屏幕的光标处显示指定字符串。详见 IDE 帮助。

56. random [分类: 数学]

格式: int random(int m);

随机返回 0 到某个大值 m 之间的任意一个整数值。

57. run_for [分类: 程序]

格式: void run_for(double sec, void <function_name>);

该函数接受另一个指定函数作为参数，令其运行给定时间。如果指定函数在给定时间之前就退出，则 run_for 函数将在其退出 1 秒内返回。参数 sec 用于给定指定函数的运行时间，单位为秒。

58. seconds [分类: 时间]

格式: double seconds();

返回一个用浮点数据类型表示的系统时间，单位是秒。分辨率为 1 毫秒。

59. set_a_button_text [分类: 传感器]

格式: void set_a_button_text(char txt[]);

将 a 按钮的显示更改为指定字符，而不再是 a。

60. set_analog_pullup [分类: 传感器]

格式: void set_analog_pullup(int port, int pullupTF);

该函数用于决定指定模拟端口是否启用上拉电阻。不启用上拉电阻的端口为浮动类型。如 set_analog_pullup(3,0);将端口 3 设置为浮动类型（不启用上拉电阻）。由于大部分传感器都没有内置的上拉电阻（ET 传感器例外），因此 Link 开启后的端口默认状态为启用上拉电阻，但在使用 ET 传感器前必须调用此函数令端口设置为浮动类型。程序退出后端口将自动恢复默认状态。

61. set_b_button_text [分类: 传感器]

格式: void set_b_button_text(char txt[]);

将 b 按钮的显示更改为指定字符，而不再是 b。

62. set_c_button_text [分类: 传感器]

格式: void set_c_button_text(char txt[]);

将 c 按钮的显示更改为指定字符，而不再是 c。

63. set_digital_output [分类：输出]

格式: void set_digital_output(int port, int inout);

Link 的数字端口可以设置为输入或输出。默认情况下为输入。函数 set_digital_output(9, 1)即将数字端口 9 设置为输出。数字端口为 9~15。

64. set_digital_pullup [分类: 传感器]（*高级内容，一般情况下不涉及）

格式: void set_digital_pullup(int port,int pullupTF);

将数字端口设置为启用上拉电阻。

65. set_digital_value [分类: 输出]

格式: void set_digital_value(int port, int value);

当一个数字端口被设置为输出后，可使用该函数来设置输出值 0 或 1。数字端口号 8~15。

66. set_pid_gains [分类: 驱动器] （*高级内容）

格式: int set_pid_gains(int motor, int p, int i, int d, int pd, int id, int dd);

该函数用于设置电机使用的 PID 参数。 p、i 和 d 分别是比例(P)、积分(I)和 D 微分(I)控制参数的分子，pd、id、dd 分别是 PID 参数的分母。因此所有的参数均为整型，而实际使用的参数却成为了浮点型。如果某个电机过于灵敏，则应该降低 P、D 值；反之，如果电机的动态响应很慢，则 P、D 参数应增加。默认值在固件安装时便已经给定，也可在 Link 中从用户界面调整。

67. set_servo_position [分类: 舵机]

格式: int set_servo_position(int srv, int pos);

将舵机转动到指定位置。位置值 pos 为 0~2047，舵机号 srv 为 0~3。

68. set_x_button_text [分类: 传感器]

格式: void set_x_button_text(char txt[]);

将 x 钮的显示更改为指定字符（串）以代替 x。另见 extra_buttons_hide, get_extra_buttons_visible。

69. set_y_button_text [分类: 传感器]

格式: void set_y_button_text(char txt[]);

将 y 钮的显示更改为指定字符（串）以代替 y。另见 extra_buttons_hide, get_extra_buttons_visible。

70. set_z_button_text [分类: 传感器]

格式: void set_z_button_text(char txt[]);

将 z 钮的显示更改为指定字符（串）以代替 z。另见 extra_buttons_hide, get_extra_buttons_visible。

71. setpwm [分类: 驱动器]

格式: int setpwm(int m, int dutycycle);

令指定电机 m 以指定的占空比 dutycycle 运行（取值范围-100~100）。

72. side_button (or black_button)　[分类: 传感器]

格式: int side_button();

读取边侧传感器的值（0 或 1）。

73. side_button_clicked　[分类: 传感器]

格式: int side_button_clicked();

获取边侧按钮的状态（按下或未按下）。如果按下并保持再释放，返回 1，不按下则返回 0，按下不释放则程序暂停。语句

```
while (side_button()==0) {
while (side_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (side_button_clicked()==0) {. . .}
```

74. sin　[分类: 数学]

格式: double sin(double angle);

返回给定角度 angle 的正弦值。角度单位为弧度。

75. sqrt　[分类: 数学]

格式: double sqrt(double num);

返回给定值 num 的开方值。

76. tan　[分类: 数学]

格式: double tan(double angle);

返回给定角度 angle 的正切值。角度单位为弧度。

77. thread_create　[分类: 线程] （*高级内容）

格式: thread thread_create(<function name>);

该函数用于为其它函数创建一个和主函数并行运行的线程。函数返回一个线程号，数据类型为 thread。这种数据类型用于记录线程号，系统用它来记录活动线程。注意，为了让返回值有效必须把它赋值给一个 thread 类型的变量。当线程里的函数开始执行（通过调用 thread_start），在线程函数结束返回或被销毁之前，线程将一直处于活动状态。如果线程没有被销毁，该线程可以被再次启动，

否则，为了运行该线程函数，用户将必须新建另一个线程。

78. thread_destroy [分类: 线程] （*高级内容）

格式: void thread_destroy(thread id);

该函数用于销毁一个线程，即通过将该线程的线程号提供给 thread_destroy 函数此线程即被销毁。下面的例子示例了主函数通过调用 thread_create 为 check_sensor 函数创建一个线程，然后 1 秒钟后不管该线程是否活动立即摧毁它的过程：

```
int main() {
    thread tid;
    tid = thread_create(check_sensor);
    thread_start(tid);
    msleep(1000);
    thread_destroy(tid);
}
```

79. thread_start [分类: 线程] （*高级内容）

格式: void thread_start(thread id);

该函数用于开启一个线程，令该线程函数与主函数并行执行。函数的参数必须是由 thread_create 返回的线程号。线程号必须为 thread 数据类型，线程将一直处于活动状态直到线程函数自己结束返回或通过指令被销毁。

80. thread_wait [分类: 线程] （*高级内容）

格式: void thread_wait(thread id);

该函数用于等待由 thread_start 开启的线程的结束，参数 id 为该线程的线程号。

81. x_button [分类: 传感器]

格式: int x_button();

读取软按键 x 按钮的值，该按钮为附加按钮。可通过调用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

82. x_button_clicked [分类: 传感器]

格式: int x_button_clicked();

获取 x 按钮的状态（按下或未按下） 。如果按下并保持再释放，返回 1，不按下则返回 0，按下不释放则程序暂停。语句

```
while (x_button()==0) {
while (x_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (x_button_clicked()==0) {. . .}
```

该按钮为附加按钮。可使用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

83. y_button [分类: 传感器]

格式: int y_button();

读取软按键 x 按钮的值，该按钮为附加按钮。可通过调用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

84. y_button_clicked [分类: 传感器]

格式: int y_button_clicked();

获取 y 按钮的状态（按下或未按下） 。如果按下并保持再释放，返回 1，不按下则返回 0，按下不释放则程序暂停。语句

```
while (y_button()==0) {
while (y_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (y_button_clicked()==0) {. . .}
```

该按钮为附加按钮。可使用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

85. z_button [分类: 传感器]

格式: int z_button();

读取软按键 z 按钮的值，该按钮为附加按钮。可通过调用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

86. z_button_clicked [分类: 传感器]

格式: int z_button_clicked();

获取 z 按钮的状态（按下或未按下） 。如果按下并保持再释放，返回 1，不按下则返回 0，按下不释放则程序暂停。语句

```
while (z_button()==0) {
while (z_button()==1); . . .} //用于消除按键抖动
```

等价于

```
while (z_button_clicked()==0) {. . .}
```

该按钮为附加按钮。可使用函数 extra_buttons_show() 来显示。另见 extra_buttons_hide，get_extra_buttons_visible。

6.11 Link 的视觉系统库函数

1. camera_close [分类: 摄像头]

格式: void camera_close();

关闭摄像头。另见 camera_open，camera_open_device。

2. camera_load_config [分类: 摄像头]

格式: int camera_load_config(char name[]);

在 Link 中载入配置文件并以之替代默认配置文件。文件名应以.config 为后缀。载入成功函数返回 1，反之为 0。另见 camera_open，camera_open_device。

3. camera_open [分类: 摄像头]

格式: int camera_open(int res_numb);

以 Link 默认的通道配置开启摄像头。可通过 Link 上的“Settings(设置)→Channels(通道)”界面进行配置。函数需通过 res_numb 参数指定分辨率为 **LOW_RES(低分辨率), MED_RES(中分辨率)或 HIGH_RES(高分辨率)**。开启成功函数返回 1，反之为 0。另见 camera_open_device, camera_close。

4. camera_open_device [分类: 摄像头]

格式: int camera_open_device(int number, int res_numb);

如果有两个摄像头接入，number 参数指定摄像头编号，0 号摄像头是首选，其次为 1 号。一次只能使用一个摄像头。同时选择默认配置，选定分辨率。另见 camera_open， camera_close。

5. camera_update [分类: 摄像头]

格式: int camera_update();

将传回最新的图像信息给程序进行处理。成功返回 1，反之为 0.

6. get_channel_count [分类: 摄像头]

格式: int get_channel_count();

返回现正在使用的配置中的通道数。另见 get_object_count。

7. get_code_num [分类: 摄像头]

格式: int get_code_num(int channel, int object);

以一个整数的形式返回和给定通道和给定编号的目标物相关的数据。如两者不存在，返回-1，另见 get_object_data。

8. get_object_area [分类: 摄像头]

格式: int get_object_area(int channel, int object);

返回给定通道上的给定编号目标物的边界框的面积。如果不存在，则返回-1。

9. get_object_bbox [分类: 摄像头]

格式: rectangle get_object_bbox(int channel, int object);

返回给定通道上的给定编号目标物的边界框数据，结果为一个矩形数据类型。举例：

```
rectangle mybox;
mybox = get_object_bbox(0,2);
printf("x coord %d y coord %d\n", mybox.x, mybox.y);
```

上述代码将显示通道 0 上 2 号物体的边框的 x 和 y 坐标。

10. get_object_center [分类: 摄像头]

格式: point2 get_object_center(int channel, int object);

以 point2 的数据类型返回指定通道的给定编号目标物的中心坐标。举例：

```
point2 cntr;
cntr = get_object_center(0,2);
printf("Center: x coord %d y coord %d\n", cntr.x, cntr.y);
```

上述代码将显示通道 0 上的 2 号目标物的外接边框中心坐标。

11. get_object_centroid [分类: 摄像头]

格式: int get_object_centroid(int channel, int object);

以 point2 的数据类型返回指定通道的给定编号目标物的形心坐标。（形心是该颜色像素块的中心）举例：

```
point2 cntd;
cntd = get_object_centroid(0,2);
printf("centroid: x coord %d y coord %d\n", cntd.x, cntd.y);
```

上述代码将显示通道 0 上的 2 号目标物的像素块中心坐标。

12. get_object_confidence　　[分类: 摄像头]

格式: double get_object_confidence(int channel, int object);

返回指定通道的给定编号目标物的辨识可信度，其值为 0.0 到 1.0。如果通道或目标物不存在，返回 0.0。

13. get_object_count　　[分类: 摄像头]

格式: int get_object_count(int channel);

返回给定通道上识别到的目标物的总数量。根据目标物的面积进行派讯，面积大的序号小。如果通道或目标物不存在，则返回-1。另见 get_channel_count。

14. get_object_data　　[分类: 摄像头]

格式: char *get_object_data(int channel, int object);

返回二维码通道识别出的字符串数据。如果无数据关联则返回 0。此字符串数据并不一定以 null 结尾，但可以通过数组来读取。如：get_object_data(0,0)[0]; get_object_data(0,0)[1]。camera_update 将使该指针指向的数据失效。另见 get_object_data_length。

15. get_object_data_length　　[分类: 摄像头]

格式: int get_object_data_length(int channel, int object);

返回对指定通道的给定编号物体所识别的 QR 二维码的字符串长度。如果没有数据或通道或二维码数据无效，则返回 0。另见 get_object_data。

6.12 KIPR Link 中用于控制 iRobot Create 的库函数

6.12.1 Create 串口相关函数

1. create_clear_serial_buffer [分类：Create 串口相关]

格式：void create_clear_serial_buffer ();

清除串口缓冲区中一切尚未读取/发送的数据。

2. create_connect [分类: Create 串口相关]

格式: int create_connect();

建立 Link 到 Create 的串口连接。这个语句一般会和 msleep 共同使用，因为需要超过 1 秒钟的时间，令 Link 和 Create 建立连接(一般调用 msleep(1500)就足够了)。如果程序执行时 Create 并没有被开启，这个函数将会阻塞程序的继续执行，直至 Create 开启才继续。如果连接成功则返回 0，反之为一个负数。调用此函数是 Link 连接 Create 的第一个步骤，该函数将使 Create 处于默认的 create_safe 安全模式。

3. create_disconnect [分类: Create 函数]

格式: void create_disconnect();

断开 Link 与 Create 的连接，并停止任何正在运动的电机。

4. create_read_block [分类: Create 串口函数]

格式: int create_read_block(char *data, int count);

通过串口，令 Create 发送给定字节数 count 的字节数据，数据以字符串 data 的形式发送。读取成功返回 1，否则为 0。

5. Create_write_byte [分类: Create 串口函数]

格式: void Create_write_byte (char byte);

通过串口，由 Link 向 Create 发送一个字节的数据 byte。

6.12.2 Create 配置函数

1. create_full [分类: Create 函数]

格式: void create_full();

完全模式，Create 将执行所有指令，无论发生任何情况（如：传感器感知到跌落）也不会自动停止。

2. create_passive [分类: Create 函数]

格式: void create_passive();

使 Create 进入被动模式（电机不可用）。

3. create_safe [分类: Create 函数]

格式: void create_safe();

安全模式，Create 将执行所有指令，但是当自带的传感器感知到跌落或到达（桌子）边缘时，将自动断开连接并停止。

4. create_start [分类: Create 函数]

格式: void create_start();

使 Create 进入运动模式(电机可用)。

5. get_create_mode [分类: Create 函数]

格式: int get_create_mode();

返回 Create 的当前模式(0=关闭; 1=被动; 2=安全; 3=完全)。在被动模式下，电机运动命令将不工作。在安全和完全模式下所有命令都可工作。在安全模式下，如果场地边缘传感器感知到桌面边缘（因此 Create 有跌落危险）或轮降传感器激发（同样认为 Create 有跌落危险）则自动停止电机并断开连接。在完全模式下，Create 不管传感器读到任何值都继续工作，而不会自动停止。

6.12.3 Create 运动函数

1. create_drive [分类: Create 运动函数]

格式: void create_drive(int speed, int radius);

驱动 Create 转弯前进，speed 为速度，radius 为半径。速度范围是 20~ 500mm/s。

2. create_drive_direct [分类: Create 运动函数]

格式: void create_drive_direct(int r_speed, int l_speed);

r_speed 和 l_speed 分别为为左右轮指定的速度，单位为 mm/s。

3. create_drive_straight [分类: Create 运动函数]

格式: void create_drive_straight(int speed);

驱动 Create 以 speed 速度直线行驶，单位为 mm/s。

4. create_spin_CW [分类: Create 运动函数]

格式: void create_spin_CW(int speed);

顺时针方向旋转，边缘的速度为 speed 单位为 mm/s。

5. create_spin_CCW [分类: Create 运动函数]

格式: void create_spin_CCW(int speed);

逆时针方向旋转，边缘的速度为 speed 单位为 mm/s。

6. create_stop [分类: Create 运动函数]

格式: void create_stop();

所有电机停止转动。

7. get_create_distance [分类: Create 运动函数]

格式: int get_create_distance();

返回 Create 从开始运动或距离值被重置后到目前累计运动的距离。后退时该值将减小。单位是 mm。

8. get_create_normalized_angle [分类: Create 运动函数]

格式: int get_create_normalized_angle();

返回 Create 从开始运动或角度值被重置后到目前所转过的累计角度——即标准化的 0~359 度。逆时针方向转动将增加该值，反之减小。

9. get_create_overcurrents [分类: Create 运动函数] （*高级内容，实际很少使用）

格式: int get_create_overcurrents();

返回一个 16 位的表示过载电流状态值：第 16 位表示左轮的过载电流，第 8 位表示右轮，第 4 位表示 LD2，第 2 位表示 LD0，第 1 位表示 LD1。

10. get_create_requested_left_velocity [分类: Create 运动函数]

格式: int get_create_requested_left_velocity();

返回为 Create 指定的左轮速度值，-500~500mm/s。

11. get_create_requested_radius [分类: Create 运动函数]

格式: int get_create_requested_radius();

返回给 Create 指定的转弯半径，返回值的范围为-2000 到 2000，单位为 mm。

12. get_create_requested_right_velocity [分类: Create 运动函数]

格式: int get_create_requested_right_velocity();

返回给 Create 指定的右轮速度值，-500~500mm/s。

13. get_create_requested_velocity [分类: Create 传感器函数]

格式: int get_create_requested_velocity();

返回给 Create 指定的速度值，-500~500mm/s。

14. get_create_total_angle [分类: Create 运动函数]

格式: int get_create_total_angle();

返回 Create 从开始运动或上一次角度重置后直至目前所转过的累计角度。逆时针方向转动将增加该值，反之减小。

15. set_create_distance [分类: Create 运动函数]

格式: void set_create_distance(int dist);

设置累计距离的值至 dist。

16. set_create_normalized_angle [分类: Create 运动函数]

格式: void set_create_normalized_angle(int angle);

设置累计标准化角度的值至 angle。

17. set_create_total_angle [分类: Create 运动函数]

格式: void set_create_total_angle(int angle);

设置累计角度的值至 angle。

6.12.4 Create 传感器函数

1. get_create_advance_button [分类: Create 传感器函数]

格式: int get_Create_advance_button();

如果前进按键(>>|)被按下则返回 1，反之为 0。

2. get_create_bay_AI [分类: Create 传感器函数] （*高级内容）

格式: int get_create_bay_AI();

返回 Create 接口第 4 脚的 10 位模拟值。

3. get_create_bay_DI [分类: Create 传感器函数] （*高级内容）

格式: int get_create_bay_DI();

返回一个字节，表示 cargo bay 全部数字传感器信息：第 16 位表示 15 脚，第 8 位表示 6 脚，4 位表示 18 脚，2 位表示 5 脚，1 位表示 17 脚。

4. get_create_cwdrop [分类: Create 传感器函数]

格式: int get_create_cwdrop();

如果中央脚轮悬空返回 1，否则为 0。

5. get_create_infrared [分类: Create 传感器函数] （*高级内容）

格式: int get_create_infrared();

返回远程控制探测到的红外线数据。如果没有探测到数据则返回 255。

6. get_create_lbump [分类: Create 传感器函数]

格式: int get_create_lbump();

如果左侧碰撞传感器按下，返回 1，否则为 0。

7. get_create_lcliff [分类: Create 传感器函数]

格式: int get_create_lcliff();

如果左侧边缘传感器（红外反射传感器）处于不反射红外线的表面之上（如黑色表面）或探测到边缘（悬空，如处于桌面边缘）函数返回 1，否则为 0。

8. get_create_lcliff_amt [分类: Create 传感器函数]

格式: int get_create_lcliff_amt();

返回左侧边缘传感器（红外反射传感器）的 12 位模拟值（0 到 4095）。

9. get_create_lfcliff [分类: Create 传感器函数]

格式: int get_create_lfcliff();

如果左前侧边缘传感器（红外反射传感器）处于不反射红外线的表面之上（如黑色表面）或探测到边缘（悬空，如处于桌面边缘）函数返回 1，否则为 0。

10. get_create_lfcliff_amt [分类: Create 传感器函数]

格式: int get_create_lfcliff_amt();

返回左前侧边缘传感器（红外反射传感器）的 12 位模拟值（0 到 4095）。

11. get_create_lwdrop [分类: Create 传感器函数]

格式: int get_create_lwdrop();

如果左轮悬空返回 1，否则为 0。Create 中有两个轮夹，可以用于阻止轮子悬空。

12. get_create_number_of_stream_packets [分类: Create 传感器函数]

（*高级内容）

格式: int get_create_number_of_stream_packets();

如果数据流正在被使用，返回数据流的长度。

13. get_create_play_button [分类: Create 传感器函数]

格式: int get_create_play_button();

如果运行按键（>）被按下则返回 1，反之为 0。

14. get_create_rbump [分类: Create 传感器函数]

格式: int get_create_rbump();

如果右侧碰撞传感器按下，返回 1，否则为 0。

15. get_create_rcliff [分类: Create 传感器函数]

格式: int get_create_rcliff();

如果右侧边缘传感器（红外反射传感器）处于不反射红外线的表面之上（如黑色表面）或探测到边缘（悬空，如处于桌面边缘）函数返回 1，否则为 0。

16. get_create_rcliff_amt [分类: Create 传感器函数]

格式: int get_create_rcliff_amt();

返回右侧边缘传感器（红外反色传感器）的 12 位模拟值（0 到 4095）。

17. get_create_rfcliff [分类: Create 传感器函数]

格式: int get_create_rfcliff();

如果右前侧边缘传感器（红外反射传感器）处于不反射红外线的表面之上（如

黑色表面）或探测到边缘（悬空，如处于桌面边缘）函数返回 1，否则为 0。

18. get_create_rfcliff_amt　　[分类: Create 传感器函数]

格式: int get_create_rfcliff_amt();

返回左前侧边缘传感器（红外反色传感器）的 12 位模拟值（0 到 4095）。

19. get_create_rwdrop　　[分类: Create 传感器函数]

格式: int get_create_rwdrop();

如果右轮悬空返回 1，否则为 0。Create 中有两个轮夹，可以用于阻止轮子悬空。

20. get_create_wall　　[分类: Create 传感器函数] (*高级内容)

格式: int get_create_wall();

如果右侧墙壁传感器探测到了墙壁，函数返回 1，否则为 0（不存在左侧墙壁传感器）。

21. get_create_vwall　　[分类: Create 传感器函数] (*高级内容)

格式: int get_Create_vwall();

如果探测到了一个虚拟墙壁信号源，函数返回 1，否则为 0。

22. get_create_wall_amt　　[分类: Create 传感器函数] (*高级内容)

格式: int get_create_wall_amt();

返回墙壁传感器的 12 位模拟值（0 到 4095）。

6.12.5 Create 电池函数

1. get_create_battery_capacity　　[分类: Create 传感器函数]

格式: int get_create_battery_capacity();

返回电池容量，单位 mAh.

2. get_create_battery_charge　　[分类: Create 传感器函数]

格式: int get_create_battery_charge();

返回电池电量，单位 mAh.

3. get_create_battery_charging_state　　[分类: Create 传感器函数] （*高

级内容，实际中很少用到）

格式: int get_create_battery_charging_state();

返回电池充电状态。0-未充电; 1-重新充电; 2-充满电; 3-微电流充电; 4-等待; 5-充电失败。

4. get_create_battery_current [分类: Create 传感器函数]

格式: int get_create_battery_current();

返回电池电流，单位 mA.

5. get_create_battery_voltage [分类: Create 传感器函数]

格式: int get_create_battery_voltage();

返回电池电压，单位 mV。

6. get_create_battery_temp [分类: Create 传感器函数]

格式: int get_create_battery_temp();

返回电池温度，单位 C。

6.12.6 Create 内置函数

1. create_spot [分类: Create 函数]

格式: void create_spot();

模仿 Roomba 吸尘器机器人的定点清扫程序。

2. create_cover [分类: Create 函数]

格式: void create_cover();

模仿 Roomba 吸尘器机器人的覆盖一个房间的程序。

3. create_demo [分类: Create 函数]

格式: void create_demo(int d);

运行编号为 d 的内置演示程序（详见 Create 开放接口手册）。

4. create_cover_dock [分类: Create 函数]

格式: void create_cover_dock();

此程序令 Create 不断移动直到它感应到红外信号源，然后自动停驻于信号源

处。

6.12.7 LED 和音乐函数

1. create_advance_led [分类: Create LED 和音乐函数]

格式: void Create_advance_led(int on);

输入 on 置为 1 可点亮前进 LED 灯(>>|)，置 0 则令之熄灭。

2. create_play_led [分类: Create LED 和音乐函数]

格式: void create_play_led(int on);

输入 on 置为 1 可点亮运行 LED 灯(>)，置 0 则令之熄灭。

3. create_power_led [分类: Create LED 和音乐函数]

格式: void create_power_led(int color, int brightness);

改变 power LED 灯的颜色和亮度，color 参数 0 表示红色，255 表示绿色；brightness 参数 0 表示熄灭，255 表示最亮。

4. get_create_song_number [分类: Create LED 和音乐函数]

格式: int get_create_song_number();

返回目前所选择的歌曲编号：0~15。

5. get_create_song_playing [分类: Create LED 和音乐函数]

格式: int get_create_song_playing();

如果正在播放歌曲，返回 1，反之为 0。

6. create_load_song [分类: Create LED 和音乐函数]

格式: void create_load_song(int num);

从内部的 gc_song_array[16][33]矩阵中挑选一首歌曲播放，每一行是一首歌，num 取值 0~15。每首歌的第一列是音符的序号（最大为 16）。剩余的列为音高和音长交替的值。详见 Create 开放接口网站介绍。

7. create_play_song [分类: Create LED 和音乐函数]

格式: void Create_play_song(int num);

播放目前所载入的歌曲。