2010 IEEE International Conference on Robotics and Automation
Anchorage Convention District
May 3-8, 2010, Anchorage, Alaska, USA

The CBC: a LINUX-based Low-Cost Mobile Robot Controller

David P. Miller, Matthew Oelke, Matthew J. Roman, Jorge Villatoro & Charles N. Winton

Abstract— **Over the last five years, a number of powerful robotics controllers have become available. Only a small percentage of these are suitable for general use in robotics. In particular, they trivially interface with a large variety of sensors and effectors, have a well constructed software IDE that works with a standard programming language, are self contained and are easy to use. The CBC2 is a new robot controller that meets these conditions. The CBC2 includes an ARM 7 based DAQ/Motor control system, an ARM 9-based CPU/Vision processor running LINUX, an integrated color display and touch screen. The CBC2 is both a USB host (allowing the use of standard cameras, mass storage and network interfaces) and a USB device for software downloads. This paper describes the CBC2, its capabilities and the KISS-C IDE and associated libraries which include functionality ranging from color tracking to PID motor control. The CBC/KISS-C system was adopted by the Botball Robotics Education program in 2009, and was used in about three hundred schools. Based on feedback from that experience, design improvements were made and the CBC2 is being used in the 2010 Botball program and is also being made available for other uses.**

I. THE NEED FOR A NEW ROBOT CONTROLLER

With the creation of the 1991 6.270 board and the accompanying IC programming environment [1], educational robotics was off and running. Since that time, a number of robot controllers have been developed by educational institutions, non-profits and commercial enterprises. The most popular controller through most of the 1990's was the Handy Board [2], a direct descendant of the 6.270 electronics, and one that shared its IC programming environment. In the very late 1990's LEGO introduced the RCX controller which, while limited in both inputs and outputs when compared to the Handy Board, was well marketed and relatively inexpensive. It, and the follow-on NXT have dominated the inexpensive robot controller market ever since. These systems use proprietary software, sensors and motor interfaces. They promote ease of use over more powerful traditional software and hardware techniques. While millions of K12 students have been exposed to robotics throughout the US through the RCX and NXT, that exposure has not resulted in an increase in engineering students; in fact the three primary robotics disciplines of CS, EE, and ME have all had relatively flat or declining per capita numbers since the introduction of the LEGO RCX[3].

Despite the market dominance of the LEGO controllers, several other controllers have been developed in recent years to overcome limitations in the RCX and NXT. Among these are the GUMStix, Qwerk, Arduino and Brainstem systems. These systems vary widely in capabilities and price, but all lack a capable integrated onboard user interface; for the most part the user interacts with the system through a PC. For an autonomous robot project in classroom settings, this is a major stumbling block. These systems also require a certain level of electronics integration which is beyond the capabilities of most K-12 classroom setups. These issues were addressed successfully with the introduction of the XBC controller [4] in 2005. The XBC incorporated a Nintendo GBA, allowing the buttons and graphical display to be used as the UI hardware. A more detailed review of robot controllers suitable for K-20 use can be found in [5].

While the XBC provided a significant improvement in the user interface compared to previous controllers, the incorporation of the GBA led to some unforeseen difficulties: 1) the nature of the interface between the DAQ board and the GBA was a significant failure point and also an opportunity for static damage to the FPGA used in the XBC; 2) the XBC was tied to the availability of the GBA – a product that was discontinued by Nintendo shortly after the introduction of the XBC.

II. THE CBC2/KISS-C SYSTEM

The CBC2/KISS-C system is a new hardware/software system developed for general robotics use. It is a successor for two earlier hardware/software systems, the HandyBoard/IC system [2] and the XBC/IC system [4], both of which have been used extensively by the KISS Institute for Practical Robotics (KIPR) in support of its educational robotics programs [6][7]. The XBC controller was an upward compatible replacement for the HandyBoard, providing a number of enhancements and employing an enhanced version of the Interactive C (IC) software. The CBC2 is a robotics controller that is a replacement for the XBC, maintaining upward compatibility for motor and sensor technology, while adding new functionality and capability. KISS-C is a replacement for IC, using an IDE similar to the one embedded in IC, but providing the full capability of ANSI C. A preliminary version of the CBC was used in the 2009 Botball season. Based on feedback from hundreds of users, the CBC

This work was supported in part by KIPR through a grant from NASA's Robotics Alliance Program

D. Miller and M. Roman are with the School of AME at the University of Oklahoma, Norman OK, 73072, USA dpmiller@ou.edu and mattjroman@gmail.com
M. Oelke and J. Villatoro are with KISS Institute for Practical Robotics, Norman OK, 73069, USA matthew.oelke@kipr.org and jorge@kipr.org
C. Winton is with School of Computing, University of North Florida, Jacksonville FL, 32224, USA cwinton@unf.edu

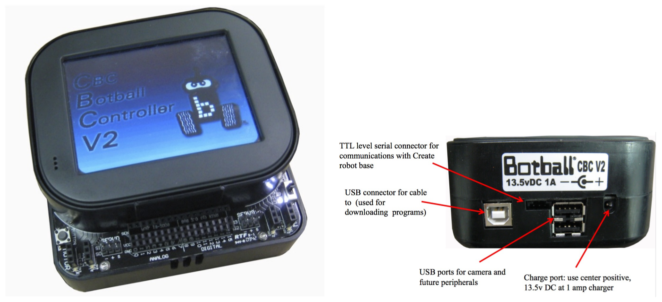

CBC Botball Controller V2
TTL level serial connector for communications with Create robot base
USB connector for cable to (used for downloading programs)
Botball CBC V2
13.5vDC 1A
USB ports for camera and future peripherals
Charge port: use center positive, 13.5v DC at 1 amp charger

Fig. 1. *The CBC2 Robot Controller, front & back*

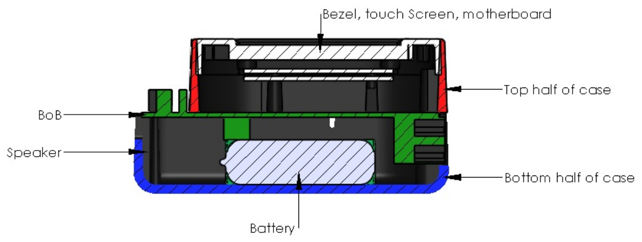


Fig. 2. *CBC2 cut away view*

hardware and software was redesigned, and the CBC2 (Fig 1) is now available for Botball and general use.

The remainder of this paper will go over the CBC2 hardware, software development environment, and library. We will also briefly discuss educational and other applications for the CBC2/KISS-C system.

III. THE CBC2 HARDWARE

The CBC2 electronics consists of a motherboard, a breakout board (BoB), a display/touchscreen and a battery. The subsections below describe how this is packaged and the design of the break-out board. The motherboard has been fully documented in [8] and was also used in the original *Chumby*[9].

A. Mechanics

The CBC2 is housed in a durable, two part injection molded plastic case (Figure 2). The top half of the case houses the screen bezel, touch screen, and motherboard. Those components are held to the top half of the case with screws. The motherboard plugs into the BoB via an 11x2 header – providing connections for a two-wire interface (TWI), SPI bus, two external USB ports, power and sound. A flex cable is used to connect the motherboard to a 1GB flash disk which is in an internal USB socket on the BoB. The bottom half of the case houses a lithium polymer (LiPo) battery pack and speaker. The two halves of the case screw together encapsulating the BoB and providing support for the ports on the back. The sensor ports, motor ports, servo ports, a software accessible button, and a power switch are exposed in the front of the CBC2 just below the touch screen and supported by the bottom half of the case (see top part of Fig 5).

B. The BoB

The Breakout Board (BoB) is responsible for all actuator and sensor I/O, supervising battery health, and communicating this shared data with the motherboard. An Atmel ARM7 32 bit MCU running at 48 MHz is the foundation the BoB is built on. The ARM7 was selected because of its speed, availability, and wide range of embedded communication and I/O ports, including 16 10-bit ADCs. Below is a list of the exposed port connections on the BoB.

- 8 digital
- 8 analog
- 4 motor with back-emf
- 4 standard hobby servo
- 3.3V serial
- USB
- TWI
- SPI

1) communication: Multiple communication protocols are used to send and receive data from the ARM7 to various parts of the BoB. The BoB and motherboard communicate through an SPI bus on the motherboard connection header. A USB B port and a 3.3 volt serial port are located across the back of the BoB. Both of these ports are connected to separate UART controllers on the ARM7 chip. The remaining USB A ports pass directly to the motherboard connection header. The accelerometers are connected to the ARM7 through a two wire interface (TWI) bus. For expansion and debugging purposes the lines of the TWI are easily accessible through a three pin connector on the BoB.

2) sensors: External digital and analog sensors may be connected through a three pin (signal, power, ground) connection on the exposed end of the BoB. Eight digital input/output ports and eight analog input ports are available. The power rail on the I/O ports can be set at 3.3 or 5 volts for all ports through a jumper. All digital ports have a software selectable 47kΩ pull-up resistor and are 5 volt tolerant inputs. Each analog port has 10 bit resolution and a software selectable 12kΩ pull-up resistor.

Internal sensing: battery voltage, acceleration and motor feedback, is done through 8 ADCs located on the ARM7 chip. The BoB includes a three axis accelerometer with set at $\pm$2g's and 8 bit resolution.

3) charging circuit: A 7.4 volt lithium polymer (LiPO) battery supplies power for both the BoB and motherboard through the motherboard connection header. At full charge the 2000 milliamp hour battery is capable of running the system minus any external actuators for over six hours. A Texas Instruments charge controller specifically designed for LiPO battery packs requires about three hours to replenish power. For fully discharged batteries the controller preconditions the pack before entering full charge mode. A charge is completed when the controller senses rapid increases in battery temperature, if the cell voltage is greater than a predefined threshold, or if the three hour timer has runout. The charger checks the battery state every few minutes when in trickle mode and will reinitiate charging if the battery falls below the nominal charged voltage. The battery state is displayed on the BoB through two colored LEDs which can turn on, off or flash giving six distinct battery/charge states.

4) actuator circuitry: Motor and servo power are directly connected to the battery. Each motor port is capable of supplying up to 2 amps of current through easily replaceable socket mounted H-bridges on the bottom of the board. Motor ports can be controlled directly by varying the duty cycle of the PWM signal sent to each motor or through closed loop PID control using back-emf [10].

Measuring the back electromotive force to obtain closed loop motor control was selected for two reasons. First it does not require any external hardware to sense motor position. Second it has an acceptable degree of accuracy and precision when compared with the typical motors used. The back-emf circuitry is based on the XBC [4], the electromotive voltage from the motor is read with the 10 bit ADC. This voltage is proportional to the rotational speed the motor is spinning at. The integral of speed over the time increment between measurements is taken to obtain the motors rotational position. Each motor has an associated 32 bit position counter updated prior to calculating the PID control. The motor control loop and all sensors are updated at 100Hz. The PID gains are tuned for modified hobby servo motors but are accessible through software and can be changed on the fly.

IV. CBC2 SOFTWARE

Programming the CBC2 involves writing C code in the KISS-C IDE. The code can be simulated on the user's PC, since it is standard C code, and there are simulator versions of all of the custom libraries. The source code is then downloaded (via USB) to the CBC2, where it is compiled using the native GCC compiler. The user then interacts through the CBC2 UI to run the code, interact with the user programs, view sensor data or set parameters for the vision system.

A. KISS-C IDE

The KISS-C IDE is based on Qt [11] and QScintilla [12]. Qt provides a powerful set of tools for cross-platform UI devlopment, and QScintilla provides a text editing widget that provides syntax highlighting, auto indenting, and tool tip support. These are combined into a pluggable system that allows a software developer to write simple plugins that add features such as compiler support and hardware support for robotics controllers. The plugin provided with the CBC2 has features for compiling, simulating, and downloading code to the CBC.

Compiling code is a simple process utilizing GNU's GCC compiler in Linux, Apple's GCC compiler on OS X, and MinGW's port of GCC on Windows. GCC was chosen because it is open-source, widely available, and ANSI standards compliant. A compatibility library is provided so that any program that is intended to run on the CBC2 will simulate properly on a user's computer. This compatibility layer relies on GLFW, a library that was intended to make a simple cross-platform api available for the purpose of opening an OpenGL window, capturing user input, and supporting multi-threading across all platforms.

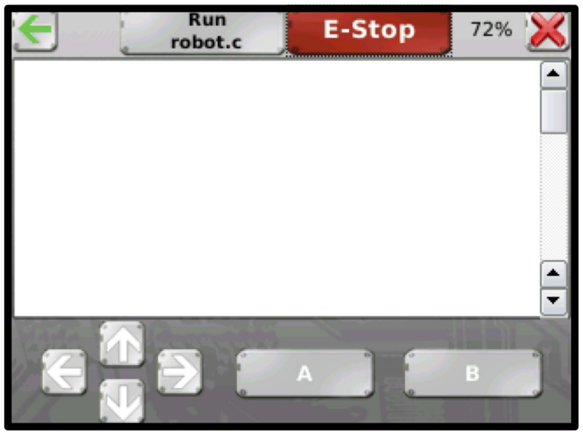


Fig. 3. *The CBC2 console screen with buttons*

The CBC2 simulator simply compiles the code and runs it on the user's computer. For standard programs that do not make use of any of the CBC2's special features, this is straightforward. For programs that try to access any of the CBC2's sensor ports, or perform motor control, servo control, send serial messages or make use of the CBC2 touch screen, the CBC2 simulation library comes into play. The CBC2 simulation functions access a graphical window that displays the status of the CBC2's input and output ports (see the right edge of Figure 8). The simulator is written in C and makes use of a 2D graphics library (based on OpenGL) which is included with KISS-C. To simulate the buttons on the CBC2 UI, an unbuffered keystroke function is used. The four arrow keys, `A` and `B` keys are used to represent the corresponding touch screen buttons accessible from the CBC2 console window (Figure 3). The period key is used to simulate the CBC2's one physical push button. The mouse can be used to move sliders to set the values of the various analog ports and accelerometer channels. The 8-5 keys are used to toggle the state of digital ports eight to fifteen. The simulator runs in a separate thread that is initiated automatically by KISS-C. No changes to user code are needed between running it on the simulator and running it on the CBC2 hardware.

Downloading code requires a serial port, which can be selected when KISS-C is started or selected later on. The download includes code in the user's current window and any code added through an `#include` statement. The code is then compiled on the CBC2, normally error free, but if any errors were missed in compiling on the user's computer, they are displayed on the CBC2's screen. It is intended that any code that runs in the simulator will download to the board and run (hopefully) the same way on the CBC2 with one caveat; users' computers vary widely in processing power, potentially producing timing induced differences in program behavior on the CBC2.

KISS-C's plugin system provides an API for adding capabilities to the basic editor. This means plugins for Java,

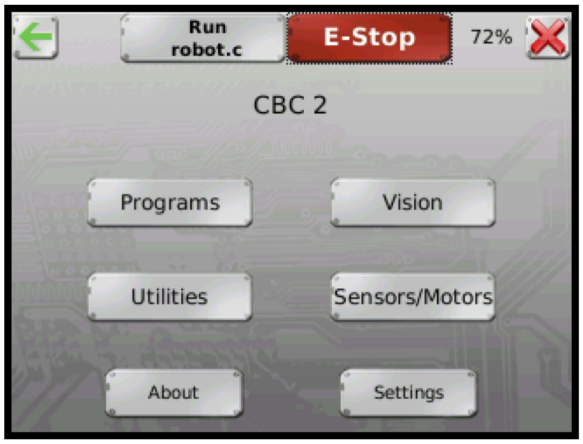


Fig. 4. *The CBC2 main menu screen*

Ruby, Python, etc. could all be developed for this system with the eventual goal that software could be written for the CBC2 in a variety of languages.

B. The CBC2 UI

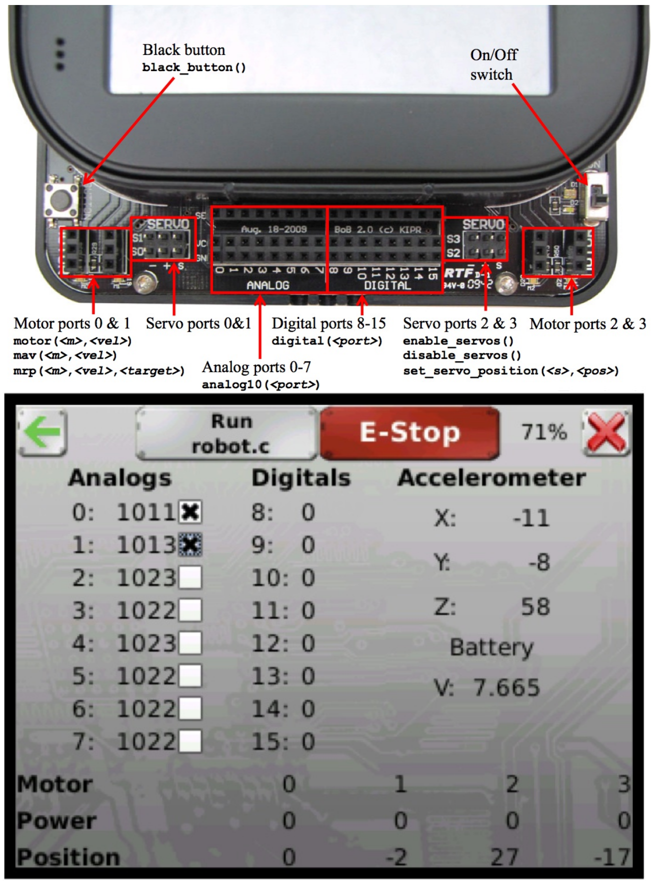
The CBC2 UI provides an interface for compiling and running programs, checking sensor values, and configuring the camera parameters, relying heavily on Qt/Embedded and Tslib. Qt/Embedded provides the user interface functionality, while Tslib provides the underlying touch screen library.

After the CBC2 has booted, a main menu screen appears (Figure 4). The user can then select a variety of other screens. The vision screen is described in Section IV-D. The console screen (Figure 3) is used to display program output and to access program readable buttons. The Program screen is brought up when a program is downloaded and displays compiler messages. It also provides access to the file system, including USB mounts. The Sensor screen (Figure 5) displays the current values for the different sensor ports as well as BEMF data for the motor ports. The screen is updated at about 10Hz. In addition, the CBC2 can also display values from up to three sensors (any of the sixteen external sensor ports plus the three accelerometer channels) in oscilloscope mode. The screen shots in Figure 6 demonstrate this display mode.

When the UI loads, it starts internal services to handle internal and serial communications. In particular, serial communications use data compression and streaming capabilities provided by Qt, allowing for more cross-platform compatibility, as well as simpler and more robust software.

C. CBC2 Firmware & User Hook

The CBC2 software upgrade process provides a method for re-loading or upgrading the software for each of the two ARM processors in the CBC2. For the BoB, the CBC2 is powered on while holding down the black button. This turns on all of the motor LEDs and puts the BoB processor in a special mode. A simple serial application is run on the user's computer, connected via USB to the CBC2 USB-B port. This



Analogs		Digitals	
0:	1011	8:	0
1:	1013	9:	0
2:	1023	10:	0
3:	1022	11:	0
4:	1023	12:	0
5:	1022	13:	0
6:	1022	14:	0
7:	1022	15:	0

Accelerometer
X: -11
Y: -8
Z: 58
Battery
V: 7.665

Motor	0	1	2	3
Power	0	0	0	0
Position	0	-2	27	-17

Fig. 5. *The CBC2 ports and sensor display screen*

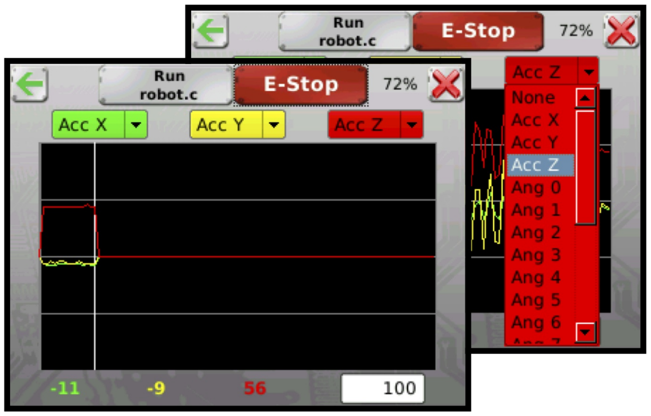


Fig. 6. *Two screen shots from the CBC displaying data in the oscilloscope mode*

installs the new firmware on the BoB. The process takes only a few seconds.

The software upgrade process for the motherboard uses hooks provided by *Chumby* in their boot scripts that allow external scripts to be loaded in place of the standard *Chumby* scripts via an external USB flash drive. Because of the large size of the upgrade files for the motherboard (the upgrade includes LINUX and the GCC tool chain) the upgrade is performed via flash drive rather than through a serial connection, which would take substantially longer.

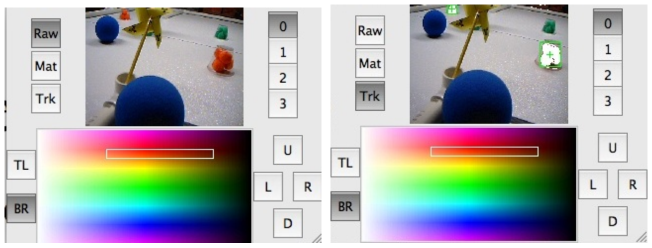


Fig. 7. *Color track information, with orange being tracked (right)*

A script placed on the internal psp partition will run ahead of the USB subsystem, and is loaded as the boot script for the CBC2 in place of the Chumby boot script. When the script runs, it searches for an external drive for which it can mount a FAT32 partition. If found, it does so and checks to see if it has a ”userhook0” file at root level in which case the upgrade process proceeds. Otherwise the script proceeds to a normal CBC2 boot.

D. *Color vision system*

The CBC2 contains two USB-A ports in the back. One of the expected uses of these ports is to connect to a USB webcam in order to perform visual color tracking. The CBC2 uses a modified 6282 camera driver pulling images off in QQGVA: 160×120 format.

The images are then processed, marking pixels that match any of the four user selected portions of HSV color space. The marked pixels are grouped into blobs, and statistics on each of the blobs (centroid, boundaries, size, axis orientation, etc) are computed. More details on the algorithm can be found in [13] and [4].

The vision system UI makes extensive use of the touch screen interface. Figure 7 shows the UI when the camera is looking at a scene. The buttons to the left of the image specify either the raw image; the image where the pixels that match the selected part of the HSV space are highlighted, or, as in Figure 7, the blobs that match have markings superimposed showing the centroid being tracked and the bounding box for that blob.

The buttons to the right of the image allow the user to select which of the four color channels they wish to view or adjust. The ten largest blobs for each of the four color channels are tracked simultaneously.

The lower half of the screen is used to adjust the portion of color space selected for the color channel being adjusted. On the left are buttons for selecting the top left and the bottom right corners of the selection box (the white rectangle that covers part of the red/orange color space in Figure 7). The four buttons to the right of the color space map are for moving the selected corner of the color space selection box. Extending the box to the right selects pixels that are in shadow, while moving the box to the right accepts more poorly saturated pixels. Up and down change the hue that is being selected. Exiting the screen saves the values to flash, so that the color selections are preserved through rebooting the CBC.

E. *The CBC2 User Library*

In addition to the standard C libraries, the KISS-C IDE automatically loads a user library to provide user access to the capabilities of the CBC. These include, but are not limited to:

- Virtual sensors in the form of soft controls on the CBC2 touch screen
- Integrated hardware sensors (touch and accelerometer)
- Vision system for web camera
- Digital I/O ports
- Fixed and floating (open collector) analog ports including functions for specialized sensors such as sonar
- DC motor ports with PID control using back EMF and PWM
- Servo motor ports

Insofar as possible, KISS-C library functions correspond to those provided in IC for the XBC. In particular, a console screen (Figure 3) implemented on the CBC2 provides as virtual sensors the programmable physical buttons present on the XBC for upward compatibility of code.

KISS-C library functions for digital I/O provide the ability to use the 8 digital ports on the CBC2 to either send or receive $0/+5V$ signals, enabling control of solid state relays among other possibilities. Since there are both fixed and floating analog ports the library functions for accessing analog ports work for a variety of analog devices such as common light sensors and the Sharp IR range finder [14].

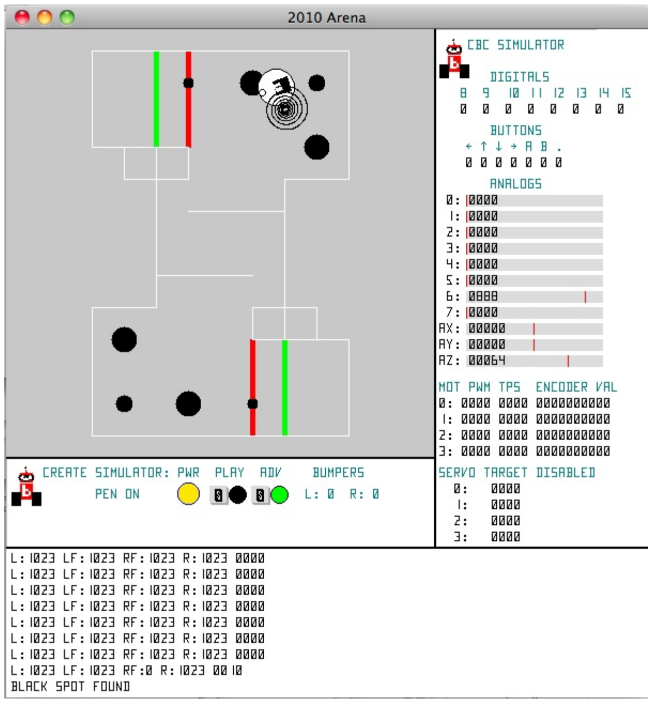
The KISS-C library includes a selection of functions for controlling DC motors. The majority of these are for taking advantage of the CBC2’s back-emf PID motor control capabilities, but also allow full access to the underlying PWM motor control.

KISS-C library functions specific to servo motors are used to enable/disable the CBC2’s servo ports, preset a servo, or move a servo to any position within approximately 180^o of travel.

Since the CBC2 is a good choice of controller for the iRobot Create module, the KISS-C library also has an extensive set of functions [15] to facilitate utilization of the Create open interface [16]. Additionally, the KISS-C IDE includes a companion simulator to the CBC2 simulator which implements the Create movement commands (Figure 8) or a differentially driven robot built with motors connected to motor ports 0 & 3. Touch sensors and reflection sensors are built into the robot simulations. `printf` statements output to the scrolling window at the bottom (which, like the CBC console, has nine lines of output). Code written and tested on the simulator can be used on the CBC2 without adjustment to operate a Create module.

V. SUMMARY & APPLICATION AREAS FOR THE CBC/KISS-C SYSTEM

The CBC/KISS-C system for robot control addresses a number of shortcomings of its predecessors. The CBC2 controller is ergonomically impressive and not just because of its well lighted touch screen. It fits comfortably in one’s



```
CBC SIMULATOR

    DIGITALS
 8  9  10 11 12 13 14 15
 0  0  0  0  0  0  0  0

    BUTTONS
 ← ↑ ↓ → A B .
 0 0 0 0 0 0 0

    ANALOGS
 0: 0000
 1: 0000
 2: 0000
 3: 0000
 4: 0000
 5: 0000
 6: 0888
 7: 0000
AX: 00000
AY: 00000
AZ: 00064

MOT PWM  TPS  ENCODER VAL
0:  0000 0000 0000000000
1:  0000 0000 0000000000
2:  0000 0000 0000000000
3:  0000 0000 0000000000

SERVO TARGET DISABLED
  0:   0000
  1:   0000
  2:   0000
  3:   0000
```

```
CREATE SIMULATOR: PWR  PLAY  ADV    BUMPERS
       PEN ON                       L: 0  R: 0
```

```
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:1023 R:1023 0000
L:1023 LF:1023 RF:0 R:1023 0010
BLACK SPOT FOUND
```

Fig. 8. *The combined Create and CBC2 simulator; the programming being simulated drives the robot in an expanding spiral path until a black mark is detected under the robot by one of its reflectance/cliff sensors. The user can turn off an on a pen attached to the center of the simulated robot – which here shows the spiral path taken by the robot in search of a black marking on the ground.*

hand, yet has multiple mounting holes on its bottom (Lego spacing). Its motor and sensor ports are logically arranged and are accessible without excess exposure. It's serial and USB ports are located so as to not become tangled with motor and sensor ports. With embedded Linux and reloadable firmware, it is a complete package, nicely complemented by the KISS-C IDE, which provides a rich library specific to the CBC, an integrated simulator, and everything one finds with a GCC compiler.

The CBC/KISS-C system is well-suited for a collegiate robotics lab, especially if paired with the iRobot Create module. It fits comfortably in the Create's cargo bay, in particular, and has a serial port voltage compatible with the Create. Since the CBC2 is already socketed for Wi-Fi, it would be an interesting project to implement a swarm of Creates operated by CBCs. It can also function as a multi channel data-logger, writing the data to the internal flash or an external memory stick.

The CBC2 was adopted for the 2010 Botball program, and so is being used by thousands of middle and high school students in hundreds of robots of their design (Figure 9). A future paper will report on their experiences, and our continued experiences with the CBC. We hope to find that the combination of ease of use while teaching and leveraging the power of standard OSes and programming languages will allow the CBC2 to have a more substantial positive impact on engineering education than those of its predecessors.

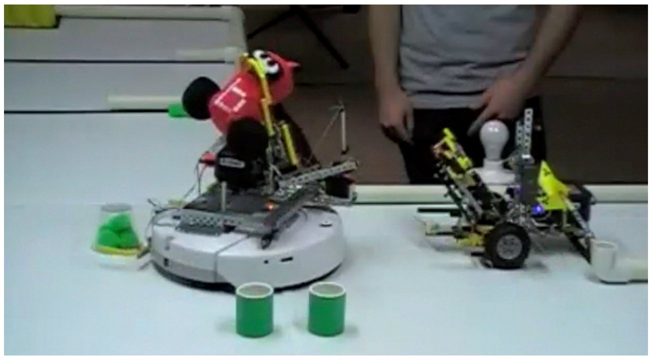


Fig. 9. *Two CBC2 controlled robots in a Botball regional tournament*

Acknowledgements

The authors wish to thank Mary Jungman, Henry Groover & bunnie at Chumby Industries for their technical support; Anne Wright & Randy Sargent for their help on the vision system and other areas; The students, teachers and instructors who used the CBC and provided testing data and user feedback; we also wish to thank the staff and management of KIPR for their support of the CBC project.

References

[1] 6.270 Organizers, "The history of 6.270 - mit's autonomous robot design competition," 2005. [Online]. Available: http://spacecats.mit.edu/~6.270/about/history.shtml

[2] F. Martin, "The handy board," 2002. [Online]. Available: http://www.handyboard.com/techdocs/

[3] Engineering Trends, "No growth in interest of us citizens for obtaining engineering degrees," Engineering Trends, Tech. Rep. 0907A, September 2007. [Online]. Available: http://www.engineeringtrends.org/IEE/0907A.php

[4] R. LeGrand, K. Machulis, D. P. Miller, R. Sargent, and A. Wright, "The XBC: a modern low-cost mobile robot controller," in *Proceedings of IROS 2005*. IEEE Press, August 2005.

[5] D. P. Miller, I. R. Nourbakhsh, and R. Siegwart, "Robots for education," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, 2008, ch. 55.

[6] C. Stein and D. Hartz, "The little robot tournament that could," in *2003 ASEE Annual Conference & Exposition: Staying in Tune with Engineering Education*. ASEE, June 2003.

[7] D. P. Miller and C. Winton, "Botball kit for teaching engineering computing," in *Proceedings of the ASEE National Conference*. ASEE, June 2004. [Online]. Available: http://www.kipr.org/papers/asee04-botballkit.pdf

[8] Chumby Industries. (2007) Complete schematics and layout for the chumby core unit with cross-references. [Online]. Available: http://files.chumby.com/hdwedocs/Rev37_release.pdf

[9] ——, "Chumby," 2010. [Online]. Available: http://www.chumby.com/pages/learn_overview

[10] S. Richards. (2006) Back-EMF. [Online]. Available: http://www.acroname.com/robotics/info/articles/back-emf/back-emf.html

[11] QT Software, "Qt - a cross-platform application and ui development framework," 2009. [Online]. Available: http://www.qtsoftware.com/

[12] Riverbank, "QScintilla," 2008. [Online]. Available: http://www.riverbankcomputing.com/software/qscintilla/

[13] Newton Research Labs, "Cognachrome vision system," 2001. [Online]. Available: http://www.newtonlabs.com/cognachrome/index.html

[14] J. C. Zufferey, "Sharp gp2d12," April 2001. [Online]. Available: http://dmtwww.epfl.ch/~jzuffere/SharpGP2D12_E.html

[15] D. P. Miller, "Quick and easy way to add vision to your irobot create or roomba," 2007. [Online]. Available: http://i-borg.engr.ou.edu/~dmiller/create/

[16] iRobot Corporation, "iRobot create open interface," 2007. [Online]. Available: http://www.irobot.com/filelibrary/create/Create%20Open%20Interface_v2.pdf