CBC v2 Manual

Version: 1.1

KIPR makes no warranty for the use of its products and assumes no responsibility for any errors which may appear in this document nor does it make a commitment to update the information contained herein. KIPR products are not intended for use in medical, life saving or life sustaining applications. KIPR retains the right to make changes to these specifications at any time, without notice.

The KISS Institute is a 501c3 nonprofit organization. Our mission is to improve the public's understanding of science, technology, engineering, and math; develop the skills, character, and aspirations of students; and contribute to the enrichment of our school systems, communities, and the nation.

Contents

1. CBC v2

About the CBC v2

The CBC v2 is a Linux-based robot controller designed by the KISS Institute for Practical Robotics for use with the KISS-C programming environment.

Featuring significant hardware and usability improvements over its predecessor, the CBC v2 is both a beginner-friendly choice for newcomers to robotics, and a powerful, feature-rich device that will appeal to experts.

CBC v2 Basic Features

- GNU/Linux based operating system
- Open-source robot control software
- Integrated color vision system
- 350MHz Freescale ARM9 processor
- Integrated battery and charge system
- Speaker & microphone
- 320 x 240 color touch screen

Input and Output

1 - 3 axis 10-bit accelerometer (software selectable 2/4/8g)
8 - digital I/O ports (hardware selectable 3.3V or 5V)
8 - 3.3V (5V tolerant) 10-bit analog input ports
4 - servo motor ports
4 - PID motors ports with full 10-bit back EMF and PID motor control
1 - 3.3V (5V tolerant) TTL serial port
2 - USB A host ports for connecting devices
1 - USB B device port to connect to your computer
1 - physical button

Other Features

- All sensor inputs have software enabled pull up resistor (digital 47k, analog 15k)
- Motor current up to 1A per port (up to 2A each with additional hardware)
- Isolated motor power up to 36V (with additional hardware)
- Servo ports output 6V
- I2C interface (with additional hardware)
- Arm 7 debug port (with additional hardware)
- JTAG port
- Vcc maximum current 500mA @3.3V, 1A @5V
- 7.4V 2000mAh Lithium Polymer battery pack (2s1p) 8C max discharge

Included Hardware

Each CBC v2 includes the CBC v2, USB cable, power adapter and USB camera.

CBC v2

USB Cable

AC Adapter

USB Camera

2. Quick Start

Turn On Your CBC v2

Plug you AC adapter into the wall and into the back of the CBC v2. You will see a green charger present light and a colored charge status light. Slide the power switch to the on position to boot the CBC v2. The CBC v2 will take 40-60 seconds to boot into the user interface. When the CBC v2 is fully booted you will see a screen similar to below.

Check the Software On Your CBC v2

The CBC is controlled by 3 main software component: the Userhook, the CBOB Firmware, and the CBC Operating System. You may want to update the Userhook and CBOB Firmware; you will not generally upgrade the CBC OS.

The Userhook controls the user interface, and the CBOB Firmware controls the break out board (the motors, servos and sensors). Check the CBC page at http://kipr.org/products/cbc-robot-controller to see and download the most recent release of each.

To **check the version of the software currently running on your CBC v2**, boot your CBC v2 and press the About button in the bottom left of the home screen.

In the bottom right of the screen, check the Userhook (UH) and Firmware (FW) versions. If either software is out of date, you can follow the instructions in the Appendices to update your CBC to the latest release.

These numbers should match the current version numbers on the website.

Install Software on Your Computer

Mac (OS X 10.4 and higher):

First, **install the most recent release of Xcode for your version of Os X**. You can find the Xcode installer on your Os X install disks, or download it from the Apple Developer Connection at http://developer.apple.com (you must register for a free account to login and download the software).

Next, double click on KISS-*x.x.x*.dmg file to mount the disk image. **Copy the KISS-C folder in the disk image to the Applications folder on your Mac.** You need to keep the KISS-C application and the library folders in the same KISS-C folder (programs you write can be kept wherever you wish).

There is no need to install a USB driver; appropriate drivers come with Os X.

Windows (XP or higher):

If you are using Windows XP, **connect the CBC v2 to the computer** using the included USB cable and turn on the CBC. For Windows Vista and and Windows 7, it is not necessary connect the CBC to your computer.

Double click on KISS-C installer (KISS-*x.x.x*.exe).That will open the KISS-C Intallation Wizard. Click Next to begin the installation process.

If you are using Windows 7, you will need to right click and select run as administrator.

On the Choose Components screen, make sure that both KISS-C and the CBC driver are selected, then click Next to choose an install location. It is recommended that you install KISS-C in the default Program Files folder**. Click Install to begin installing KISS-C to your computer.**

When the driver gets installed you will be prompted that the driver is unsigned. This is normal. Continue the install anyway.

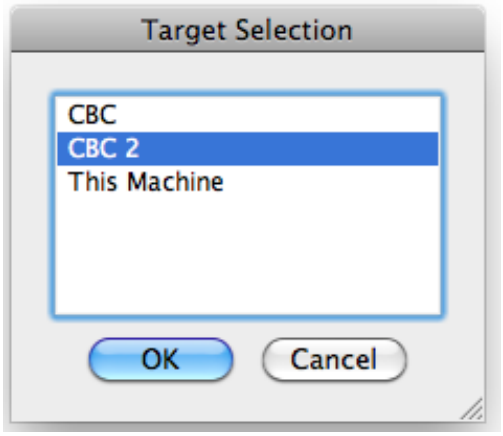
In Windows XP, you may be prompted to install the driver. Click next and then click next again (search for driver) and Windows XP will find and install the driver. KISS-C will be added to your program menu. A KISS-C shortcut will be placed on your desktop.

Download a Program to Your CBC v2

1. Connect the CBC v2 to your Computer

Turn on the CBC v2 on and connect it to your computer with the USB cable.

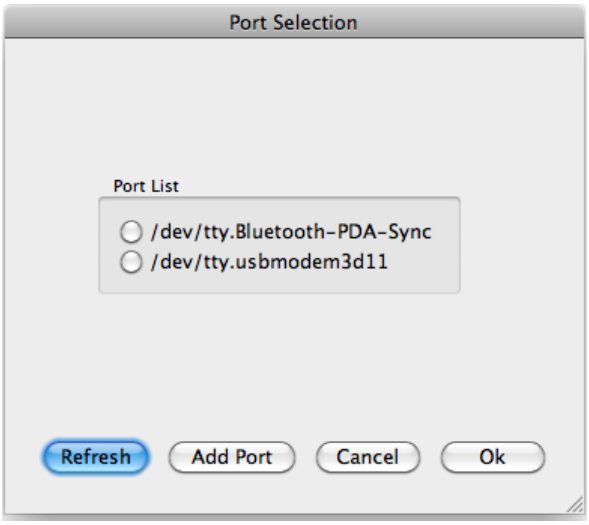
Launch KISS. When the program starts up, the Target Selection dialog will appear. Select CBC 2 from this menu and click OK to continue.

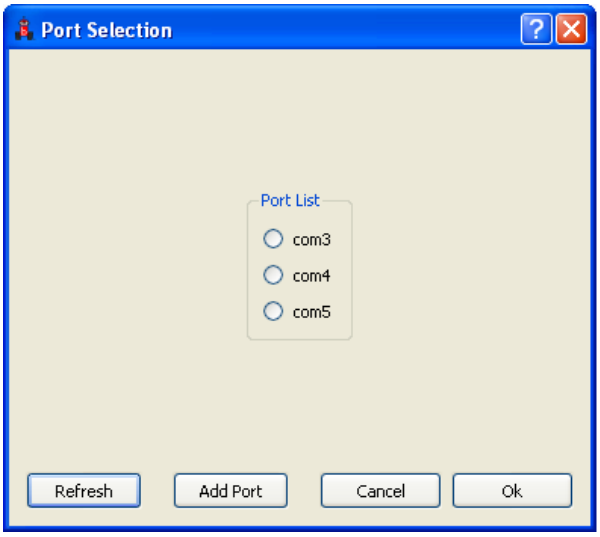


2. Select a Port to use with the CBC

Next, the **Port Selection** dialog will appear. Under Os X, the CBC will be detected as a USB modem and the window will appear as pictured.

Choose the USB modem from the Port Selection list and click Ok. If no USB modem port appears, make sure the CBC is powered on and click refresh.

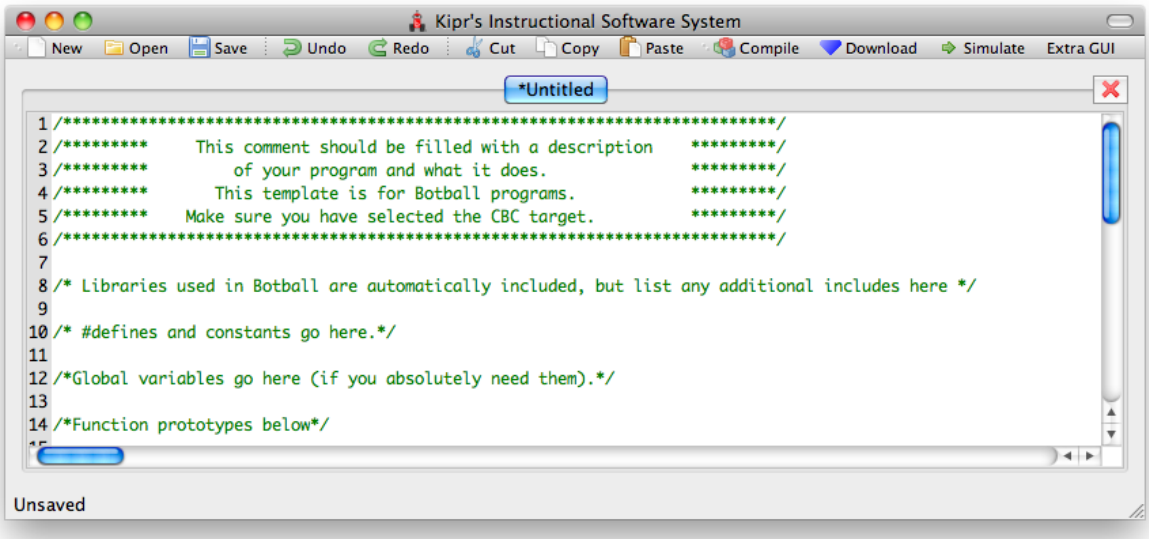




On Windows, the Port Selection dialog will show a list of available com ports. Typically, you should choose the highest numbered com port to connect to the CBC.

3. Create a new Program

Once you have selected a target and comport, you will be able to create your program in the KISS-C editor interface. Choose New from the top menu to begin a new program.



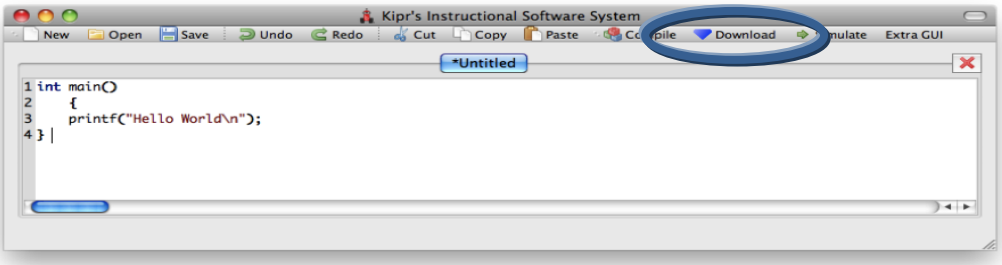
By default, KISS-C inserts a basic template useful for defining the main() function of your robot. In general, you may find this template useful, but, for the purpose of this exercise, we will start by deleting the default template.

After you delete the default template, **copy and paste the following C program into the editor**.

```
int main()
{
    printf("Hello World\n");
}
```

4. Download your program to the CBC

Next **click on the Download button** at the top left on the tool bar to transfer your program to the CBC.



If you have not already done so, you will be prompted to save your file. Save it as "HelloWorld.c". After you save, the status line at the bottom left of the window will display Downloading.

On the CBC v2 screen you will see a message with the file name and then the message compiling.

When the CBC v2 has finished compiling, press the run button at the top of the CBC v2 screen. The button will turn green while the program is running and Hello World should be printed to the screen.

3. Programming the CBC v2

Using KISS-C as the CBC v2 IDE

KISS-C is the ANSI C programming environment used to program the CBC v2. For more information on ANSI C programming pick up a ANSI C programming guide from your local book store. A highly recommended beginner's book is "Absolute Beginner's Guide to C" (2nd Edition) by Greg Perry.

To write a program for the CBC v2, plug in and turn on your CBC v2. Launch KISS-C and follow steps 1 and 2 of the previous section "Download a Program to your CBC v2".

Create a new file by clicking new or navigating to the File button and then New. A new program template will open. Edit and add to the template to suit your needs. When you have a complete program you can hit the compile button to check for compiler errors. First your program will be saved (if you have not saved this program before you will be prompted to name the file and pick the save location), then any errors will be displayed at the bottom of the screen. If you double click on an error, you will be taken to the line of the error, but remember that some errors have to do with previous lines of code. To download a program to the CBC v2, click the download button at the top. First your program will be saved, then compiled to check for errors, and finally downloaded to the CBC v2. You will not be able to download a program that does not compile.

CBC Functions

KISS-C has functions for interacting with the CBC v2. Some of the more common functions are presented in the following sections, and the complete list can be found in the Appendix.

Using Sensors

Any sensors purchased from the Botball store will work with the CBC v2. They are "keyed" so that there is only one orientation that all of the pins will be in holes.

Digital sensors typically only have two wires and are wired such that when the sensor is triggered the **SEN** and **GND** lines complete a circuit. Analog sensors can have two or three wires. In an analog sensor the resistance between the **SEN** and **GND** lines is varied. The third wire is connected to **VCC** which powers the sensor.

The default voltage on the CBC v2 from **VCC** to **GND** is 5V. See the appendix on how to change the **SEN** (sensor) voltage from 5V to 3.3V.

The specifications for creating your own sensor are in the appendix.

KISS-C Sensor Library Functions

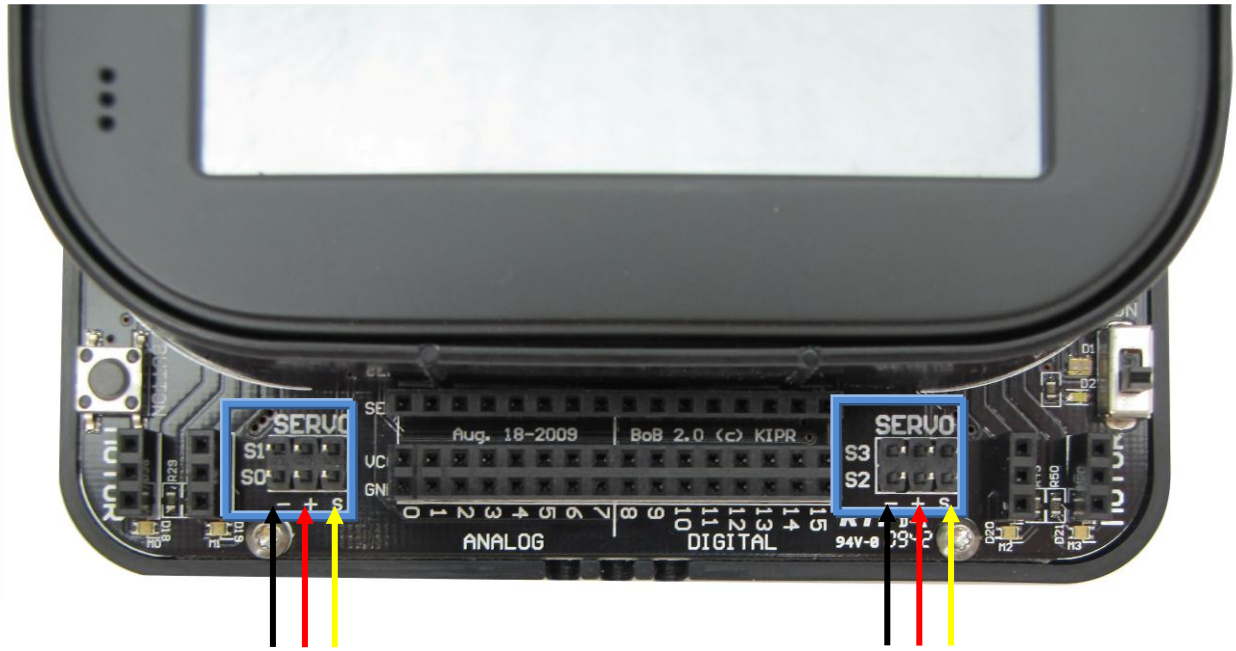
analog10(<port#>)

Returns the analog value of the port (a value in the range 0-1023). Analog ports are numbered 0-7. Light sensors and range sensors are examples of sensors you would use in analog ports.

digital(<port#>)

Returns 0 if the switch attached to the port is open and returns 1 if the switch is closed. Digital ports are numbered 8-15. Typically used for bumpers or limit switches.

Using Servos



Servos plug in to the servo ports on the front of the CBC v2. The arrows used above represent the most common coloring for servo cables (ground is black, positive is red, and signal is yellow), but yours may differ. If it does check the pin out to make sure that it is compatible with the CBC v2 before plugging it in. The servos run at 6 V.

KISS-C Servo Library Functions

enable_servos()

Enables power to the servo ports. This must be called before set_servo_position() to move a servo. When this function is called, the servo will default to position 1024 unless instructed to move elsewhere before enable_servos() is called.

disable_servos()

Disables power to servo ports. This is useful when you want to conserve battery life. Your servo will be at the mercy of any external forces and will not hold its position when disabled.

set_servo_position(<port#>, <position>)

Moves a servo plugged in a port to a position. Position ranges from 0 to 2047. The servo will immediately move to position, unless impeded by external force. Call the enable_servos() function before using this function.

Using Motors

The motors sold at the Botball Store will plug into any of the motor ports in any direction. When you instruct the motor to drive forward the motor will turn. If the motor is turning in the direction opposite of which you desire, un plug the motor, rotate the connector 180 degrees and plug the connector back in.

The motor ports run at 6V with a max current draw of 1A per motor port. Each group of ports (0 and 1, 2 and 3) is controlled by an h-bridge, so if you are going to be using close to 1A per motor, plug them into ports controlled by different h-bridges (i.e. 0 and 3) to prolong the life of your h-bridges. See the appendix for information on increasing the max current output per motor port.

Only the outside two positions of the connectors are used. When instructed to drive forward (blue LED is lit), the position closest to the screen is ground and the position closest to the front is power. See the appendix for the specifications on creating your own motor plugs.

KISS-C Motor Library Functions

motor(<motor#>,<power>)

Turns on a motor at a scaled PWM percentage. Power levels range from 100 (full forward) to -100 (full backward).

mav(<motor#>,<velocity>)

Move At Velocity moves a motor at a velocity indefinitely. Velocities range from -1000 to 1000 ticks per second.

mrp(<motor#>,<velocity>,<position>)

Move Relative Position moves a motor at a velocity from the current position to the current position plus the new position. Velocities range from 0 to 1000 ticks per second, and the position is a Long.

bmd(<motor#>)

Block Motor Done does not return until the previous mrp or mtp command finishes.

ao()

All Off turns off power to all motor ports.

KISS-C Other Commonly Used Library Functions

sleep(<seconds>)

Returns after a time equal to or slightly greater than the argument in seconds. Seconds is a Float.

printf(<string>,<arg1>,<arg2>,…)

Prints the string to the screen. If the string contains % codes then the arguments after the string will be printed in place of the % codes in the format specified by the code. See the Appendix for the full list of % codes.

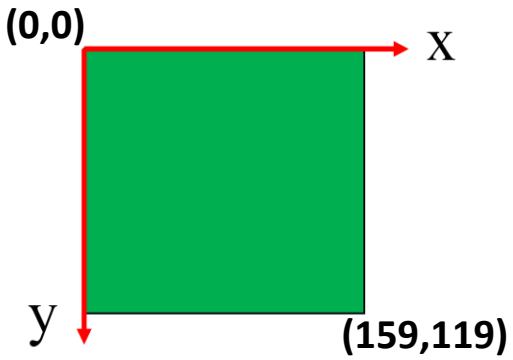
beep()

Creates a beep sound.

4. CBC v2 Vision

About vision tracking

The CBC v2 has a built in color vision tracking system. A USB web camera provides images at a rate of about 5 frames per second to the CBC v2. The CBC v2 then does real time processing of the images. The CBC v2 processes information on the 10 largest blobs on four color channels on each image. A blob is a group of adjacent pixels that are in the same color channel. As each frame is processed the blob information is stored in variables you can accesses to understand your environment. The color channels are taught to the CBC v2. The camera image size is 160 x 120. The upper left corner has coordinates (0,0) and the lower right has coordinates (159,119).



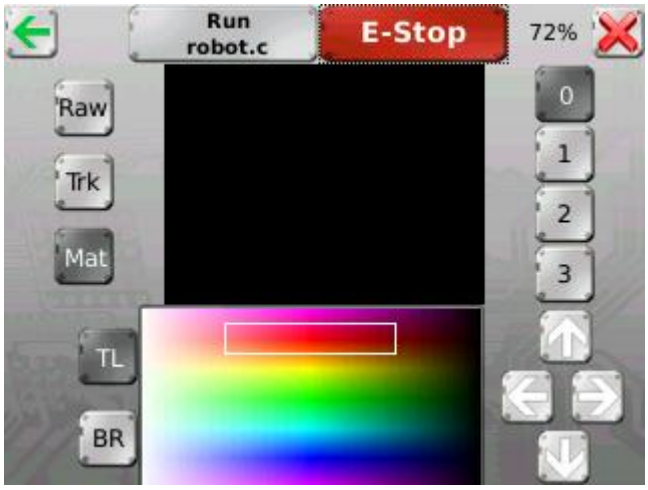
Teaching the CBC v2 Color Channels

While the CBC v2 is turned off, plug the USB camera into one of the USB A ports in the back of the CBC v2 as shown below.

Turn on the CBC v2 and allow it to boot to the main menu. Press the Vision button.

Press the Tracking button.

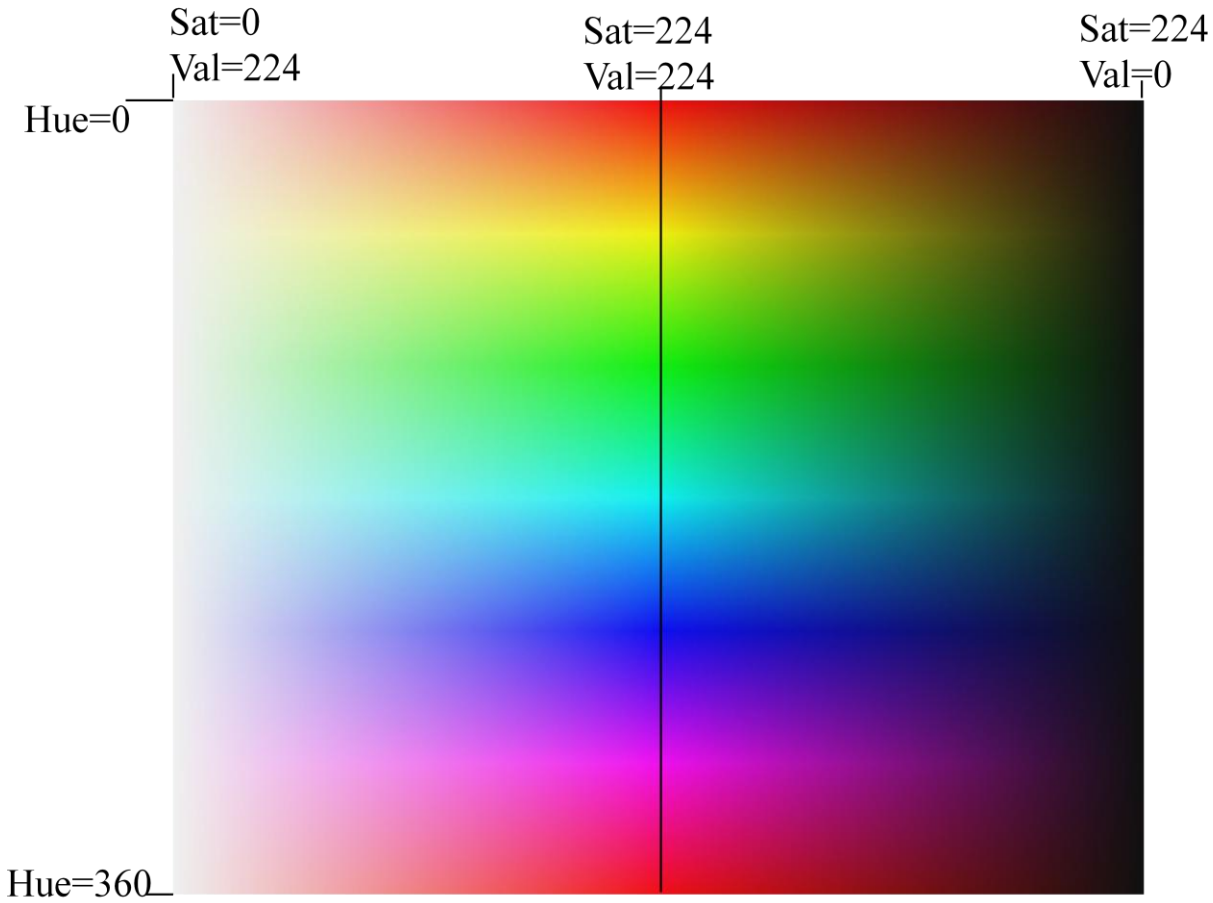
This pulls up the Vision Interface shown below.



On the top right there are buttons numbered 0-3 indicating the four channels. On the top left there are three buttons labeled Raw, Trk, and Mat. These stand for Raw camera image, Tracked camera image, and Matched camera image respectively. The Raw camera image is exactly what the camera sees. The Tracked camera image shows the pixels that fall into the bounding box area as white. The Matched camera image shows the pixels that fall into the bounding box area as white, and when the number of touching pixels that fall into the bounding box is greater than the minimum blob size shows a green bounding box around the blob and displays the centroid as a green plus.

Teaching the CBC v2 a color channel is changing the area and the location of the bounding box in the bottom center to encompass as many of the colors in the object to be tracked as well as limiting the number of non-tracked colors in the box. The bounding box is controlled by the arrows in the bottom right and the TL and BR buttons in the bottom right. TL and BR select which corner of the bounding box your are moving (Top Left or Bottom Right). Due to the nature of the bounding box, the Bottom Right corner is trapped on the right side of the HSV color selection plane, and the Top Left corner is trapped on the left side.

The HSV Color selection Plane (shown below) is a graphical representation of all of the colors the camera can see. The Hue value describes the color (i.e. red, yellow, blue), and the Saturation and Value represent how dark or light the color is (i.e. amout of black or white). Note that a value with a Saturation 224 and Value of 224 are the brightest (or pure) colors that the camera can see.



By default, the color channels are set to be a broad range of four popular (and easy to track) colors. Channel zero is red, one is yellow, two is green, and three is blue. You should adjust the bounding box for the color models to fit you specific application, but they give a good starting point.

When adjusting a color channel to your needs, first open the bounding box up so that all colors of the object you are tracking is within the bounding box. When the entire object is inside the bounding box start reducing the bounding box size to reduce the colors of environment being tracked. Note, that you may have to reduce the amount of your object in the bounding box in order to keep the environment from being tracked. You will probably not be tracking your whole object, due to inhomogeneous lighting conditions.

CBC v2 Vision Tracking Library Functions

These are the commonly used functions, for a complete list, see the appendix.

track_update()

Processes data for the current frame and make it available to the program. Always call this function before using any other camera functions so they reference the current data.

track_count(<channel>)

Returns the number of blobs on a channel. Call track_update() first.

track_x(<channel>,<number>)

Returns the x coordinate of a blob on a channel. The blobs are ranked from 0 to 9 with 0 being the largest blob size. Call track_update() first.

track_y(<channel>,<number>)

Returns the x coordinate of a blob on a channel. The blobs are ranked from 0 to 9 with 0 being the largest blob size. Call track_update() first.

Sample color tracking program if you have a servo

This sample program is a demo for using the camera on the CBC v2 if you have a servo. This program tracks an object on color channel 0 and points the servo at the object. If the object moves left the pointer points left.

Set Up

Attach the camera to the CBC v2. You need to set the color model on channel 0 to track an object you can move in front of the camera (the brighter the better). The servo needs to be set so that the pointer is just at edge of the camera's field of view. The servo also needs to be pointed so that when the servo is set to the midpoint (1024) it points at the center of the camera's field of view. For extra show use a pointer attached to the servo horn (of a color that is not in your color model). Finally download the program to the CBC v2.

Code

```
/* This program points a servo (that is plugged into port 0 and
centered on the camera's field of vision) towards an object that
fit 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");
}
```

Sample color tracking program if you do not have a servo

This sample program is a demo for using the camera on the CBC v2 if you have a servo. This program tracks an object on color channel 0 and lights up the motor ports that correspond to the object's location. If the object is in front of motor port 2 the motor port 2 light turns blue, and if the object moves to in front of motor port 1 the motor port 2 light turns off and the motor port 1 light comes on.

Set Up

Attach the camera to the CBC v2. You need to set the color model on channel 0 to track an object you can move in front of the camera (the brighter the better, but not blue). The camera needs to be pointed at the ground in front of the CBC v2 as close to the front as possible, without including the front of the CBC v2. The center of the camera's field of vision needs to be aligned with the center of the CBC v2 as well. Finally, download the program to the CBC v2.

Code

```
/*  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
correspond 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();
}
```

5. Troubleshooting

If at any point you need additional help, are uncomfortable completing a troubleshooting step, or there is a problem you cannot resolve, call Technical Support at **(405) 579-4609** between 9AM and 5PM Central Standard Time, or email **support@kipr.org**.

Problem	Solution
My CBC does not turn on.	Plug it into the charger. A solid green LED should come on indicating the charger is correctly plugged in to power. A charge status LED should also come on indicating the amount of charge in the CBC v2. If you get a flashing red LED, try un-plugging and re-plugging in the charger. If you continue to have the red blinking light, or a blinking green light, contact tech support.
All of the motor ports have red lights on when I boot my CBC.	You need to reload the Firmware (FW). See Appendix.
My CBC comes on but only loads to the splash screen and keeps rebooting continuously.	This behavior means that the Firmware (FW) is newer than the Userhook (UH). Reload the most current version of the Userhook and Firmware. See Appendix.
My CBC's touch screen is unresponsive, or it is difficult to press buttons in the user interface.	You need to recalibrate the touch screen. Turn off the CBC. Turn on the CBC and immediately press and hold on the touch screen. When you enter the special options menu you may remove your finger from the screen (if you do not get into the special options menu, turn off the CBC and try again). In the special options menu, select Restore factory setting at the bottom and then select OK (doing this will not affect stored programs or settings). Your CBC v2 will reboot twice, prompt you to recalibrate the screen, and then boot into the user interface.
My CBC will not compile code that successfully compiled on my computer.	You need to reload the Userhook (UH). Interrupting the Userhook upload process can cause this. See Appendix.
My camera is displaying a frozen or garbled image in the vision menu.	This happens when the CBC v2 loses the connection to the camera. Turn off the CBC, uninstall the camera, reinstall the camera, and turn the CBC back on.

Problem	Solution
My CBC v2 is not recognized by my Mac running OS 10.4, but is recognized by my other computer running OS 10.5 (or newer) or Windows.	The Firmware (FW) on your CBC v2 is out of date. Reload the most current version of the Firmware on your CBC. See Appendix.
My CBC v2 is not recognized by my Mac running OS 10.5 (or newer).	Turn on your CBC v2. Check to make sure that the CBC v2 is listed in the System Profiler. If the CBC is not listed there, check to make sure your USB cable is plugged in and functioning. If you are still having issues try a different USB port.
My CBC v2 is not recognized by my PC running Windows XP (or newer, not Windows 7 x64).	Turn on your CBC v2 and plug it in to your computer. Try reinstalling the driver from the KISS-C installer. Check to see if the CBC is listed in the Device Manager. If there is a device warning, you are having issues installing the driver. If no device is present in the Device manager, check to make sure your USB cable is plugged in and functioning. If you are still having issues try a different USB port.
My CBC v2 is not recognized by my PC running Windows 7 x64.	Turn on your CBC v2 and plug it in to your computer. Try reinstalling the driver from the KISS-C installer. Check to see if the CBC is listed in the Device Manager. If KIPRCBC is not listed as a USB device, you need to manually install the driver. Download the driver zip file from http://kipr.org/products/cbc-robot-controller . Unzip the driver, right click it and select install. If that does not work, open the root directory (usually c:/), then select the /windows folder. On the Tools menu in Windows Explorer, click Folder Options. Click the View tab in the new window. Under Hidden files and folders, click Show hidden files and folders. Now you should be able to see a /inf folder. Open the /inf folder and copy the kiprcbc.inf file into that folder. Unplug the CBC form the computer and turn the CBC off. Then turn the CBC back on and plug it back into the computer.

6. Appendices

Updating the Userhook or Firmware

The Firmware for the CBC v2 is made up of two parts, the Userhook, and Firmware. The most likely to be updated is the Userhook, followed by the Firmware. Follow the steps in the appropriate section below only if instructed to do so, or if the version numbers on your CBC v2 do not match the ones currently on:

http://kipr.org/products/cbc-robot-controller

Updating the Userhook

Equipment Needed: CBC v2, CBC v2 AC Adapter, Flash Drive (a high quality one if possible)

Step 1: Download the most up to date version of the userhook0 file from:

http://kipr.org/products/cbc-robot-controller

Step 2: Unplug any motors, servos, or sensors from your CBC v2 and plug it in to charge from the AC adapter.

Step 3: Copy the userhook0 file to the root (top level) of your USB flash drive.

Step 4: Un-mount the USB drive from your computer. If the drive is not properly un mounted, the userhook file could get corrupted. In OSX right click on the drive and choose Eject. In Windows under My Computer right click on the drive and select Eject.

Step 5: With the drive close by, turn on the CBC v2, but do not touch the screen.

Step 6: When you see the two circles on the screen, plug the USB drive with the userhook0 file into the **TOP** USB port in the back of the CBC v2. If the CBC v2 reports that an update has been requested, it has found the userhook0 file on your flash drive and the procedure will start in 10 seconds. If you get the screen with several circles, turn off your CBC, go to step 5 and in step 6 insert the flash drive sooner.

Step 7: You will see lots of text scroll by. Do not disturb the update process. After 5-10 minutes the CBC v2 will prompt you to remove your flash drive and restart the CBC v2. Follow these steps and the Userhook update is complete. The About screen should reflect the update, if not go to step 1.

Step 8: Remove the userhook0 file from the root of your flash drive.

Updating the CBOB Firmware

Equipment Needed: CBC v2, CBC v2 AC Adapter, USB Cable, Computer with USB port

Step 1: Download the most up to date version of the BoB Loader zip file from

http://kipr.org/products/cbc-robot-controller

Step 2: Install the CBC v2 driver. See section 2 **Quick Start** sub section **Install Software** if you do not have the driver installed.

Step 3: Unplug any motors, servos, or sensors from your CBC v2 and plug it in to charge from the AC adapter.

Step 4: Connect your CBC v2 to your computer with the included USB cable.

Step 5: In Windows unzip the bob loader zip file, which will create a current "bob loader" folder; open this folder and click on **bob_loader.exe** to run it. In OSX double click on the bob loader .dmg file**.**

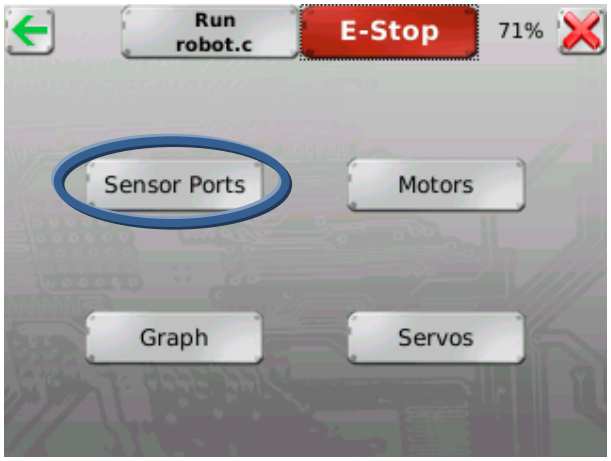
Step 6: Follow the on screen instructions to update the Firmware.

Step 7: If the BoB loader reports a successful upgrade, you are done.

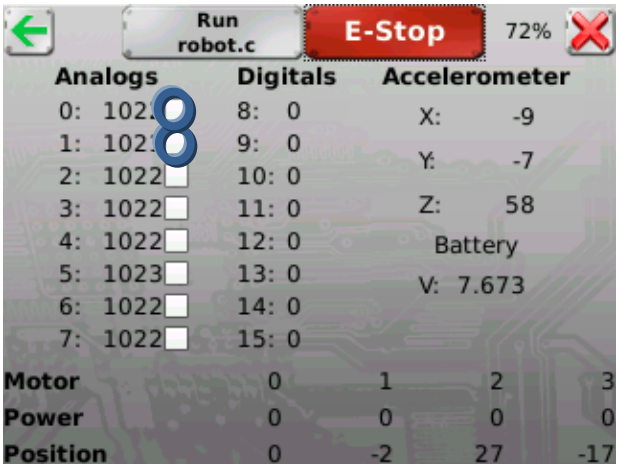
Disabling the pull up resistors on the analog ports manually

By default all of the analog pull up resistors are enabled. From the Main Menu, select the Sensors/Motors page. Note that when the CBC v2 is reset the pull up resistors are enabled.

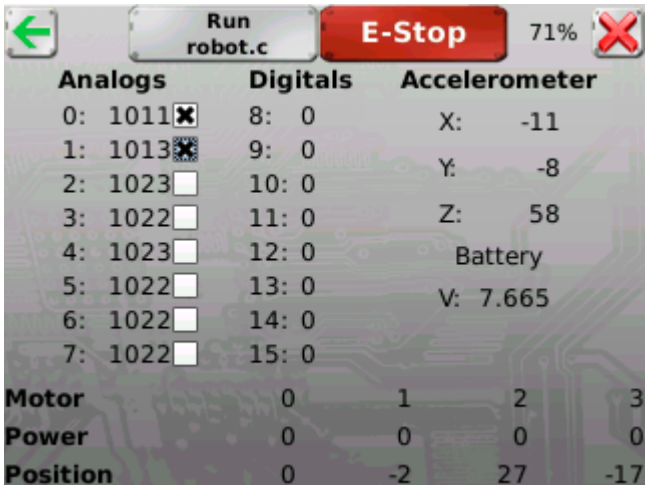
Next select Sensor Ports.



Now you will see the following screen.



When you touch the box next to an analog port, an X appears in the box letting you know that that analog port is now set to be a floating port as seen below.



In the previous picture, analog ports 0 and 1 are now set as floating ports and now no longer use the 15k pull up resistor.

Disabling the pull up resistors on the analog ports in your program

By default all of the analog pull up resistors are enabled. Use the following built in function to disable or enable the pull up resistors.

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

Each int corresponds to an analog port. To enable a port set the int to be 1; to enable a port, set the int to be 0. See the example below.

```
// 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);
}
```

In the example above, port 4 has the pull up resistors disabled. Note that in addition to disbling port 4 you have enabled all of the other ports, so be careful when using this function multiple times. You should also add a small sleep after this command to allow the states to change as shown above.

Controlling an iRobot Create with the CBC v2

The CBC v2 can communicate over a serial connection to control an iRobot Create. You will need to get a CBC Communication cable (available at https://botballstore.org). Once you have your CBC Communication cable follow the steps below to communicate with your iRobot Create.

Step 1: Connect the larger round plug of the CBC Communication cable to Mini-Din connection port in the iRobot Create (it is hidden under a removable cover above the charge port on the Create).

Step 2: Connect the 3 port female header as shown below. The power plug is optional, but when the iRobot Create is turned on your CBC v2 will charge from the iRobot Create's battery. Note that the red dot on the back label corresponds to the side the red wire is on. If you plug this connector in backwards you will not connect to the iRobot Create.

Step 3: Write a program for the CBC v2 that communicates with the iRobot Create and download it to the CBC v2.

Step 4: Set the iRobot Create on the ground.

Step 5: Power on the iRobot Create.

Step 6: Run your program on the CBC v2.

KISS-C iRobot Create Commonly Used Library Functions

These are the commonly used functions, for a complete list, see the appendices.

create_connect();

Connects the CBC v2 to the iRobot Create. Call this function first. By default the iRobot Create will be in "safe mode". This function returns 0 if successful and a negative number if not successful.

create_disconnect();

Disconnects the CBC v2 from the iRobot Create. Returns Create to proper state and resets XBC baud rate to KISS-C rate.

create_full();

Create will move however you tell it -- even if that is a bad thing. In particular, the Create will not stop and disconnect, even if it is picked up or the cliff sensors fire.

create_stop();

Stops the drive wheels.

create_drive_straight(<speed>);

Drives straight at a set speed. Speed range is 20-500mm/s.

create_drive_direct(<left motor speed>,<right motor speed>);

Specifies individual left and right motor speeds from 20-500 mm/s.

get_create_lbump(<lag>);

Returns 1 if left bumper is pressed, 0 otherwise. Data has been gathered within lag seconds, lag is a float (0.1 is recommended).

get_create_rbump(<lag>);

Returns 1 if right bumper is pressed, 0 otherwise. Data has been gathered within lag seconds, lag is a float (0.1 is recommended).

get_create_lfcliff(<lag>);

Returns 1 if left front cliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds, lag is a float (0.1 is recommended).

get_create_rfcliff(<lag>);

Returns 1 if right front cliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds, lag is a float (0.1 is recommended).

Sample Program for Controlling an iRobot Create with the CBC v2

Here is a program to make the iRobot Create drive in a circle with a radius of 0.25 meters at a speed of 200 mm/sec for 10 seconds and prints the distance traveled and the angle covered.

Set Up

Fully charge your CBC v2 and iRobot Create. Connect the CBC v2 to the iRobot Create with the CBC Communication cable. Place the CBC v2 into the cargo bay of the iRobot Create. Set the iRobot Create on the floor with an adequate area cleaned out in front of the iRobot Create. Note that if the iRobot Create detects a ledge with the cliff or wheel drop sensors (like being picked up), the program will stop and the iRobot Create will play a sad tone.

Code

```
/*  This is a program to make the iRobot Create drive in a
circle with a radius of 0.25 meters at a speed of 200
mm/sec for 10 seconds and prints the distance traveled
around the circle and the angle that the turn covered*/

int main()

{

create_connect();
set_create_distance(0);
set_create_normalized_angle(0);
create_drive(200, 250);
sleep(10);
create_stop();
create_sensor_update();
printf("distance = %d\n", get_create_distance(0.1));
printf("angle = %d\n", get_create_normalized_angle(0.1));
create_disconnect();
kissSimPause();


}
```

KISS-C Library Functions for the CBC v2

(alphabetic order)

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.

tone [Category: Output]

Format: void tone (float frequency, float length);
Produces a tone at pitch frequency (measured in Hertz) for length seconds. Returns when the tone is finished. Both frequency and length are floats.

up_button [Category: Sensors]

Format: int up_button();
Reads the value (0 or 1) of the move up button.

KISS-C Vision Library Functions for the CBC v2

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).

KISS-C iRobot Create Library Functions for the CBC v2

create_connect [Category: Create Function]

Format: int create_connect();
First step for connecting CBC to Create. Returns 0 if sucessful and a negative number if not. This function puts the Create in the create_safe mode.

create_disconnect [Category: Create Function]

Format: void create_disconnect();
Returns Create to proper state and resets XBC baud rate to KISS-C rate

create_start [Category: Create Function]

Format: void create_start();
Puts Create into active mode (with motors)

create_passive [Category: Create Function]

Format: void create_passive();
Puts Create into passive mode (no motors)

create_safe [Category: Create Function]

Format: void create_safe();
Create will execute all commands, but will disconnect and stop if drop or cliff sensors fire.

create_full [Category: Create Function]

Format: void create_full();
Create will move however you tell it -- even if that is a bad thing. In particular, the Create will not stop and disconnect, even if it is picked up or the cliff sensors fire.

create_spot [Category: Create Function]

Format: void create_spot();
Simulates a Roomba doing a spot clean

create_cover [Category: Create Function]

Format: void create_cover();
Simulates a Roomba covering a room

create_demo [Category: Create Function]

Format: void create_demo(int d);
Runs built in demos (see Create OI)

create_cover_dock [Category: Create Function]

Format: void create_cover_dock();

Create roams around until it sees an IR dock and then attmpts to dock

The functions starting "get_create_" all take a single floating point argument which indicates that the sensor data should be updated if it is older than the argument. In other words, calling get_create_lbump(0.1) indicates that if the sensor data is less than 1/10th of a seconds old, then the cached value will be returned, but if it is older than a new value from the Create will be returned and cached. Calling with an argument of 0 will force a new value to be retrieved from the Create. Note that there is significant overhead in talking with the Create and so the lag times passed to these functions should not be smaller than needed. Values less than 0.1 should be avoided and larger times should be used for the angle and distance functions.

get_create_mode [Category: Create Function]

Format: int get_create_mode(float lag);

Returns the Create's mode (0 off; 1 passive; 2 safe; 3 full). Data has been gathered within lag seconds.

get_create_lbump [Category: Create Sensor Function]

Format: int get_create_lbump(float lag);

returns 1 if left bumper is pressed, 0 otherwise. Data has been gathered within lag seconds.

get_create_rbump [Category: Create Sensor Function]

Format: int get_create_rbump(float lag);

returns 1 if right bumper is pressed, 0 otherwise. Data has been gathered within lag seconds.

get_create_lwdrop [Category: Create Sensor Function]

Format: int get_create_lwdrop(float lag);

returns 1 if left wheel has dropped, 0 otherwise. Data has been gathered within lag seconds.

get_create_cwdrop [Category: Create Sensor Function]

Format: int get_create_cwdrop(float lag);

returns 1 if caster wheel has dropped, 0 otherwise. Data has been gathered within lag seconds.

get_create_rwdrop [Category: Create Sensor Function]

Format: int get_create_rlwdrop(float lag);

returns 1 if right wheel has dropped, 0 otherwise. Data has been gathered within lag seconds.

get_create_wall [Category: Create Sensor Function]

Format: int get_create_wall(float lag);

returns 1 if wall is detected by right facing sensor, 0 otherwise. Data has been gathered within lag seconds.

get_create_lcliff [Category: Create Sensor Function]

Format: int get_create_lcliff(float lag);
returns 1 if left cliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds.

get_create_lfcliff [Category: Create Sensor Function]

Format: int get_create_lfcliff(float lag);
returns 1 if left front cliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds.

get_create_rfcliff [Category: Create Sensor Function]

Format: int get_create_rfcliff(float lag);
returns 1 if right frontcliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds.

get_create_rcliff [Category: Create Sensor Function]

Format: int get_create_rcliff(float lag);
returns 1 if right cliff sensor is over black or a cliff, 0 otherwise. Data has been gathered within lag seconds.

get_create_vwall [Category: Create Sensor Function]

Format: int get_create_vwall(float lag);
returns 1 if a virtual wall beacon is detected, 0 otherwise. Data has been gathered within lag seconds.

get_create_overcurrents [Category: Create Sensor Function]

Format: int get_create_overcurrents(float lag);
returns the overcurrent status byte where 16's bit indicates overcurrent in left wheel; 8's bit in right wheel, 4's bit is LD2, 2's bit is LD0 and 1's bit is LD1. Data has been gathered within lag seconds.

get_create_infrared [Category: Create Sensor Function]

Format: int get_create_infrared(float lag);
returns the byte detected from the remote control, 255 if no byte has been detected. Data has been gathered within lag seconds.

get_create_advance_button [Category: Create Sensor Function]

Format: int get_create_advance_button(float lag);
returns 1 if advance button is being pressed, 0 otherwise. Data has been gathered within lag seconds.

get_create_play_button [Category: Create Sensor Function]

Format: int get_create_play_button(float lag);
returns 1 if play button is being pressed, 0 otherwise. Data has been gathered within lag seconds.

get_create_distance [Category: Create Sensor Function]

Format: int get_create_distance(float lag);
returns the accumulated distance the Create has traveled since it was turned on or the distance was set. Moving backwards reduces this value. The distance is in millimeters. The value is truncated to the nearest millimeter every time this function is updated so having the lag time be too small will cause an artificially small value. Suggested lag values are no smaller than 5/speed where speed is the Create's speed in mm/sec. Data has been gathered within lag seconds.

set_create_distance [Category: Create Sensor Function]

Format: void set_create_distance(int dist);
Sets the current value that will be returned by get_create_distance to the value dist.

get_create_normalized_angle [Category: Create Sensor Function]

Format: int get_create_normalized_angle(float lag);
returns the accumulated angle the Create has turned since it was turned on or the distance was set -- normalized to the range 0 to 359 degrees. Turning CCW increases this value and CW decreases the value. The value is truncated to the nearest degree every time this function is updated so having the lag time be too small will cause an artificially small value. Suggested lag values are no smaller than 10/(difference between left and right wheel speeds). Data has been gathered within lag seconds.

get_create_total_angle [Category: Create Sensor Function]

Format: int get_create_total_angle(float lag);
returns the accumulated angle the Create has turned since it was turned on or the distance was set. Turning CCW increases this value and CW decreases the value. The value is truncated to the nearest degree every time this function is updated so having the lag time be too small will cause an artificially small value. Suggested lag values are no smaller than 10/(difference between left and right wheel speeds). Data has been gathered within lag seconds.

set_create_normalized_angle [Category: Create Sensor Function]

Format: void set_create_normalized_angle(int angle);
Sets the current value that will be returned by get_create_normalized_angle to the value angle.

set_create_total_angle [Category: Create Sensor Function]

Format: void set_create_total_angle(int angle);
Sets the current value that will be returned by get_create_total_angle to the value angle.

get_create_battery_charging_state [Category: Create Sensor Function]

Format: int get_create_charging_state(float lag);
0-not charging; 1-recondition charging; 2-full charging; 3-trickle charging; 4-waiting; 5-charge fault. Data has been gathered within lag seconds.

get_create_battery_voltage [Category: Create Sensor Function]

Format: int get_create_battery_voltage(float lag);
returns the battery voltage in mV. Data has been gathered within lag seconds.

get_create_battery_current [Category: Create Sensor Function]

Format: int get_create_battery_current(float lag);
returns the current flow in mA. Data has been gathered within lag seconds.

get_create_battery_temp [Category: Create Sensor Function]

Format: int get_create_battery_temp(float lag);
returns the battery temperature in degrees C. Data has been gathered within lag seconds.

get_create_battery_charge [Category: Create Sensor Function]

Format: int get_create_battery_charge(float lag);
returns the battery charge in mAh. Data has been gathered within lag seconds.

get_create_battery_capacity [Category: Create Sensor Function]

Format: int get_create_battery_capacity(float lag);
returns the battery capacity in mAh. Data has been gathered within lag seconds.

get_create_wall_amt [Category: Create Sensor Function]

Format: int get_create_wall_amt(float lag);
returns 12 bit analog value from wall sensor. Data has been gathered within lag seconds.

get_create_lcliff_amt [Category: Create Sensor Function]

Format: int get_create_lcliff_amt(float lag);
returns 12 bit analog value from left cliff sensor. Data has been gathered within lag seconds.

get_create_lfcliff_amt [Category: Create Sensor Function]

Format: int get_create_lfcliff_amt(float lag);
returns 12 bit analog value from left front cliff sensor. Data has been gathered within lag seconds.

get_create_rfcliff_amt [Category: Create Sensor Function]

Format: int get_create_rfcliff_amt(float lag);
returns 12 bit analog value from right frontcliff sensor. Data has been gathered within lag seconds.

get_create_rcliff_amt [Category: Create Sensor Function]

Format: int get_create_rcliff_amt(float lag);
returns 12 bit analog value from right cliff sensor. Data has been gathered within lag seconds.

get_create_bay_DI [Category: Create Sensor Function]

Format: int get_create_bay_DI(float lag);
returns byte containing all digital sensors from the cargo bay: 16'bit for pin 15, 8's bit for pin 6, 4's bit for pin 18, 2's bit for pin 5 and 1's bit for pin 17. Data has been gathered within lag seconds.

get_create_bay_AI [Category: Create Sensor Function]

Format: int get_create_bay_AI(float lag);
returns 10 bit analog value on pin 4 from the cargo bay. Data has been gathered within lag seconds.

get_create_song_number [Category: Create Sensor Function]
Format: int get_create_song_number(float lag);
returns currently selected song 0-15. Data has been gathered within lag seconds.

get_create_song_playing [Category: Create Sensor Function]

Format: int get_create_song_playing(float lag);
returns 1 if song is playing, 0 otherwise. Data has been gathered within lag seconds.

get_create_number_of_stream_packets [Category: Create Sensor Function]

Format: int get_create_number_of_stream_packets(float lag);
if data streaming is being used, it returns the size of the stream. Data has been gathered within lag seconds.

get_create_requested_velocity [Category: Create Sensor Function]

Format: int get_create_requested_velocity(float lag);
asks Create how fast it was told to be moving -500 to 500mm/s and returns that value. Data has been gathered within lag seconds.

get_create_requested_radius [Category: Create Sensor Function]

Format: int get_create_requested_radius(float lag);
asks Create the size of its turning radius and returns that value. Data has been gathered within lag seconds.

get_create_requested_right_velocity [Category: Create Sensor Function]

Format: int get_create_requested_right_velocity(float lag);
asks Create how fast it was told to be moving right wheel and returns that value. Data has been gathered within lag seconds.

get_create_requested_left_velocity [Category: Create Sensor Function]

Format: int get_create_requested_left_velocity(float lag);
asks Create how fast it was told to be moving left wheel and returns that value. Data has been gathered within lag seconds.

create_stop [Category: Create Movement Function]

Format: void create_stop();
Stops the drive wheels

create_drive [Category: Create Movement Function]

Format: void create_drive(int speed, int radius);
Drives in an arc (see below for point turns and straight). Speed range for all commands is 20-500mm/s

create_drive_straight [Category: Create Movement Function]
Format: void create_drive_straight(int speed);
Drives straight at speed in mm/s

create_spin_CW [Category: Create Movement Function]

Format: void create_spin_CW(int speed);
spins Clockwise with edge speed of speed in mm/s

create_spin_CCW [Category: Create Movement Function]

Format: void create_spin_CCW(int speed);
spins Counterclockwise with edge speed of speed in mm/s

create_drive_direct [Category: Create Movement Function]

Format: void create_drive_direct(int r_speed, int l_speed);
Specifies individual left and right speeds in mm/s

create_advance_led [Category: Create Function]

Format: void create_advance_led(int on);
Pass 1 to turn on light and 0 to turn it off

create_play_led [Category: Create Function]

Format: void create_play_led(int on);
Pass 1 to turn on light and 0 to turn it off

create_power_led [Category: Create Function]

Format: void create_power_led(int color, int brightness);
color 0 is red and 255 green; brightness 0 is off and 255 is full brightness

create_load_song [Category: Create Function]

Format: void create_load_song(int num);
Picks a song from gc_song_array[16][33]. Each row is a song. First column of each song is number of notes (max is 16). Remaining columns alternate between pitch and duration. See Create OI for details.

create_play_song [Category: Create Function]

Format: void create_play_song(int num);
Plays any of the songs that have been loaded