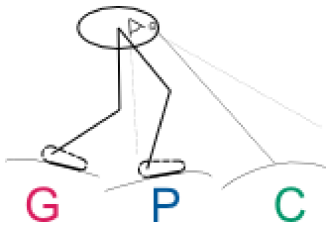
Northeastern University
College of Computer and Information Science




Home
Projects (Past)
News (Older)
Research Areas
People
Publications
Software
Courses
Seminar
Funding
Directions
Links
Wiki

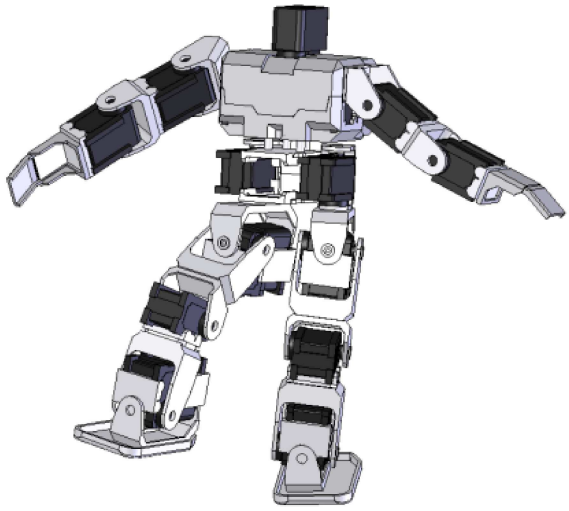
# GPC: Bioloid Java Remote Brain

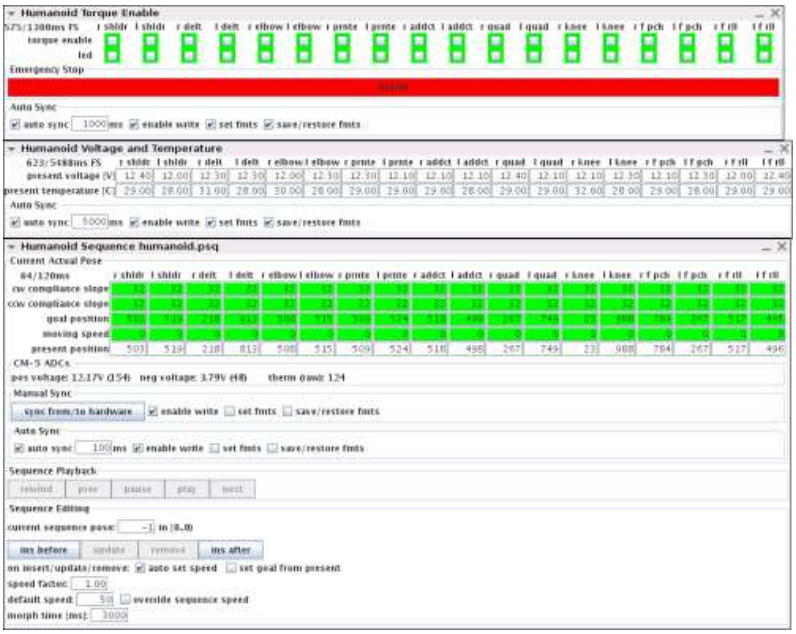
Page Contents



The Bioloid Remote Brain firmware and Java host library *BRBrain* replaces the factory-provided graphical programming system for the low-cost Robotis Bioloid robot construction system with a host-based Java environment. With BRBrain, Java code running on a host PC controls the Bioloid via a serial tether or a Bluetooth link.

We used BRBrain to implement a humanoid stair-stepping experiment.






## About BRBrain

BRBrain includes a Java host library and custom C firmware for the CM–5 controller. Together these two components implement remote brain functionality: sensor and actuator data is exchanged in (near) real time between the CM–5 and a host workstation, allowing high-level control code to run (and be debugged) directly on the workstation. The control code can be written directly in Java and can also optionally use an included JScheme shell for Scheme-language programming.

The factory-provided software for the Bioloid system is easy to use but limited in capability—its visual programming environment is only suited for the most simple behaviours. BRBrain totally replaces this programming environment with Java running on a host computer, with two trade-offs:

- Constant communication is required with the host. In practice it is relatively easy and inexpensive to use a wireless connection (e.g. bluetooth) so that the robot can still operate un-tethered.
- The maximum control bandwith and latency are limited by the communication channel to the host. In practice we typically achieve about 10Hz control update rate to all 18 servos in a humanoid.

The relative benefits of using BRBrain can be significant, since the host computer can be a workstation with many orders of magnitude more computational power and memory than the small 8-bit microcontroller in the CM–5, and it will also typically have "nice" things like a graphical display, disk, network, keyboard, and mouse.

Note that BRBrain does not allow you to run Java code directly on the robot's CM–5 controller. You run Java code on a host PC that remains in constant communication with the CM–5, e.g. using a standard RS232 serial cable or a a bluetooth link (details below).

The source code and precompiled binaries are provided below under the GNU GPL.

## Features and Architecture

This section gives a high-level overview of the BRBrain library. See the javadoc and the source for full details.

The custom CM–5 firmware exchanges communications packets over a serial link with a BRBrain object in a JVM on a host computer. It does not contain much functionality itself: the whole point is that it turns the CM–5 into a relay that allows Java code running on the host PC to read and write registers on the connected Dynamixels. Both AX–12 and AX-S1 are supported.

The nominal usage pattern for the host Java code is thus

1. create a BRBrain object, specifying communications parameters that will let it talk to a CM–5 running the custom

firmware

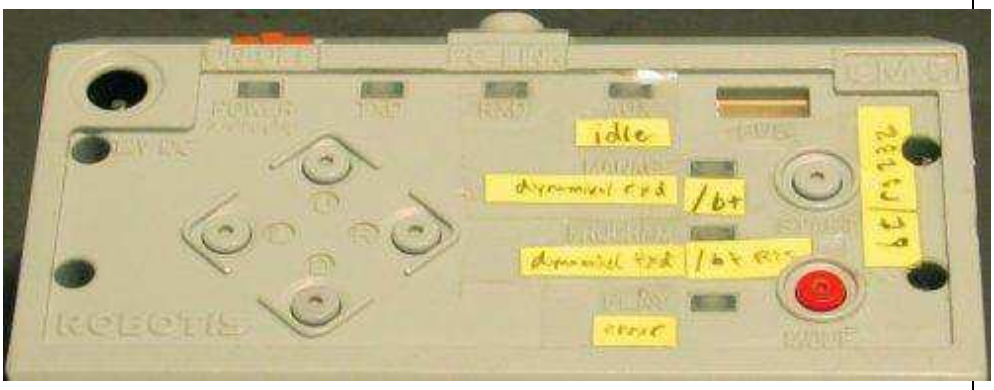
2. configure read and write *formats*, which define the blocks of Dynamixel registers that will be read and written

3. read and write blocks of register values; the BRBrain object communicates with the custom firmware on the CM–5, which in turn communicates with the Dynamixels, to transfer the data between the Dynamixels and the Java code.

The BRBrain class header javadoc includes example code.

The host can either connect to the RS–232 port on the CM–5 or to a Bluetooth module you can add to the CM–5. The CM–5 firmware can only communicate with at host PC on one of these two ports at a time; the the "START" button is re-purposed to toggle between them. The firmware boot-up default is configurable at build time (pre-built binaries are provided for both ports).

The custom firmware also re-purposes some of the LEDs on the CM–5, as shown in this image (click to enlarge):




The BRBrain library further includes code for representing a pose, which is the contents of a (configurable) set of registers on a (configurable) set of Dynamixels connected to the CM–5, as well as for an ordered set of poses. Each of these knows how to read and write itself to human-readable text files. In addition, generic GUI components are provided for poses and pose sequences. So a somewhat higher-level way to use the system is to write Java code to read and/or write poses or pose sequences to the hardware, possibly also providing GUIs for the user to view or edit them.

An executable REPL-style interactive JScheme interpreter is also included. This is the highest-level way to use the system. An example scheme script is provided which, when run in this interpreter, will bring up GUIs appropriate for a standard 18 servo Bioloid humanoid. You can then use the GUIs to

- manipulate the AX-S1
- enable or disable some/all servos
- display the current temperature, voltage and motion data from the servos

- write new motion data to the servos
- add the current pose of the humanoid to the current sequence
- edit, save, and load the current sequence
- play back the current sequence on the humanoid

There's a sreenshot at the top of the page (click to enlarge).

The full BRBrainShell JScheme API is defined in brbrain-shell-api.scm, with additional scheme sugar in brbrain-shell-extra.scm. In general, the JScheme api is a thin cover over the corresponding Java API.

You can run this interpreter by executing `java -jar BRBrain-newest.jar` or by using one of the provided convenience scripts such as `humanoid-rfcomm.sh` *address*. In any case, you interact with the interpreter via the console, and it may be more convenient to use a console with specific support for scheme interaction, for example, Emacs Quack (tip: open brbrain-shell-api.scm and brbrain-shell-extra.scm in the same Emacs session and you'll get M-/ name completion for the BRBrain API).

## Known Bugs and Limitations

- The BRBrain CM–5 firmware does not (yet) support charging the battery in the CM–5. A work-around is to temporarily re-flash the standard firmware from Robotis, charge the battery, and then re-flash the BRBrain custom firmware. In practice we usually run the Bioloid with its power tether attached, which largely avoids having to charge the battery.

- The bash scripts included for configuring the BlueSMiRF module were developed and tested for an earlier version of that product based on a chipset from Mitsumi. Sparkfun has has since revised the product and replaced the Mitsumi chipset with one from Roving Networks with similar functionality but a potentially incompatible command set. The host scripts have not been tested with the new chipset and may need revision to support it.

- Actual real-time performance depends on the abilities of your host operating system, JVM, and communications link. We typically use a standard Sun JVM on Ubuntu Linux over a Bluetooth serial link, which gives us a soft-real-time implementation capable of about 10Hz control update rate to all 18 servos of a standard Bioloid humanoid.

- The file I/O communications implementation requires the serial port parameters (115.2kbps, 8N1, no flow control, no special characters) to be set externally.

## Future Work

- Support "toss" mode with standard CM–5 firmware.
- Support zigbee as an alternative to bluetooth with CM5.

- Support USB2Dynamixel instead of CM–5.

## Requirements and Installation

The BRBrain host code has been tested under 1.5+ JREs from Sun on Linux. Other JREs may work.

The BRBrain host Java code can establish a connection to the CM–5 running the BRBrain firmware either by doing file I/O directly on a platform-dependent device file representing a serial port connected to the CM–5, or it can use the RXTX Java serial port library for platform-independent serial port access. In the latter case you'll need to install RXTX separately. You'll also need to install RXTX if you need to (re)build the BRBrain host Java code.

The first thing you'll need to do is flash the custom firmware onto your CM–5. If you've added the custom bluetooth interface as described below, and you want it to be used by default every time the firmware boots, use this firmware. Otherwise, use this firmware to default to RS–232 communication with the host.

Flashing the custom firmware on the CM–5 will replace whatever firmware it currently has. Make sure you have a copy of the previous firmware (which will be the Robotis standard firmware unless you've already re-flashed) so that you can restore it if necessary. As noted above, the custom firmware does not yet include battery charging, so you'll need to flash the standard firmware temporarily every time you want to charge the battery. Robotis provides the standard firmware on the CD-ROM that ships with the Bioloid kit, search for a file named Bioloid_VerNNN.bin, where NNN is a number. On my CD-ROM the file is `Examples/ROM file/Bioloid_Ver114.bin`. You should also be able to find the most recent CM–5 firmware on the Robotis web site.

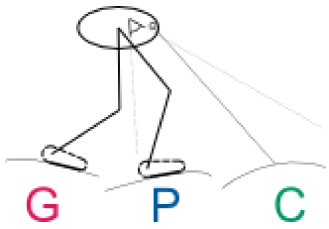
The file you will be sending is not a hex file, as is commonly used to program microcontrollers, rather it is the verbatim binary contents to be flashed. If you have a hex file you can turn it into a binary file like this: `avr-objcopy -O binary -I ihex foo.hex foo.bin`.

The firmware is flashed by interacting with the CM–5 bootloader (a special part of the CM–5 firmware that is always resident). There are a few different ways to use the bootloader:

- The Robotis "Behavior Control Programmer" can be used to update the CM–5 firmware according to its instructions; presumably this just automates the manual process described next.

- You can use a serial terminal program (such as c-kermit or minicom) to interact directly with the bootloader. The communication parameters are 57600 8N1. Send # characters continuously (i.e. hold down shift–3 on your keyboard) while resetting the CM–5. You should see

Northeastern University
College of Computer and Information Science




Home
Projects (Past)
News (Older)
Research Areas
People
Publications
Software
Courses
Seminar
Funding
Directions
Links
Wiki

```
SYSTEM O.K. (CM5 Boot loader V1.31)
```

or similar in your terminal window, and then type

```
help
```

for a menu. The "Load" function is used to flash a new program onto the CM–5, normally to address 0. Initiate the load in the bootloader by typing

```
load
```

and hitting enter. The bootloader should respond with something like

```
Write Address : 00000000
Ready..
```

which means it is now waiting for your terminal to send the data. For example, with minicom you can just switch to another terminal (you are on Linux, right?) at this point and issue the command

```
cat foo.bin > /dev/ttySN
```

where foo.bin is the file to flash and N is the serial port number where the CM–5 is connected. You don't need to exit minicom while you do this. When the transfer is complete the CM–5 should respond with something like this

```
Error
Rewriting:0X0002
Size:0X00009FFF  Checksum:51-00
```

As long as the size is correct it appears the "Error" message can be ignored. Perhaps it is expecting a checksum which never arrives.

- The BRBrain library itself has code that automates the process of interacting with the bootloader. It prompts you when to press each required CM–5 button as you go through the process, and it does a readback-verify after the re-flash is complete. You can access this code through [BRBrainShell(#shell). For example, on Linux, invoke the shell like this

```
java -jar BRBrain-newest.jar /dev/ttySN
```

where N is the serial port number where the CM–5 is connected. You should get a startup message and then a scheme prompt

```
brbrain>
```

Now issue the command

```
brbrain> (flash-cm5 "foo.bin")
```

where foo.bin is the file to flash, and follow the instructions.

## Download

BRBrain is released in source and binary form under the GNU GPL.

The distribution jar comes with everything rolled into one: java source, C source, scheme source, html (M4 and markdown) source, the makefiles and related stuff, plus precompiled versions of everything (java class files, firmware binaries compiled from C code, html, and javadoc). Further, it includes the pure java dependencies of BRBrain, which are at the time of this writing just a modified version of JScheme. A lighter-weight jar is also available which does not include the java dependencies. You do not need JScheme unless you are actually running BRBrainShell.

Two versions of the custom CM–5 firmware are supplied. One is configured to initially listen for communication from the host PC on the CM–5 built-in RS232 serial port. The other is configured to initially listen on the bluetooth interface which you can add to your CM–5 according to the instructions below.

## Build Requirements

The BRBrain build has been tested on Linux. Other platforms may work. You should only need to (re)build the code if you have made a change; it is distributed with pre-built binaries.

The build system is based on the Super-Ninja Makefile. You'll need

- a 1.5+ JDK with the binaries on your path
- (GNU) make and m4
- avr-gcc and avrlibc to (re)build the firmware, if desired
- the RXTX library.

The build depends on the RXTX library for portable access to serial ports. This is not a pure java dependency, RXTX contains both Java and native code. You'll need to manually install it before the build will succeed.

Note that the RXTX library need *not* be present at run-time unless you actually use it. For example, if you choose the direct file I/O communications option, the pre-compiled host library should not complain if RXTX is not available.

You'll probably want to create a directory into which to unpack the jar (`jar xvf BRBrain-newest.jar`). The code for BRBrain proper will

Northeastern University
College of Computer and Information Science

G P C

Home
Projects (Past)
News (Older)
Research Areas
People
Publications
Software
Courses
Seminar
Funding
Directions
Links
Wiki

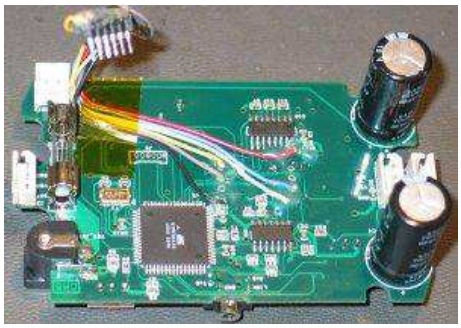
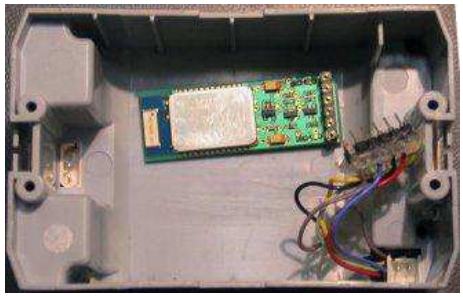
be under the unpacked directory `brbrain/`; other unpacked directories contain the bundled pure-Java dependencies (like jscheme).

From within the `brbrain/` directory,

- `make makefiles` generates makefiles in any subpackages
- `make project-clean` removes most precompiled stuff
- `make project-realclean` removes all precompiled stuff
- `make project` (re)builds the binary files for the project
- `make` (re)builds the binary files in the current directory
- `make project-javadoc` (re)builds the project documentation.

## Bluetooth

As it ships, the CM–5 circuit board contains pads for the connection of a Zigbee module. We prefer to use SparkFun BlueSMiRF modules. Here's how we installed one inside the CM–5 instead of a Zigbee module.

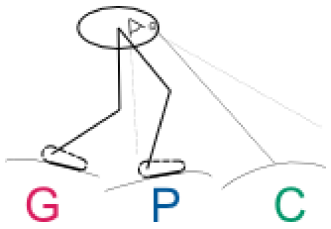




| BlueSMiRF Pin | BlueSMiRF Signal | CM–5 Pad | Wire Color |
|---|---|---|---|
| 1 | CTS-I | Z_RST | yellow |
| 2 | PWR | U10 pin 16 | red |
| 3 | GND | GND | black |
| 4 | TX-O | Z_TX | blue |
| 5 | RX-I | Z_RX | green |
| 6 | RTS-O | Z_LED | gray |

## Disclaimer

THIS INFORMATION AND/OR SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR

Northeastern University
College of Computer and Information Science




SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS INFORMATION AND/OR SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

 | last updated Sat Nov 3 12:30:54 EDT 2012 | site home