中国教育国际交流协会

传感器和驱动器使用手册

BotBall 中国赛区组委会

版本：Botball 2011

目录

第一章 概况

1.1 如何使用本手册

这本手册的目的是说明如何使用所有的 BotBall 驱动器和传感器。你可以通过阅读本手册来了解如何使用驱动器和传感器的基本知识。每个部分包含一种类型的设备，以及套件中提供的所有的 BotBall 配套原件设备的信息。

本手册也可以作为一个参考源。如果你发现自己在使用某个传感器或者驱动器时有困难，你就可以拿起本手册，在目录中查找这个让你头疼的设备，并且翻到所在页。

1.2 什么是驱动器？

驱动器把电能转换为机械能。BotBall 提供了两种不同类型的驱动器：连续旋转驱动器和位置伺服驱动器。当通电时，连续旋转驱动器不断的旋转。位置伺服驱动器则只能旋转 180 度。这两种驱动器通过电源和信号指示来决定朝哪个方向旋转。

CBC 有 4 个连续选择驱动器端口和 4 个位置伺服驱动器端口，在 CBC 前端每侧有 2 个。

1.3 什么是传感器？

传感器是把测量的物理量转为可观信号的设备。对于 BotBall，传感器将返

回 0V 和 5V 之间的直流电压给 CBC。CBC 把该电压分成 2^8 或 2^{10} 倍，并产生比例于从传感器得到的反馈量的一个数。两个传感器可以返回相同的值，但可能意味着不同的物理量。

CBC 具有 8 个模拟和 8 个数字传感器端口。通过程序或手动设置，所有的模拟端口都可以设置为浮动模拟端口。传感器被连接到 CBC 的前端的插口上。请自行确保模拟传感器插入模拟端口并且数字传感器插入数字接口。

CBC 具有内置的加速度传感器。加速度传感器可以感知三个方向的加速度。

第二章 驱动器

在 BotBall 套件中的连续选择驱动器的电机插头可以以任何方向插入驱动器端口。当你指示驱动器向前驱动，它将旋转且蓝灯会亮（红灯则指示驱动器正在反向驱动）。如果你希望驱动器转向相反的方向，那么可以拔出电机，将连接头 180 度反向插入。

驱动器端口电压为 6V，每个驱动器端口的最大电流为 1A。每一组端口（0

和 1，2 和 3）是由一个单一的 H 桥电路控制。所以如果你需要使用 2 个大电流（接近 1A）的驱动器，则应该把它们分别插入不同的 H 桥电路控制的端口（比如 0 和 3，或 1 和 3 等）以延长 H 桥寿命。

2.1 CBC 是如何感知驱动器的位置的？

CBC 采用闭环反电动势 PID 控制系统。闭环意味着 CBC 可以监测驱动器的位置。CBC 用 PWM 命令驱动电动机，PWM 是脉冲宽度调制。用这种方法你可以改变驱动器的转速。如果 PWM 输出 100%则为全电压驱动，驱动器将全速全功率运动。如果 PWM 输出 50%则只有 50%占空比的脉冲电压驱动驱动器，它将以一半的速度和全功率运动。当电机的转轴转动时，它还将产生反电动势。CBC 可以测量它，并用闭环控制驱动器的速度。PID 控制器是比例积分微分控制的简称。PID 控制有助于帮助 CBC 提供更好的 PWM 驱动运动功能。

2.2 什么是 tick？

一个 tick 是驱动器最小可测量的转动量。由于 CBC V2 采用反电动势测量驱动器的位置，每个驱动器 tick 的数量是受驱动器的物理性质和内部齿轮的限制的。连续旋转伺服驱动器每圈有大约 1100 个 tick。

CS-60 连续旋转驱动器

性能：

扭矩: 49oz in

速度@ 60°: 0.16 sec

SG-5010 连续旋转驱动器

性能：

扭矩： 156oz in

速度@ 60°: 0.11 sec

2.3 使用驱动器

机器人通常采用驱动器作为运动的动力。驱动机器人有几种不同的方式，但最简单，最常见的是两个车轮直接驱动。连续旋转驱动器可以提供足够的扭矩和足够速度使机器人运动。这种方式下，一个轮子被安装在机器人的一侧的驱动器上，在另一侧安装另一个驱动器和车轮。在机器人的前端或后端安装脚轮，还可以在两侧也安装脚轮来保证机器人的稳定性。这种驱动方式下，只要令 2 个驱动器都向前运动，机器人就向前运动了。转向也很简单，驱动一个驱动器向前而另一个驱动器向后，机器人以两轮的连线中点为轴原地转动。而通过驱动驱动器以不同的速度向前运动，机器人将以一个弧度转向。

在写代码之前，请先测试驱动器的极性（和连接线有关）。（见附录）

2.4 KISS-C 驱动器库函数

在附录中查看完整列表

motor(<motor#>,<power>)

说明 :以一定功率(power, PWM 占空比)驱动电机(motor#为电机编号)。功率范围从 100（向前）到-100（向后）。

mav(<motor#>,<velocity>)

说明：以一定速度驱动连续转动驱动器。速度范围从 1000 到-1000ticks 每秒。

ao()

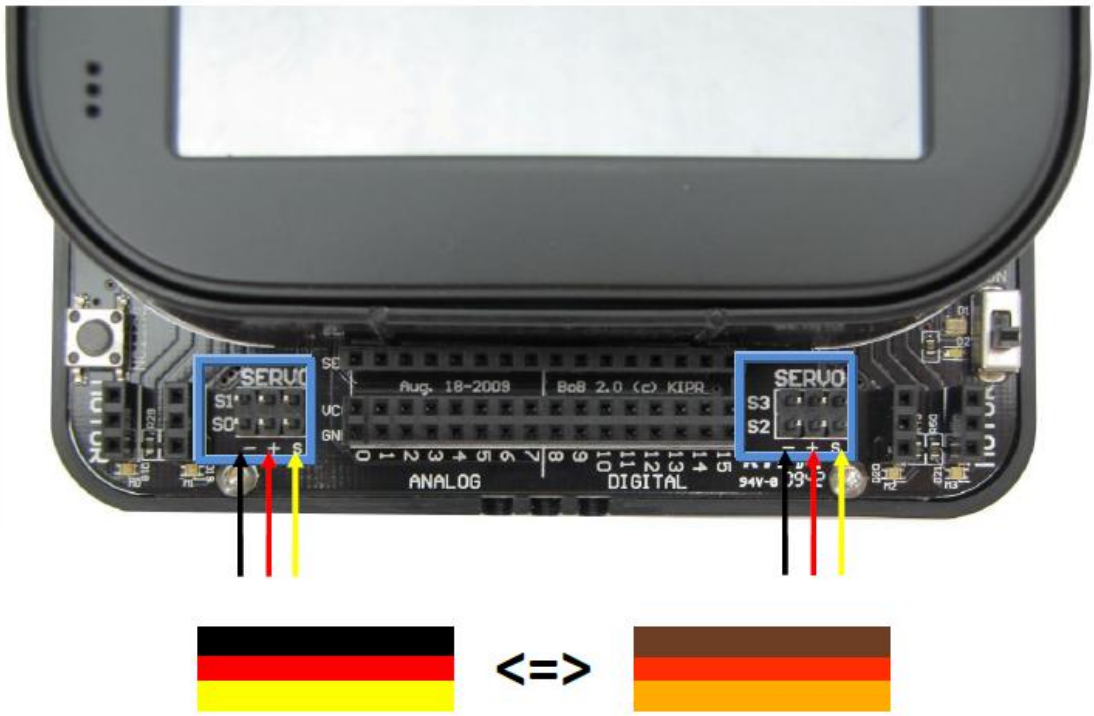
说明：关掉所有电机，关闭所有驱动器的端口电源。

2.5 示例代码

```
/*This program drives a two wheeled direct drive robot
forward for 3 seconds, stops, turns in place for half a
second, and stops for good.  The motors are plugged into
ports 0 and 3 so that the robot moves forward when driven
forward.  If the robot turns in place or drives backwards
at the beginning, unplug the offending motor plug and
rotate it 180 degrees and plug it back in.*/

int main(){
    motor(0,100);//turn on motor in port 0 forward
    motor(3,100);//turn on motor in port 3 forward
    sleep(3.0);  //wait for 3 seconds
    ao();        //turn off both motors
    mav(0,100);  //turn on motor in port 0 forward
    mav(3,-100); //turn on motor in port 3 backward
    sleep(0.5);  //wait for 0.5 seconds
    ao();        //turn off both motors
}
```

第三章 伺服驱动器



位置伺服驱动器连接到 CBC 前端的伺服端口上。位置伺服驱动器的接头是有方向的，上面用到的箭头代表最常见的伺服电缆颜色：地面是黑色或棕色的，正极是红的，信号是黄或橙色。位置伺服驱动器工作在 6 V。

3.1 CBC 如何感知伺服驱动器的位置？

位置伺服驱动器中含有一个控制位置的控制板，以及一个旋转传感器感来感知当前伺服传感器的转动位置。我们需要为位置伺服驱动器提供一个电脉冲信号以指示它应在哪儿定位。伺服驱动器将设法保持在设定的位置直到改变信号或断电。一个标准的伺服驱动器可以选择大约 180 度的范围（分为 2048 个位置），在此范围内它的旋转位置可以控制。

3.1.1 伺服驱动器的预设位置

伺服驱动器总是试图转动到达所设定的位置，即使这意味着它会因此损坏。因此，你需要确保伺服驱动器上的负载不超过它的最大转矩限制，否则就会发生损害。当你第一次调用 enable_servos()命令，如果还没有指定一个位置，则伺服驱动器默认到达中间位置（1024）。

SG-5010 标准伺服驱动器

性能

扭矩: 156oz in

速度@ 60°: 0.11 sec

SG-90 微型伺服驱动器

性能

扭矩: 22oz in

速度@ 60°: 0.11 sec

3.2 使用

伺服驱动器通常使用于机器人的手臂和手爪或其他需要高精度和可重复性装备上。当你的程序设置了一个伺服驱动器的转动位置后，需要等待它转动到达所需的位置。因为伺服驱动器内部有机械装置使其在 0 度和 180 度停止转动，所以有时你的程序有可能令伺服驱动器超过它的转动范围。这意味着内部机械装置将导致伺服驱动器发出嗡嗡声，并可能对伺服驱动器造成永久性损伤。这时，

你应该改变程序中的设定位置来防止驱动器发出嗡嗡声。一个安静的伺服驱动器才是工作良好的伺服驱动器。参见附录找到伺服驱动器的更多信息。

3.3 KISS-C 伺服驱动器库函数

在附录中查看完整列表

enable_servos()

给伺服驱动器端口供电。在控制伺服驱动器运动之前必须先调用。当调用该函数时，伺服驱动器将默认复位到 1024 位置（中间位置），除非在调用 enable_servos()之前已经指示它移动到其他位置。

set_servo_position(<port#>, <position>)

port#为端口号，position 为位置。令伺服驱动器转动到 position 所指示的位置。位置的范围从 0 到 2047。函数调用后，除非受到外力的阻碍，伺服驱动器将立即移动到给定的位置。在使用这个函数前调用 enable_servos()函数。

3.4 示例代码

```
/*This program moves two servos.  Servos need to be plugged
into ports 0 and 3.  The program presets servo 0 to position
150.  Then the program waits for the black button to be
pushed.  Then it enables servos, servo 0 goes to position
150 and servo 3 goes to 1900.  Then servo 3 goes to 1900.
Finally servo 0 goes to 1900 and servo 3 goes to 150 and the
program ends. See appendix for finding your servo's range.

int main(){
    set_servo_position(0,150);     //preset port 0 to 150
    printf("servo 0 at position 0\n");
    printf("press black button to continue\n");
    while(!black_button()){}       //wait for black button
    enable_servos();               //enable the servos
    sleep(1);                      //wait for servo to move
    set_servo_position(3,1900);    //move port 3 to 1900
    sleep(1);                      //wait for servo to move
    set_servo_position(0,1900);    //move port 0 to 1900
    set_servo_position(3,150);     //move port 3 to 150
    sleep(3);                      //wait for servos to move
    disable_servos();              //power down servos
}
```

第四章 模拟传感器

BotBall 的传感器有防插反设计 ,插入 CBC 的端口时只有一种方向才会让所有引脚都插入。

模拟传感器有的接头为两针接头，有的为三针接头。CBC 的端口上的“S”代表传感器传回的信号(SEN)，”V”代表电源正极(VCC)，”G”代表地(GND)。在模拟传感器中 SEN 和 GND 线之间的电阻是变化，连接到 VCC 的接线用来给传感器供电。

将传感器插入到一个模拟端口后，传感器将返回一个电压值。CBC 可以以 10 位或 8 位格式返回模拟值，10 位格式比 8 位的格式精确四倍。

4.1 KISS-C 模拟传感器库函数

analog10(<port#>)

返回 10 位模拟端口读数（取值范围为 0-1023 ）。模拟端口号（ port# ）范围为 0-7。

analog(<port#>)

返回 8 位模拟端口读数（取值范围在 0-255 ）。模拟端口号（ port# ）范围为 0-7。

set_each_analog_state(<port0>,<port1>,<port2>,<port3>,<port4>,<port5>,<port6>,<port7>)

此功能为一高级功能，用于设置模拟端口为浮动型或上拉电阻型。参数 port0 到 port7，为 1 则表示设置对应的端口为浮动型，为 0 则设置对应端口为上拉电阻型。请注意，当 CBC 重新启动或当程序退出时所有的传感器端口都默认复位为非浮动型（ 上拉电阻型 ）。如果你使用的传感器要求必须设置端口为浮动型的，你就需要使用这个函数，如 ET 或声纳传感器。使用时，根据这个短睡眠命令的陈述让发生改变有时间。

4.2 光传感器

- **性能**

角度范围：20 度

光敏感波长：880nm

- **描述**

光传感器可看做是一个可变电阻。光线增强则传感器的电阻加大，CBC 相应读数值降低。

- **使用**

光传感器通常用于在比赛的开始时感受启动灯光，并启动 BotBall 机器人。几个光传感器可排成阵列来跟踪光线的变化，还可以用光源来引导机器人运动。

- **示例代码**

```
/*This program runs the light calibration function.  You
need a light sensor plugged into port 0 and a light to
shine at the light sensor. Follow the on screen
instructions to run the calibration code. Remember low
sensor values are light on and high values are light
off.  After the calibration, the CBC beeps, prints a
message and beeps again. This program can also be run in
the simulator.*/

int main()
{
     wait_for_light(0);
     // wait for light sub routine using port 0
     beep();
     cbc_display_clear();
     printf("Running Code\n");
     beep();
}
```

4.3 “大礼帽”传感器

- **性能**

最大探测距离：15mm

敏感波长：940-850nm

- **描述**

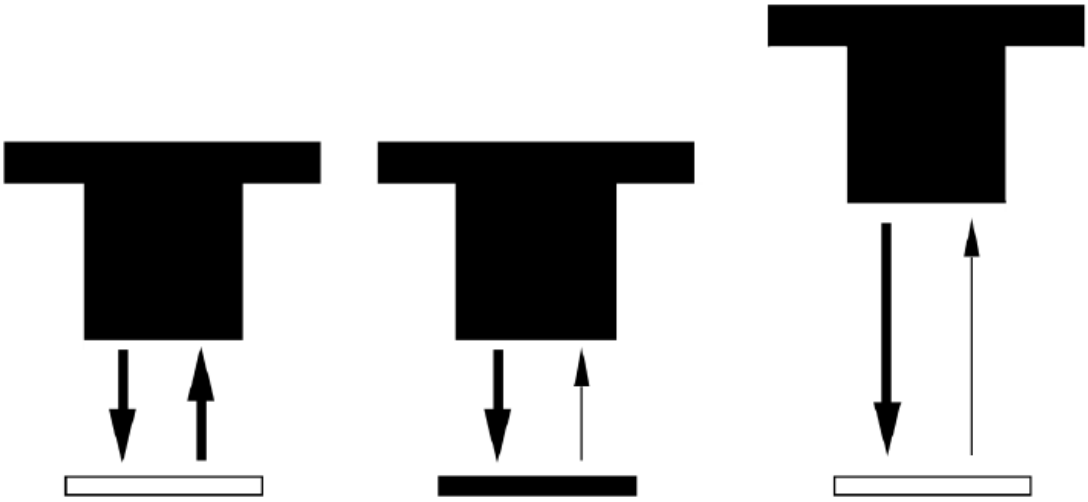
“大礼帽”传感器因其形状而得名。该传感器是一个真正的短距离反射传感器，其中包含有一个红外发射器和红外采集器。红外发射器发出红外光，红外采集器测量有多少红外光被反射回来。

- **使用**

该传感器有两种用途。首先是作为一个黑线检测器。黑色材料通常吸收红外光并且反射非常少量的红外光，白色的材料则几乎不吸收的红外光，反射回大多数红外光。如果这个传感器被安装在一个固定的高度面上就很容易从一个白色的表面区分出一条黑色的线。

第二用途是作为一种短距离传感器。请注意，因为有时近处的黑色物体看起来像白色的远方的物体。“大礼帽“传感器针对黑与白做好校准后才会工作得最佳。

红外光被反射回来的量取决于表面纹理，颜色和表面的距离。（见下图）



- **示例代码 1**

```
/*This program demonstrates how to follow a line with a top hat
sensor. This is for a robot with the left motor in port 0, right
motor in port 3 and a top hat sensor in port 0 and mounted at
the front of the robot. This program uses bang-bang control. You
will need to adjust the threshold value to work for you (here it
is 512, right in the middle)*/

int main(){
    while(1){                    //loop forever
        if(analog10(0)>=512){    //if the top hat sees light color
            mav(0,750);          //left motor fast
            mav(3,100);          //right motor slow
        }
        if(analog10(0)<512){     //if the top hat sees dark color
            mav(0,100);          //left motor slow
            mav(3,750);          //right motor fast
        }
    }
}
```

- **示例代码 2**

```
/*This program shows how to use a Top Hat sensor as a distance
sensor. This program drive motor 0 forward until the Top Hat
sensor in port 15 is triggered.  If the sensor becomes un
triggered, the motor will move forward again. You can change 512
to be the distance you want to detect*/

int main()
{
    while(1){                        //loop forever
        fd(0);                       //drive motor
        while(analog10(0)>512){      //if sensor detects an object
        ao();}                       //stop motor
    }
}
```

4.4 “小礼帽“传感器

- **性能**

最大探测距离：12mm

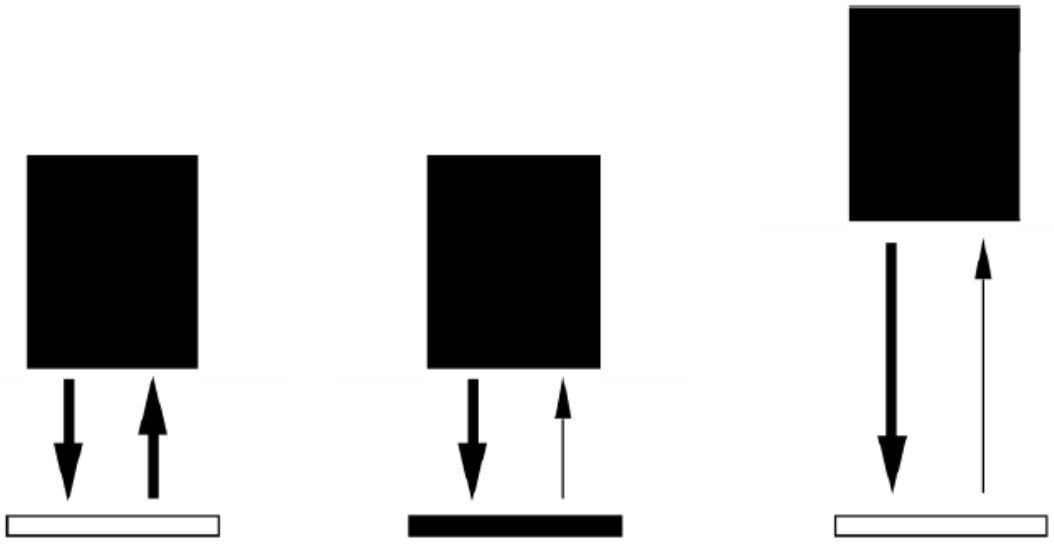
光敏感波长：940-850nm

- **描述**

“小礼帽”传感器因其兄弟“大礼帽“传感器的形状而得名。两个传感器工作原理是一样的。但是此传感器的探测距离更短，它同样包含有一个红外发射器和红外采集器。红外发射器发出红外光，红外采集器测量有多少光被反射回来。

- **使用**

- 该传感器有两种用途。首先是作为一个黑线检测器。黑色材料通常吸收红外光并且反射非常少量的红外光，白色的材料则几乎不吸收的红外光，反射回大多数红外光。如果这个传感器被安装在一个固定的高度面上就很容易从一个白色的表面区分出一条黑色的线。
- 第二用途是作为一种短距离传感器。请注意，因为有时近处的黑色物体看起来像白色的远方的物体。“大礼帽“传感器针对黑与白做好校准后才会工作得最佳。
- 红外光被反射回来的量取决于表面纹理，颜色和表面的距离。（见下图）



- **示例代码 1**

```
/*This program demonstrates how to follow a line with a top hat
sensor. This is for a robot with the left motor in port 0, right
motor in port 3 and a top hat sensor in port 0 and mounted at
the front of the robot. This program uses bang-bang control. You
will need to adjust the threshold value to work for you (here it
is 512, right in the middle)*/

int main(){
    while(1){                    //loop forever
        if(analog10(0)>=512){    //if the top hat sees light color
            mav(0,750);          //left motor fast
            mav(3,100);          //right motor slow
        }
        if(analog10(0)<512){     //if the top hat sees dark color
            mav(0,100);          //left motor slow
            mav(3,750);          //right motor fast
        }
    }
}
```

- **示例代码 2**

```
/*This program shows how to use a Top Hat sensor as a distance
sensor. This program drive motor 0 forward until the Top Hat
sensor in port 15 is triggered.  If the sensor becomes un
triggered, the motor will move forward again. You can change 512
to be the distance you want to detect*/

int main()
{
    while(1){                     //loop forever
        fd(0);                    //drive motor
        while(analog10(0)>512){//while sensor detects an object
        ao();}                    //stop motor
    }
}
```

4.5 ET 传感器

- **性能**

最大探测距离：80cm

光敏感波长：940-800nm

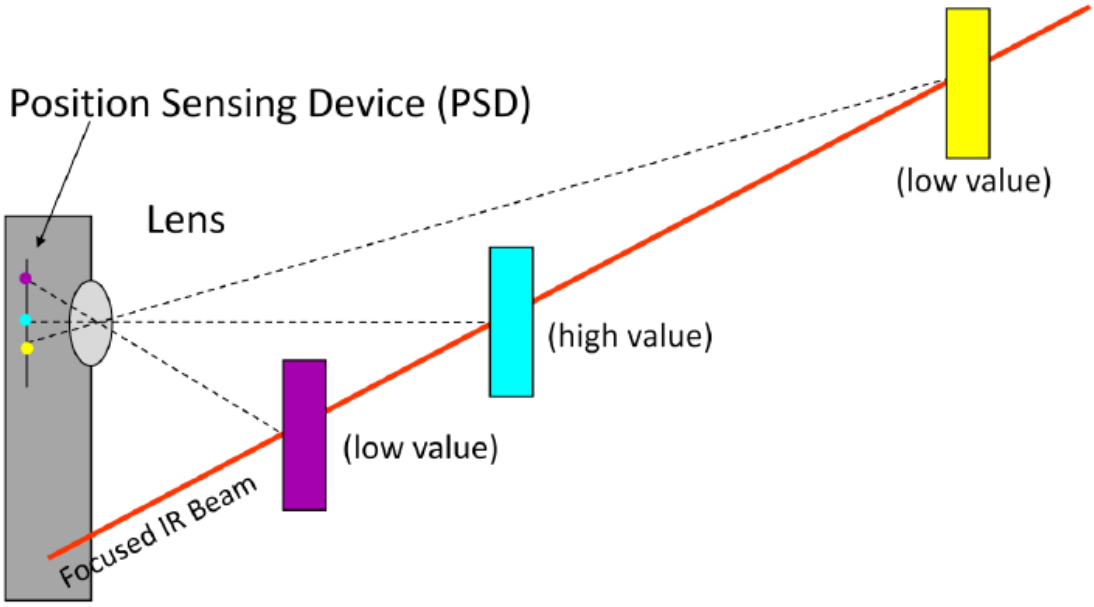
- **描述**

“ET”传感器因其形状像电影“外星人 ET”而得名。该传感器可以发出调制频率的红外光束，并测量被反射回来的红外光的入射角度。然后用三角测量法计算物体的距离。由于光线是调频的，此传感器不易因照明条件的变化而产生误差。

- **使用**

使用此传感器时，模拟端口必须设置为浮动端口类型，见附录中设置端口为浮动端口的方法！

该传感器是一种强大的中距离传感器。当它检测到一个在 5cm 处的物体时，该传感器读数为最高值。如果物体距离变近或变远，读数的值都会降低（见下图）。为了分辨物体的远近，解决办法是令所探测的物体不可能接近到距离传感器 5cm 范围内。


Position Sensing Device (PSD)
Lens
(low value)
(high value)
(low value)
Focused IR Beam

- **示例代码**

```
/*This program demonstrates how to use the ET for
sensing distances.  This program uses an ET in port 0
and motors in ports 0 and 3.  The robot drives forward
until it detects an object.  If the object gets
closer, the robot backs up, if the object gets further
away, the robot moves closer.  When using this type of
programming watch out for dead zones (where the sensor
values do nothing).*/

int main(){
    set_each_analog_state(1,0,0,0,0,0,0,0);
    //set port 0 to floating
    sleep(0.02);          //wait for state to change
    while(1){                       //loop forever
        if(analog10(0)>512){        //if too close
            bk(0);                  //back up
            bk(3);
        }
        if (analog10(0)<=512){      //if too far away
            fd(0);                  //get closer
            fd(3);
        }
    }
}
```

4.6 声纳传感器

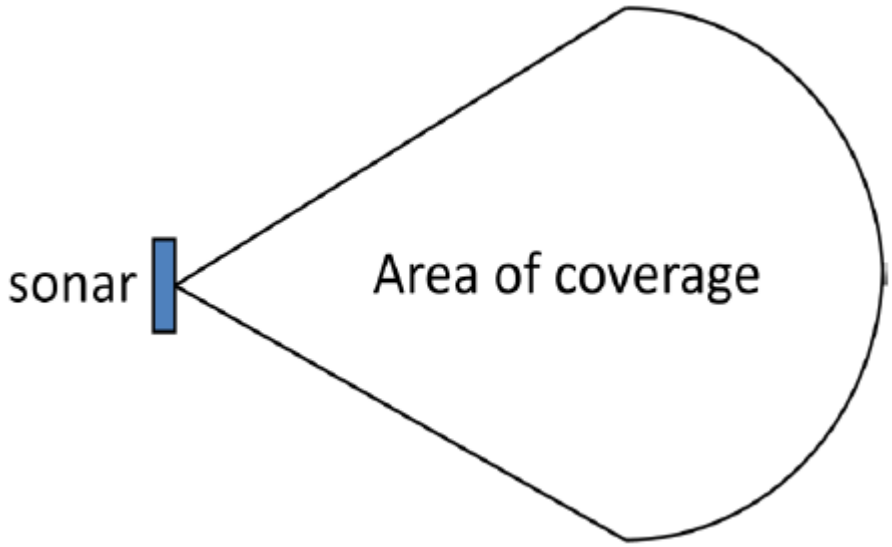
- **性能**

  探测范围：6-254 英寸

  刷新速率：20Hz(或每 50ms 一次)

- **描述**

声纳，SONAR 是 SOund Navigation And Ranging 的缩写。声纳传感器通过发送超声（ping）波并测量回声的时间来工作。测量到回声时间后，传感器通过就可以利用声音传播的速度计算障碍物的距离了。声纳发出的超声波是呈锥形传播的，所以声纳测量范围也是个辐射状的锥形（见下图）。由于声纳传感器是通过返回回声的时间测量的，所以它将返回测量区内最近物体的距离，因为最先到达的回声是由最近物体反射的。当调用 analog()函数时，返回值大致相当于以英寸为单位的距离。



- **使用**

使用此传感器时，模拟端口必须设置为浮动端口类型，见附录中设置端口为浮动端口的方法！

声纳传感器主要用于检测距离。由于声纳传感器用声音探测，一些对象对于声纳可能是不可见的，如 Botguy 玩偶，它会吸收声音而不反射回声，因此传感器不能发现它。声纳传感器最易探测的对象是很光滑坚硬的物体。一种常用的方

法是使用声纳传感器和 ET 传感器或“大礼帽“传感器区分硬和软的物体。当第一次使用声纳传感器，或当 CBC 启动时，传感器需要 350ms 的校准时间。在这段时间里，不应有物体接近传感器的 17 英寸范围内，否则读数将会被关闭。

- **示例代码**

```
/*This program demonstrates how to use the SONAR for
sensing distances.  You need to plug the SONAR sensor into
port 0.  This program beeps if anything changes from when
the program starts, i.e. if someone walks by. Remember -
when the SONAR is plugged in, or the CBC is turned on, the
SONAR needs about 17 inches of free space in front of it
for 350 ms while it calibrates.*/

int main(){
    int i=0;                    //comparator variable
    set_each_analog_state(1,0,0,0,0,0,0,0);
    //set port 0 to floating
    sleep(0.02);                //wait for state to change
    i=analog(0);                //take initial SONAR reading
    while(1){                   //loop forever
        if(analog(0)>(i+15)){     //if object moves away
            beep();
        }
        if(analog(0)<(i-15)){     //if object moves closer
            beep();
        }
    }
}
```

第五章 数字传感器

BotBall 的传感器有防插反设计 ,插入 CBC 的端口时只有一种方向才会让所有引脚都插入。

数字传感器的接头通常为两针接头。CBC 的端口上的“S”代表传感器传回的信号(SEN)，” V”代表电源正极(VCC)，” G”代表地（GND）。数字传感器的 2 根线一根连接 SEN 另一根连接 GND。

数字传感器既可以被插入到一个模拟端口又可以被插入到数字端口使用，如果数字传感器被插入模拟端口使用，那么返回值是 0 或者 255（而不是 1 或 0）。

5.1 KISS-C 数字传感器库函数

digital(<port#>)

如果连到端口（port#为端口号）的开关打开，返回 0；如果开关关闭，返回 1。数字端口号范围为 8-15。

analog(<port#>)

将数字传感器连接到模拟端口时使用这个函数，在数字端口用光了的情况下是很有必要的。如果连到端口的开关打开，返回 0；如果开关关闭，则返回 255。

5.2 槽传感器

- **性能**

敏感波长：940-850nm

- **描述**

该传感器是一个光学的槽传感器。在 U 型槽的一端是红外发射器，另一端是红外采集器。当传感器的光束被阻挡时，红外采集器不能接收到红外信号，该传感器则被触发。当使用这种传感器时要注意，因为某些对象可能并不会阻挡红外光。

- **使用**

该传感器有两种主要用途。首先是作为一个编码器。编码器用于测量驱动器的转动速度或位置，在驱动器的轴上连接一个带很多槽孔的从动轮，它和驱动器的轴同步转动。编码器利用槽传感器记录经过它的每个从动轮上的槽孔。从动轮的槽孔越多，则编码器的分辨率越高。当车轮转动时从动轮随之转动，编码器上的光线会不停地被遮挡，通过计算遮挡的次数，就可以测量转动速度或距离。

第二个主要的用途是作为限位检测器。如果你期待的运动到位后，零件阻挡了红外光线，即可读取传感器检测这种状态。

- **示例代码 1**

```
/*This program shows how to use a slot sensor as an
encoder. An encoder counts the times that a slot
passes in front of the sensor.  This program uses the
slot sensor in port 15 to count the number of times
triggered and prints it to the screen.*/

int main()
{
    int i=0;                    //counter variable
    while(1){                   //loop forever
        while(!digital(15)){}  //wait until empty
        while(digital(15)){}   //wait until triggered
        i++;                    //add 1 to the count
        printf("triggered %d times\n",i);    //print
    }
}
```

- **示例代码 2**

```
/*This program shows how to use a slot sensor as a
limit sensor. This program drive motor 0 forward until
the slot sensor in port 15 is triggered.  If the
sensor becomes un triggered, the motor will move
forward again.*/

int main()
{
    while(1){                       //loop forever
        fd(0);                      //drive motor
        while(!digital(15)){       //while triggered
            ao();}                  //stop motor
    }
}
```

5.3 大按钮传感器

- **性能**

平均寿命：100,000 周期

驱动力：160 ± 50 gf

- **描述**

触摸传感器是一个机械开关。按下开关后两个接触点形成电路回路。由于这种工作方式，它的状态只可能是开（1）或者是关（0）中的一种。

- **使用**

该传感器常用来检测机器人是否接触到障碍物，如为机器人设置一个保险杠。

- **示例代码**

```
/*This program shows how to use a touch sensor as a
bump sensor. This program drive motor 0 forward until
the touch sensor in port 15 is triggered.  If the
sensor becomes un triggered, the motor will move
forward again.*/

int main()
{
     while(1){                          //loop forever
          fd(0);                        //drive motor
          while(digital(15)){           //while triggered
               ao();}                   //stop motor
     }
}
```

5.4 杠杆开关传感器

- **性能**

平均寿命：50,000 周期

驱动力：5Gms. max.

- **描述**

杠杆开关传感器是一个机械开关。按下开关后两个接触点形成电路回路。由于这种工作方式，它的状态只可能是开（1）或者是关（0）中的一种。

- **使用**

该传感器常用来检测机器人是否接触到障碍物，如为机器人设置一个保险杠。

- **示例代码**

```
/*This program shows how to use a touch sensor as a
bump sensor. This program drive motor 0 forward until
the touch sensor in port 15 is triggered.  If the
sensor becomes un triggered, the motor will move
forward again.*/

int main()
{
    while(1){                       //loop forever
        fd(0);                      //drive motor
        while(digital(15)){         //while triggered
            ao();}                  //stop motor
    }
}
```

5.5 小按钮传感器

- **性能**

平均寿命：100,000 周期

驱动力：160 ± 50 gf

- **描述**

小按钮传感器是一个机械开关。按下开关后两个接触点形成电路回路。由于这种工作方式，它的状态只可能是开（1）或者是关（0）中的一种。

- **使用**

该传感器常用来检测机器人是否接触到障碍物，如为机器人设置一个保险

- **示例代码**

```
/*This program shows how to use a touch sensor as a
bump sensor. This program drive motor 0 forward until
the touch sensor in port 15 is triggered.  If the
sensor becomes un triggered, the motor will move
forward again.*/

int main()
{
    while(1){                       //loop forever
        fd(0);                      //drive motor
        while(digital(15)){         //while triggered
            ao();}                  //stop motor
    }
}
```

第六章 加速计

- **性能**

  范围：每个轴(X,Y,Z)加速度范围为±2G

- **描述**

  CBC 中内置有一个三轴加速度计。它不断地返回 X，Y 和 Z 轴的加速度。加速度是速度随时间的变化量。所以速度不变时加速度为 0，1G 的加速度读数大约为 50。

- **使用**

  加速计通常用来感知 CBC 的方向。

- **示例代码**

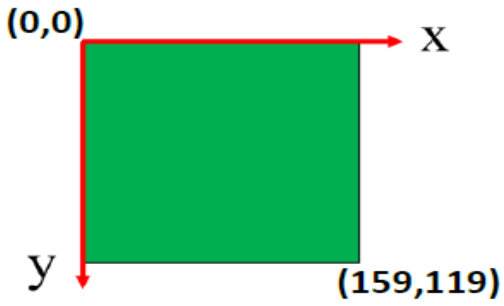
```
/*This program shows how to use the accelerometer.  The
accelerometer is built in, so you just need to load the
code on to your CBC.  The CBC will beep and print a message
if turned upside down.*/

int main(){
    while(1){                   //loop forever
        if(accel_z()<-50){      //if CBC reads -1G...
            beep();             //usually only if upside down
            printf("HELP!  I'm Upside Down!\n");
            sleep(0.5);
        }
    }
}
```

第七章 摄像头

7.1 关于视觉追踪

CBC v2 中已内建了一个颜色视觉跟踪系统。一个 USB 摄像头以 5 帧/秒的速度为 CBC v2 提供图像。CBC v2 随后对图像进行实时处理。CBC v2 在四个颜色通道上分别将图像分割为 10 个色块。每个色块是一组被认为是同一颜色通道的像素（一组颜色相近的像素）。当每一帧的图像都被自动处理为色块，并且信息以变量的形式存储，你可以通过访问这些变量了解机器人所处的环境。在 CBC V2 中可以对颜色通道进行设定。摄像头图像大小为 160 * 120 像素。它的坐标如图所示，左上角为坐标(0,0)和右下角为坐标(159，119)。



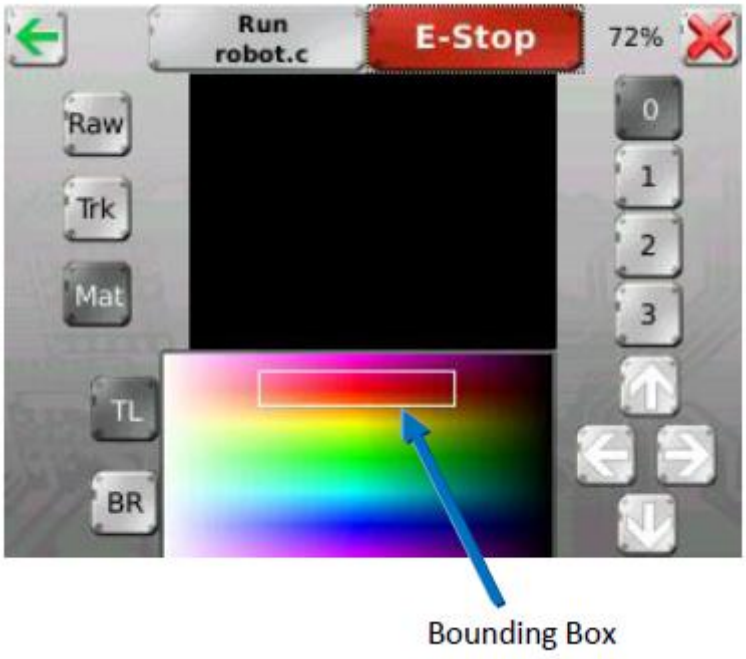
7.2 设定 CBC V2 的颜色通道

- 关闭 CBC V2，把 USB 摄像头插入 CBC V2 后面的 USB 端口中，如下图所示：

- 打开 CBC V2，开机到主菜单。按“Vision“按钮。

- 按下“Tracking“按钮

- 如图打开摄像头设置界面

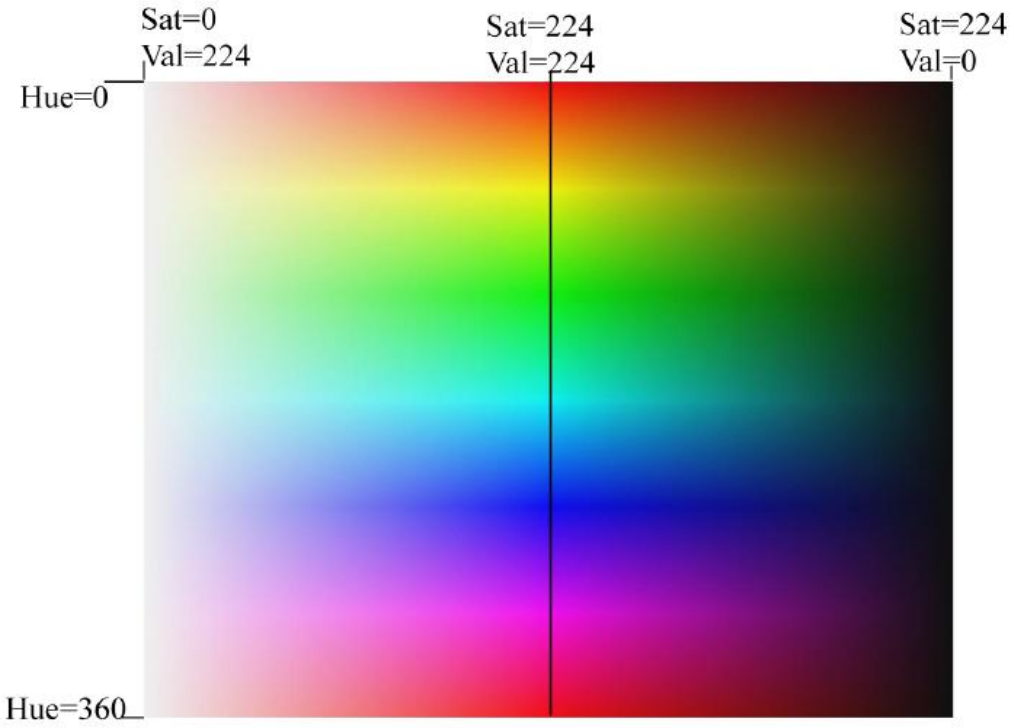


在右上方有编号 0 - 3 的按钮为四个通道。在左上角有三个按钮标记着 Raw（原始图像）、Trk（跟踪）和 Mat（匹配），它们分别代表摄像头的原始图像、跟踪后的摄像头图像、匹配后的摄像头图像。原始图像就是摄像机目前拍摄到的图像。跟踪图像将落入颜色通道的像素显示为白色。匹配图像

用包括了整个色块的最小边框显示，它包括一个绿色的边界框和一个绿色加号，绿色加号代表了色块的质心位置。

设置 CBC V2 的颜色通道的目的是改变代表颜色通道的边框的区域和大小，来令待跟踪对象的颜色尽可能多地进入颜色通道，而非跟踪对象尽可能少地进入颜色通道。这个边框是由在屏幕底部右下方的四个箭头和在底部左下角的“TL“和”BR“按钮控制。TL（top left）和 BR(bottom right)按钮选择你移动的边界框的角落是左上角还是右下角。由于边界框的属性，右下角处于 HSV 颜色选择平面的右边，左上角是在其左边。

HSV 颜色选择平面(如下所示)是摄像头可以看到的所有颜色的表示。H 代表了色相值(Hue) ,它描述了颜色(如红、黄、蓝)。S 为饱和度(Saturation),V 为亮度（Value），它们共同描述了颜色的暗或明亮(黑或白的程度)。注意,饱和度的值为 224 且亮度的值为 224 的颜色是摄像头可以看到的最明亮(或最纯)的颜色。



默认情况下，颜色通道被设置为四种最常见(和容易被跟踪)的颜色范围。通道 0 是红色，通道 1 是黄色，通道 2 是绿色，通道 3 是蓝色。你可以根据实际应用调整颜色模型的边界，这样做时默认值是一个很好的起点。

当根据需要调整颜色通道时，首先打开 Mat 匹配图像画面，并令你整个要跟踪的对象都处于边界框里。然后逐步减少边界框的大小，以去除被误跟踪的环境中其它物体的数量。注意,为了令环境中其它物体被误跟踪的数量较少，你可能不能将你要跟踪的整个对象都纳入边界框中（由于非均匀的照明条件）。

7.3 CBC V2 视觉跟踪库函数

下列是最常用的函数，请参阅附录查看完整的列表。

track_update()

令 CBC V2 处理当前帧的图像，并令处理结果可被程序使用。在调用其它任何视觉跟踪库函数前，总是先调用这个函数。

track_count(<channel>)

返回一个通道上的色块数。使用前先调用 track_update()。

track_x(<channel>,<number>)

返回通道上的某个色块的 X 坐标。色块号(number)从 0 到 9 排列，0 是最大的色块的编号。使用前先调用 track_update()。

track_y(<channel>,<number>)

返回通道上的某个色块的 X 坐标。色块号(number)从 0 到 9 排列，0 是最大的色块的编号。使用前先调用 track_update()。

7.4 与伺服驱动器配合的颜色跟踪程序示例

此示例程序演示了在 CBC V2 中对摄像头和伺服驱动器的使用。这个程序跟踪颜色通道 0 上的物体 ,并且令一个连接到伺服驱动器上的乐高零件总是指向该对象。

- **设置**

把摄像头连接到 CBC V2。你需要首先设置颜色通道 0 上颜色模型，以跟踪你将要使用的颜色物体（越亮越好）。然后，设置伺服驱动器令指针零件（不要和你待跟踪的物体颜色一样）正好可以指向摄像头的两边视角边缘，而伺服驱动器在中点(1024)时指针指在摄像头视角的中心。最后，编写并下载程序到 CBC V2。

- **代码**

```
/* This program points a servo (that is plugged into port 0 and
centered on the camera's field of vision) towards an object that
fits into the color model defined for channel 0*/

int main()
{
    int offset, x, y;
    enable_servos();
    track_update(); // get most recent camera image and process it
    while(black_button() == 0) {
        x = track_x(0,0);  // get image x data
        y = track_y(0,0);  //    and y data
        if(track_count(0) > 0) {  // there is a blob
            printf("Blob is at (%d,%d)\n",x,y);
            offset=5*(x-80); //amount to deviate servo from center
            set_servo_position(0,2014+offset);
        }
        else {
            printf("No object in sight\n");
        }
        sleep(0.2); // don't rush print statement update
        track_update();  // get new image data before repeating
    }
    disable_servos();
    printf("All done\n");
}
```

7.5 编写不用伺服驱动器的颜色跟踪程序示例

这个示例演示了不使用伺服驱动器的颜色跟踪程序。它跟踪颜色通道 0 上的一个物体，然后点亮电机驱动端口上的小灯，以指示物体位置。如果物体位于 2 号驱动器端口之前，端口 2 的小灯变成蓝色；如果对象移动到 1 号驱动器端口前，端口 2 的灯灭，端口 1 的灯点亮。

- **设置**

把摄像头连接到 CBC V2。你需要首先设置颜色通道 0 上颜色模型，以跟踪你将要使用的颜色物体（越亮越好，但不要是蓝色的）。令摄像头朝向 CBC V2 前面的地面，但不要将 CBC V2 纳入镜头内。令镜头的视野和 CBC V2 的朝向对齐。最后，编写并下载程序到 CBC V2。

- 程序

```
/*For this program, point the camera so it is looking in front of the
motor ports on the CBC v2.  This program turns on the motor light that
corresponds to the position of an object on channel 0.  i.e. if an
object on channel 0 is in front of motor port 2 the motor 2 light will
come on. */

int main()
{
     int x, xMax = 160;
     while(black_button()==0){
          track_update();  //get most recent camera image and process it
          x = track_x(0,0);    // get image x data
          if(track_count(0)>0){    // there is a blob
               printf("Blob x position is %d\n",x);
               if(x >= 0 && x < (xMax/4)){    // if object is by motor 3
                    fd(3);}
               else if(x >= (xMax/4) && x < (xMax/2)){ // object by 2
                    fd(2);}
               else if(x >= (xMax/2) && x < ((3*xMax)/4)) // object by 1
                    fd(1);}
               else if(x >= ((3*xMax)/4) && x <= xMax){    // object by 0
                    fd(0);}
          }
          sleep(0.1);
     }
ao();
}
```

第八章 故障排除

如果在任何时候你需要额外的帮助以排除故障，或遇到有你不能解决的问题，请在美国中部时间上午 9 点到下午 5 点间，拨打技术支持电话(405)579 - 4609，或发电子邮件到 support@kipr.org。

问题	解决方法
当启动 CBC 时，所有电机端口红灯亮	需要重新烧入 CBC 固件，请参看 CBC 手册。
CBC 中摄像头上的图像死掉或乱掉了。	当 CBC 与摄像头失去连接时会发生这

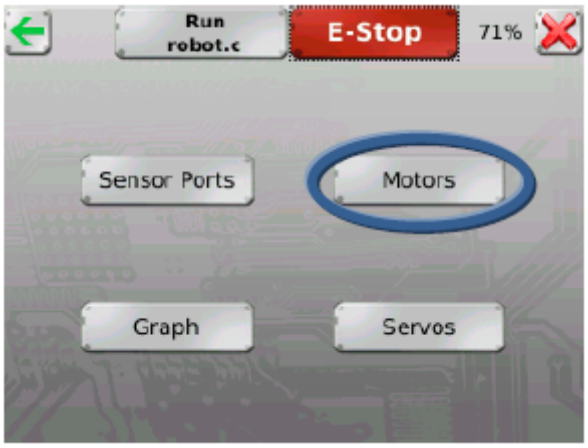
	种情况。关掉 CBC，拔下并重新插上摄像头，再次开启 CBC。
模拟传感器的数值只有很小的变化量。	确认你的传感器不是浮动端口型的。如果确实不是，需要启动上拉电阻。参加附录中如何设置上拉电阻型和浮动型端口。
模拟传感器只能返回 1 或 0，没有其它数值。	确认你的模拟传感器确实插入了模拟端口中（0 到 7 号）。
CBC 控制器曾经在电脑上被认做 COM 口，但现在再也不认了。	拔下 USB 线，重启 KISS-C 环境，并再次插入 USB 线。
电机转动方向是错误的	把电机插头反过来插。

第九章 附录

9.1 CBC 中内建的连续转动驱动器测试

CBC 已建成一个无需编写代码即可对驱动器和传感器进行测试的页面。从主菜单中点击 Sensors/Motors 按钮。

下面选择 Motors：



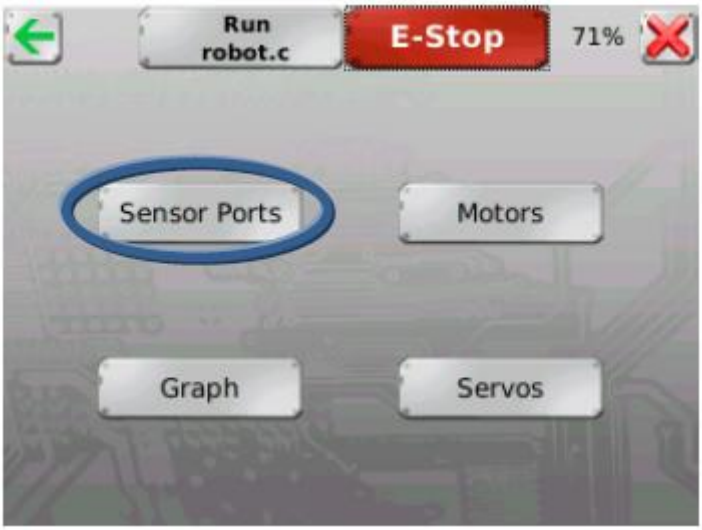
下面选择 Test：

现在你将看到电机测试页面。从这里你可以使用在右上角的箭头改变驱动器端口，默认情况下的端口为 0 号。之后，可以通过选择单选按钮，选择待测试项目为 Power(功率)、Velocity(速度)和 Position（位置）。然后你输入所需的功率、速度、和/或位置值。点击白色的输入框，将跳出一个小键盘另你输入数字。下方的文字显示了当前位置。在左下角有一个 Clear 按钮，用来清除驱动器的位置。在右下角有一个 Go 按钮。按这个按钮来开始你的测试。

9.2 内建的驱动器位置显示

有两个地方来可以查看当前驱动器的位置。第一个是在驱动器测试页面(见上节介绍)，第二个是在传感器端口屏幕。从主菜单选择 Sensors/Motors 页面。

接下来选择 Sensor Ports：



接下来你将看到如下屏幕：

屏幕的底部显示了当前的电机位置信息和目前的电机功率。

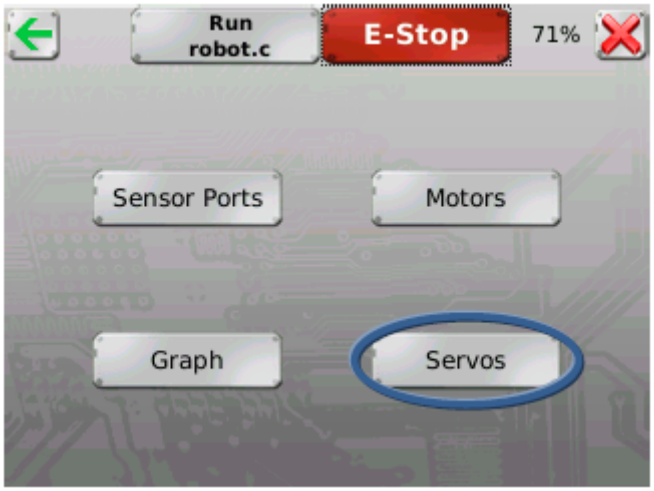
9.3 检查驱动器极性

CBC 自身没有办法判断电机正在驱动机器人前进还是后退。你必须确保电机连接线以正确方向插入端口中。有很多方法可以做到这一点，你可以去驱动器测试页面(见 9.1)，给驱动器供电，，确保它转向正确。你也可以使用驱动器位置显示(见 9.2)，用手转动驱动器来看看位置增加(前进)或减少(后退)。最后，如果你用手较快地转动电机，CBC 上驱动器端口的 LED 灯将被点亮，蓝色代表前进，红色代表向后移动。如果你发现驱动器的极性反了，只要拔下连接线并反向重新插好，那么现在驱动器的转向就是正确方向了。

9.4 内建的伺服驱动器测试

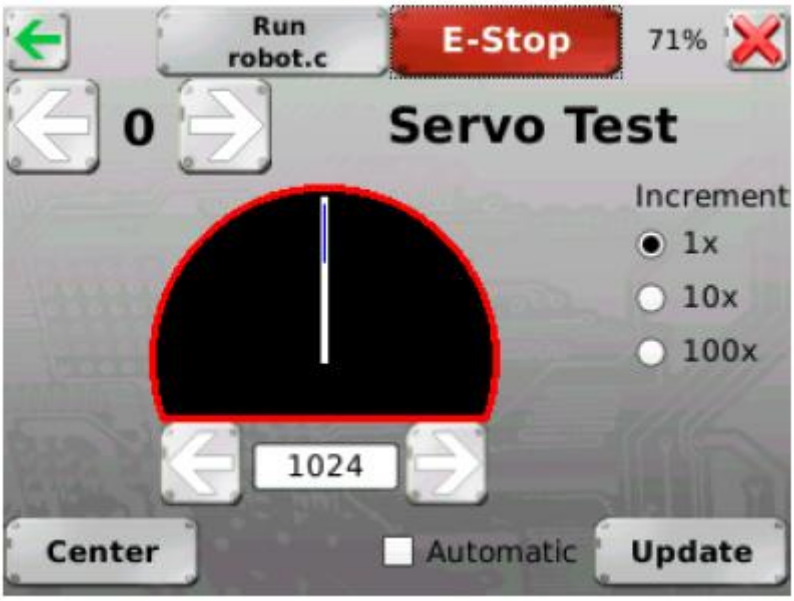
CBC 已建成一个无需编写代码的伺服驱动器测试页面来测试。从主菜单选择 Sensors/Motors 页面。

下面选择 Servos 按钮：



现在你将看到伺服驱动器测试页面。从这里你可以用右上角的箭头改变伺服驱动器的端口，默认情况下它是端口 0。你可以使用屏幕下方的中心左边和右边

的箭头设置伺服驱动器的位置。设置完成后，点击右下方的更新按钮，可以移动伺服驱动器到所设置的位置。你还可以选中“Automatic”选框，然后每次改变位置后，不用点击 Update 按钮，伺服驱动器就会自动运动到该位置。最后，左下角的按钮 Center 设置了位置 1024。



当你离开伺服驱动器测试页面后，所有的伺服驱动器都将回到 disable 状态。

9.5 找到伺服驱动器的运动范围

每个伺服驱动器有不同范围的值。虽然它们都可以旋转大约 180 度，但实际值方面是有差别的。为了找到你使用的伺服驱动器的范围，首先进入伺服驱动器测试页面，并设置伺服驱动器的位置为 0。然后，增加位置值直到伺服驱动器不再超过限位(over driven，此时电机不移动)且开始移动。这时的值就是你的伺服驱动器的最低值。

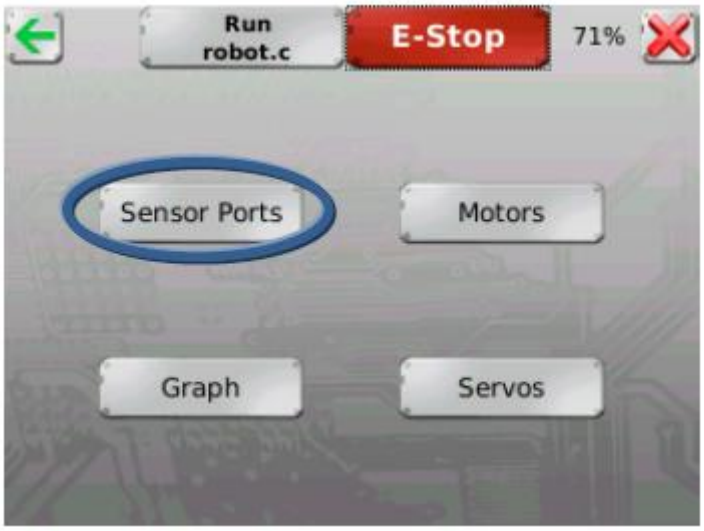
此后，从最高值开始重复这个过程，减少值直到伺服驱动器开始移动。这就是你的伺服驱动器的最高值。

你可以计算你的伺服驱动器的 tic 和转动角度之间的关系，计算方法为用转动角度除以最高值减最低值。例如，如果您的伺服驱动器旋转 180 度，最小值 100，最大值 1900，那么结果为：1（度/tic）=(180 /[1900 - 100])，也即 0.1（度/tic）。这个结果可以用在计算公式中成比例使用。例如，你需要把伺服驱动器从 0 度调到 45 度。根据此例子，你将发送的伺服驱动器位置值为：(100 +[45 / 0. 1])=550。发送 550，伺服驱动器就会停在 45 度位置。记住，伺服驱动器的精度为±1 度，所以更小幅度地调整伺服驱动器是不可能的。

9.6 手动禁用模拟端口的上拉电阻

默认情况下所有的模拟端口的上拉电阻都是被启用。从主菜单选择 Sensors/Motors 页面。注意，当 CBC V2 在重新启动时，上拉电阻被禁用。

选择 Sensors Ports 按钮：



下面你将看到如下屏幕：

当你触摸模拟端口旁边的单选框时，框中显示“x”，此时模拟端口就被设置成为了一个浮动端口，如下所示：

在前面的图中,模拟端口 0 和 1 现在设置为浮动端口 ,不再启动 15 k 欧姆的上拉电阻。

9.7 在你的程序中禁用模拟端口的上拉电阻

默认情况下所有的模拟端口上拉电阻都是被启用。使用以下内部功能禁用或启用上拉电阻。

set_each_analog_state(int a0, int a1, int a2, int a3, int a4, int a5, int a6, int a7);

每个整数对应一个模拟端口，参数 a0 对应 0 号，a1 对应 1 号……设置该端口对应的参数为 1 禁用端口；设置参数为 0 启用端口。看下面的例子。

```
// disable pull up resistor on analog port 4

int main()
{
   set_each_analog_state(0,0,0,0,1,0,0,0);
   sleep(0.02);
}
```

在上面的例子中，4 号端口的上拉电阻被禁用。注意除了禁用 4 号端口的上拉电阻，你还同时启用了其它所有端口的上拉电阻。所以多次使用此函数时，需要注意这一点。在调用此函数后，你还应该在添加一个短的睡眠延时，让状态得到改变，如上例中所示。

9.8 KISS-C 中 CBC V2 的库函数（按首字母排序）

- a_button [Category: Sensors]

  Format: int a_button();

  Reads the value (0 or 1) of the A button.

- alloff [Category: Motors]

  Format: void alloff();

  Turns off all motors. ao is a short form for alloff.

- analog [Category: Sensors]

  Format: int analog(int p);

  Returns the value of the sensor installed at the port numbered p. The result is an integer between 0 and 255. The function can be used with analog ports 0 through 7.

- analog10 [Category: Sensors]

  Format: int analog10(int p);

  10-bit version of the analog function. The returned value is in the range 0 to 1023 rather than 0 to 255.

- ao [Category: Motors]

  Format: void ao();

  Turns off all motors.

- atan [Category: Math]

Format: float atan(float angle);

Returns the arc tangent of the angle. Angle is specified in radians; the result is in radians.

- b_button [Category: Sensors]

  Format: int b_button();

  Reads the value (0 or 1) of the B button.

- beep [Category: Output]

  Format: void beep();

  Produces a tone. Returns when the tone is finished.

- bk [Category: Motors]

  Format: void bk(int m);

  Turns motor m on full speed in the backward direction.

  Example:

  bk(1);

- black_button [Category: Sensors]

  Format: int black_button();

  Reads the value (0 or 1) of the Black button on the CBC (or a period on the simulator).

- block_motor_done [Category: Motors]

  Format: void block_motor_done(int m);

  Function does not return until specified motor completes any executing

speed or position control moves.

Example:

mrp(0,500,20000L);

block_motor_done(1);

- bmd [Category: Motors]

  Format: void bmd(int m);

  Function does not return until specified motor completes any executing speed or position control moves.

  Example:

  mrp(0,500,20000L);

  bmd(1);

- cbc_display_clear [Category: Output]

  Format: void cbc_display_clear();

  Clear the CBC display.

- cbc_printf [Category: Output]

  Format: void cbc_printf(int col, int row, char s[], . . .);

  Perform a standard printf starting at screen location col, row.

- clear_motor_position_counter [Category: Motors]

  Format: void clear_motor_position_counter(int motor_nbr);

  Reset the position counter for the motor specified to 0.

- cos [Category: Math]

Format: float cos(float angle);

Returns cosine of angle. Angle is specified in radians; result is in radians.

- defer [Category: Processes]

  Format: void defer();

  Makes a process swap out immediately after the function is called. Useful if a process knows that it will not need to do any work until the next time around the scheduler loop. defer() is implemented as a C built-in function.

- digital [Category: Sensors]

  Format: int digital(int p);

  Returns the value of the sensor in sensor port p, as a true/false value (1 for true and 0 for false). Sensors are expected to be active low, meaning that they are valued at zero volts in the active, or true, state. Thus the library function returns the inverse of the actual reading from the digital hardware: if the reading is zero volts or logic zero, the digital() function will return true. Valid for digital ports 8-15.

- disable_servos [Category: Servos]

  Format: void disable_servos();

  Disables the servo motor ports (powers down all servo motors).

- down_button [Category: Sensors]

  Format: int down_button();

  Reads the value (0 or 1) of the move down button.

- enable_servos [Category: Servos]

  Format: void enable_servos();

  Enables all servo motor ports.

- exp10 [Category: Math]

  Format: float exp10(float num);

  Returns 10 to the num power.

- exp [Category: Math]

  Format: float exp(float num);

  Returns e to the num power.

- fd [Category: Motors]

  Format: void fd(int m);

  Turns motor m on full in the forward direction.

  Example:

  fd(3);

- freeze [Category: Motors]

  Format: void freeze(int m);

  Freezes motor m (prevents continued motor rotation, in contrast to off, which allows the motor to "coast").

- get_motor_done [Category: Motors]

  Format: int get_motor_done(int m);

  Returns whether the motor has finished a move with specified position.

- get_motor_position_counter [Category: Motors]

  Format: int get_motor_position_counter(int m);

  Returns the current motor position value for motor m (a value which is continually being updated for each motor using back EMF; a typical discrimination for a given motor is on the order of 1100 position "ticks" per rotation)

- get_servo_position [Category: Servos]

  Format: int get_servo_position(int srv);

  Returns the position value of the servo in port srv. The value is in the range 0 to 2047. There are 4 servo ports (0, 1, 2, 3).

- kill_process [Category: Processes]

  Format: void kill_process(int pid);

  The kill_process function is used to destroy processes. Processes are destroyed by passing their process ID number to kill_process. If the return value is 0, then the process was destroyed. If the return value is 1, then the process was not found. The following code shows the main process creating a check_sensor process, and then destroying it one second later:

```
int main(){
int pid;
pid = start_process(check_sensor);
sleep(1.0);
```

kill_process(pid);}

- kissSimEnablePause [Category: Simulator]

  Format: void kissSimEnablePause();

  Will pause the simulation if the space bar is pressed when this is called.

- kissSimPause [Category: Simulator]

  Format: void kissSimPause();

  Will pause the simulation when this is called. Press the space bar to resume.

- left_button [Category: Sensors]

  Format: int left_button();

  Reads the value (0 or 1) of the move left button.

- log10 [Category: Math]

  Format: float log10(float num);

  Returns the logarithm of num to the base 10.

- log [Category: Math]

  Format: float log(float num);

  Returns the natural logarithm of num.

- mav [Category: Motors]

  Format: void mav(int m, int vel);

  This function is the same as move_at_velocity

- motor [Category: Motors]

  Format: void motor(int m, int p);

  Turns on motor m at scaled PWM duty cycle percentage p. Power levels range from 100 for full on forward to -100 for full on backward.

- move_at_velocity [Category: Motors]

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

  Moves motor m at velocity vel indefinitely. The velocity range is -1000 to 1000 ticks per second.

- move_relative_position [Category: Motors]

  Format: void move_relative_position(int m, int speed, int pos);

  Moves motor m at velocity vel from its current position curr_pos to curr_pos + pos. The speed range is 0 to 1000 ticks per second.

  Example:

  move_relative_position(1,275,-1100L);

- move_to_position [Category: Motors]

  Format: void move_to_position(int m, int speed, int pos);

  Moves motor m at velocity vel from its current position curr_pos to pos. The speed range is 0 to 1000. Note that if the motor is already at pos, the motor doesn't move.

- mrp [Category: Motors]

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

This function is the same as move_relative_position.

- mtp [Category: Motors]

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

  This function is the same as move_to_position.

- msleep [Category: Time]

  Format: void msleep(int msec);

  Waits for an amount of time equal to or greater than msec milliseconds.

Example:

```
/*wait for 1.5 seconds */
msleep(1500);
```

- off [Category: Motors]

  Format: void off(int m);

  Turns off motor m.

  Example:

```
off(1);
```

- power_level [Category: Sensor]

  Format: float power_level();

  Returns the current power level in volts.

- printf [Category: Output]

  Format: void printf(char s[], . . .);

  Prints the contents of the string referenced by s to the cursor position on

the screen.

- r_button [Category: Sensors]

  Format: int r_button();

  Reads the value (0 or 1) of the R (shoulder) button.

- random [Category: Math]

  Format: int random(int m);

  Returns a random integer between 0 and some very large number.

- right_button [Category: Sensors]

  Format: int right_button();

  Reads the value (0 or 1) of the move right button.

- run_for [Category: Processes]

  Format: void run_for(float sec, void <function_name>);

  This function takes a function and runs it for a certain amount of time in seconds. run_for will return within 1 second of your function exiting, if it exits before the specified time. The variable sec denotes how many seconds to run the given function.

- seconds [Category: Time]

  Format: float seconds();

  Returns the count of system time in seconds, as a floating point number. Resolution is one millisecond.

- set_analog_floats [Category: Sensors]

Format: void set_analog_floats(int mask);

This function uses a number between 0 and 255 to set which port are to be set floating.

- set_each_analog_state [Category: Sensors]

Format: void set_each_analog_state(int a0, int a1, int a2, int a3, int a4, int a5, int a6, int a7);

This function is used to set weather or not the analog ports are set to floating points or to pullup resistors. Passing a 1 sets the corresponding port to floating. Please note that all sensor ports are set to non-floating when the CBC is rebooted or when a program exits.

- set_pid_gains [Category: Motors]

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

This function is used to adjust the weights of the PID control for the motors. The p, i and d parameters are the numerators for the p, i and d coefficients. The pd, id and dd parameters are their respective denominators. Thus all of the parameters are integers, but the actual coefficients can be floats. If a motor is jerky, the p and d terms should be reduced in size. If a motor lags far behind, they should be increased. The default values are 30,0,-30,70,1,51.

- set_servo_position [Category: Servos]

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

Sets the position value of the servo in port srv. The value of pos must be in

the range 0 to 2047. There are 4 servo ports (0, 1, 2, 3).

- setpwm [Category: Motors]

  Format: int setpwm(int m, int dutycycle);

  Runs motor m at duty cycle dutycycle (values -100 to 100)

- sin [Category: Math]

  Format: float sin(float angle);

  Returns the sine of angle. angle is specified in radians; result is in radians.

- sleep [Category: Time]

  Format: void sleep(float sec);

  Waits for an amount of time equal to or slightly greater than sec seconds. sec is a float.

  Example:

  /*wait for 2 seconds */

  sleep(2.0);

- sonar [Category: Sensors]

  Format: int sonar();

  Returns the approximate distance in mm.

- sqrt [Category: Math]

  Format: float sqrt(float num);

  Returns the square root of num.

- start_process [Category: Processes]

  Format: int start_process(<function name>);

  The start_process function is used to start a process, which then runs in parallel with other active processes. The system keeps track of each running process by assigning a process ID number to it. start_process returns the process ID number for each process it starts. The process runs until it finishes or until it is terminated by kill_process.

  The following code shows the main process creating a check_sensor() process, and then destroying it one second later:

```
int main(){
    int pid;
    pid=start_process(check_sensor());
    sleep(1.0);
    kill_process(pid);
}
```

- tan [Category: Math]

  Format: float tan(float angle);

  Returns the tangent of angle. angle is specified in radians; result is in radians.

- up_button [Category: Sensors]

  Format: int up_button();

Reads the value (0 or 1) of the move up button.

9.9 KISS-C 中 CBC V2 的视觉跟踪库函数

- track_is_new_data_available [Category: Vision API]

  Format: int track_is_new_data_available();

  Returns 1 if new data is available since the last call of track_update(), 0 if no new data is available.

- track_update [Category: Vision API]

  Format: void track_update();

  Processes tracking data for a new frame and makes it available for retrieval by the track_property() calls bellow.

- track_get_frame [Category: Vision API]

  Format: int track_get_frame();

  Returns the frame number used to generate the tracking data.

- track_count [Category: Vision API]

  Format: int track_count(int ch);

  Returns the number of blobs available for the channel ch, which is a color channel numbered 0 through 3.

- track_size [Category: Vision API]

  Format: int track_size(int ch, int i);

  Returns the size of blob from channel ch (range 0-3), index i (range 0 to

track_count(ch)-1) in pixels.

- track_x [Category: Vision API]

  Format: int track_x(int ch, int i);

  Returns the pixel x coordinate of the centroid for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_y [Category: Vision API]

  Format: int track_y(int ch, int i);

  Returns the pixel y coordinate of the centroid for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_confidence [Category: Vision API]

  Format: int track_confidence(int ch, int i);

  Returns the confidence for seeing the blob as a percentage of the blob pixel area/bounding box area (range 0-100, low numbers bad, high numbers good) for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_bbox_left [Category: Vision API]

  Format: int track_bbox_left(int ch, int i);

  Returns the pixel x coordinate of the leftmost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_bbox_right [Category: Vision API]

  Format: int track_bbox_right(int ch, int i);

  Returns the pixel x coordinate of the rightmost pixel for the blob from

channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_bbox_top [Category: Vision API]

  Format: int track_bbox_top(int ch, int i);

  Returns the pixel y coordinate of the topmost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_bbox_bottom [Category: Vision API]

  Format: int track_bbox_bottom(int ch, int i);

  Returns the pixel y coordinate of the bottommost pixel for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_bbox_width [Category: Vision API]

  Format: int track_bbox_width(int ch, int i);

  Returns the pixel x width of the bounding box for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1). This is equivalent to track_bbox_right - track_bbox_left.

- track_bbox_height [Category: Vision API]

  Format: int track_bbox_height(int ch, int i);

  Returns the pixel y height of the bounding box for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1). This is equivalent to track_bbox_bottom - track_bbox_top.

- track_angle [Category: Vision API]

  Format: int track_angle(int ch, int i);

Returns the angle in radians of the major axis for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1). Zero is horizontal and when the left end is higher than the right end the angle will be positive. The range is -PI/2 to +PI/2.

- track_major_axis [Category: Vision API]

  Format: int track_major_axis(int ch, int i);

  Returns the length in pixels of the major axis of the bounding ellipse for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).

- track_minor_axis [Category: Vision API]

  Format: int track_minor_axis(int ch, int i);

  Returns the length in pixels of the minor axis of the bounding ellipse for the blob from channel ch (range 0-3), index i (range 0 to track_count(ch)-1).