Using Robotis Bioloid Systems for Educational Robotics

C. N. Thai and M. Paulishen
University of Georgia
Biological and Agricultural Engineering Department
Athens GA 30602-4435
E-mail: thai@engr.uga.edu
Web site: http://www.engr.uga.edu/~mvteachr

***Abstract*— This is a report on our evaluation of Robotis' Bioloid robotic systems for their use in a senior-level engineering robotics course. We had found that Robotis had provided robust embedded C and LabView libraries that we were able to intermix, adapt and extend for our needs in sensor and actuator interfacing, especially in ZigBee applications to create a mobile wireless sensor network whereas the PC acted as the base station to 3 carbots sending back their NIR sensor data and the number of node hops needed for a data packet to get back to base. In the area of harmonizing multiple controllers on a single robotic platform, Robotis did not have tools readily usable for us, but we managed to develop a basic motion capture tool to work on a quadruped robot platform in a master-slave controllers configuration.**

I. INTRODUCTION

Currently in the U.S.A. and for undergraduate education in Robotics, we believe that Worcester Polytechnic Institute is the only university that currently offers a stand-alone B.S. degree in Robotics Engineering [1], while other universities such as Rose-Hulman Institute of Technology adopted the approach of a multidisciplinary robotics minor for students majoring in Computer Science, Electrical, Computer, Mechanical or Software Engineering [2]. With the recent approval by the University System of Georgia's Board of Regents for new B.S. degrees in Mechanical Engineering and Electrical and Electronics Engineering to come on line in Fall 2013 at the University of Georgia (UGA), and when combined with the existing Computer Systems Engineering B.S. degree, an emphasis area in Robotics is looking very viable to be developed for undergraduate students enrolled in the above three degrees at UGA. As an initial step, we are looking at using robotics as an instructional approach to integrate hardware, software and communication technologies at a senior-year level course whereas students would already have taken courses on Microcontrollers, Sensors and Transducers, Kinematics, Dynamics and Machine Design. The goal is to provide students with a basic practicum in Embedded Robotics wherein the students will learn about the programming of embedded controllers, the actuation of servo motors, the interfacing of sensors, inter-computer serial communications, and the control of autonomous as well as remotely piloted systems. After extensive research into commercial robotics systems from the cost point of view as well as from the ability to expand hardware and software sophistications into future graduate robotics courses, we opted to go with the Bioloid systems from Robotis (http://www.robotis.com). The most attractive feature of the Bioloid systems is their potential for link-based locomotion, allowing us to go beyond wheel-based systems [3]. Robotis systems allowed students to start programming with a proprietary but free high-level integrated development environment called RoboPlus, but recently Robotis had also provided interfaces to other popular programming tools such as MATLAB and LabView as well as lower-level C programming libraries at its support web site (http://support.robotis.com/en/).

The objective of our paper was to evaluate the suitability of the LabView and C (embedded side and PC side) programming tools beyond the examples that were provided by Robotis, in particular for multi-controller communication and control applications as wireless robot to robot communications are becoming important issues to consider in robotics [4][5].

II. BIOLOID SYSTEMS DESCRIPTION

Robotis offers several Bioloid systems based on the Atmel AVR microcontroller currently packaged in 3 versions (see Fig. 1):

1. The Comprehensive kit uses the CM-5 controller which was available commercially in 2005. It can interface with the actuator module AX-12+ and the integrated sensor module AX-S1 (NIR and sound) which are connected in a daisy chain fashion using TTL serial protocols rated at 1 Mbps. It can also communicate via RS-232 and ZigBee.
2. The Premium kit uses the CM-510 controller which was made available in mid 2009. In addition to the CM-5 capabilities with the

AX-12+ and AX-S1, the CM-510 can also interface with user-created sensors using memory-mapped I/O ports. It also has RS-232 and ZigBee communications capabilities.

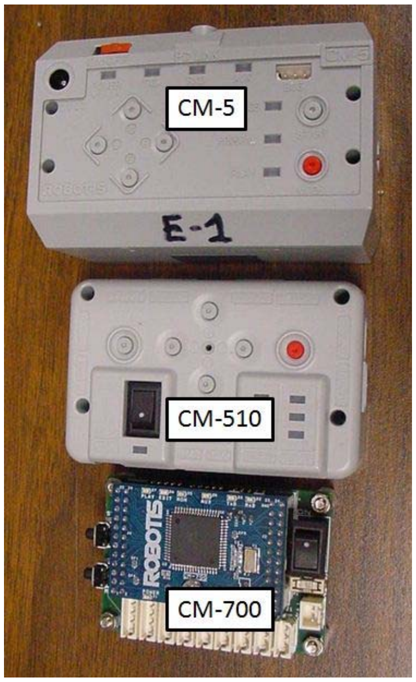


Figure 1. Available Bioloid controllers CM-5, CM-510 and CM-700.

3. The CM-700 became available in mid 2010 as a bare-bone controller having all the capabilities of the CM-5 and CM-510, and also RS-485 interfaces to the more advanced servo motors from Robotis series RX and EX.

For communications between a PC and the various CM-5/510/700 controllers, there are a variety of options (see Fig. 2):

1. Plain RS-232 9-pin cable between PC COM ports to a mini-jack port on the CM-5 or CM-510.
2. For newer PCs with only USB ports, ones can use the USB2Dynamixel module to connect the above 9-pin cable to an available USB port. The USB2Dynamixel module also allows ZigBee communications between the PC and various CM-XXX controllers when used with the Zig2Serial module and a Zig-100 daughter board.
3. The CM-700 can only use the USB-based LN-101 module for either program development tasks from the PC or during run-time uses the Zig-110 for ZigBee communication between itself and the PC or with other CM controllers, but not both at the same time as only one physical port is available for external communications. `

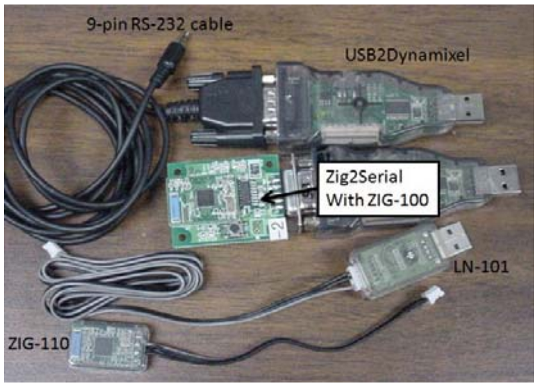


Figure 2. Communication options between PC and various CM-5/510/700 controllers.

Regarding software programming tools on the PC side, the student can start from a beginner IDE called RoboPlus Task and later go to direct API programming using Linux (gcc) or MS Windows (Visual Basic, C++, C#, LabView and MATLAB).

The CM-5/510/700 controllers are based on the Atmel AVR microcontroller at 16 MHz, with either 128 KB or 256 KB of Flash memory. The user-accessible memory area is divided into 3 independent but cooperating sections:

1. The main user logic resides in the TASK section which has standard features such as “main” and other user-defined functions. Familiar selection and repetition structures are available, but no parameter arrays can be defined by the user currently. A special function named CALLBACK can also be defined here but only once.
2. The CALLBACK section is executed every 7.8 ms which is also the refresh time period for all servo motors. Limited commands are allowed in the CALLBACK section to prevent collisions with the other commands from the TASK section.
3. The MOTION section contains the definitions of various static “poses” that the robot can take. Each pose is essentially a data structure representing coordinated absolute positions of the relevant servo motors used to build a given robot. These static poses can be further modified by applying JOINT OFFSETs which can be computed during run-time from user-defined algorithms responding to changes in selected servos.

However these RoboPlus tools will not be available when Embedded C applications are used as the original Bioloid firmware is effectively overwritten by the Embedded C application.

III. Evaluation of Embedded C Applications

We tested three C/C++ IDEs for their usage with the CM-510 and CM-700 controllers: Microsoft's Visual C++ Express 2010, WinAVR and AVRStudio. All three use the "avr-gcc" and "make" tools; however, AVRStudio fully integrates the makefile into the configuration settings of the GUI so no text based modifications are required. Microsoft's Visual C++ has no ability to compile code for the ATmega2561 controller inside the CM-510/700 and requires the installation of WinAVR in order to use the "avr-gcc" and "make" tools contained therein. Visual C++ also requires the various commands for using the makefile to be re-entered each time when creating a new project and cannot understand the syntax of makefiles (comments vs. commands). Programmer's Notepad (PN) is included with WinAVR and is more forgiving of project creation/modification than Visual C++ and can understand the syntax of makefiles allowing them to be more easily edited (text color differentiates between comments, commands, values, etc.). PN does not have a built-in makefile editor; however, while building all example projects from Robotis within both Visual C++ and Programmer's Notepad, an easily edited makefile template was created. This makefile has the most likely edited options at the top and should require only changing the target name to match that of the project. AVRStudio is created by Atmel and it includes "avr-gcc" as a precompiled plug-in, a makefile graphical editor, and a real time software debugger (should one have an Atmel compatible debugger/programming device). From our overall experiences, we plan to use AVRStudio for our students.

There are 3 main issues to consider when creating C projects for the Bioloid systems:

1. CM-XXX controller specific I/O ports and interrupts for sensors and such things as buttons, buzzer, microphone, LEDs.
2. Control of the Dynamixel devices such as servo AX-12+ and integrated sound/NIR sensor AX-S1.
3. Whether USB/RS-232 or ZigBee communication is used.

As the CM-700 does not have sound capabilities, Robotis did not provide any "sound" example for the CM-700 on its support web site, thus our first exercise was to interface an AX-S1 to it and tried to access its buzzer and microphone facilities. The solution was straightforward:

- Include "dynamixel.h" to access dynamixel functions.
- Use either write_byte or read_byte functions depending on the user needs:

  dxl_write_byte(ID, ADDRESS, DATA) or

  dxl_read_byte(ID, ADDRESS, DATA).

  Where ID = 100 for the AX-S1;

  ADDRESS = 40 for Buzzer Index (i.e. music note type);

  ADDRESS = 41 for Buzzer Time (i.e. music duration);

  ADDRESS = 35 for Sound Data;

  ADDRESS = 37 for Count of Detected Sound Claps;

  DATA = appropriate value according to AX-S1 control table [0-255].

In the second exercise, we combine dynamixel functions, LED functions and A/D port acquisition functions to create a more comprehensive embedded C demo program that had 5 modes of operations selected via sequential pushing of the START button:

1. Mode 1 (default mode at start of program) counts in binary with the PLAY, EDIT, MON, and AUX LEDs as output and incrementing with each sound detect count of the microphone on the AX-S1.
2. Mode 2 lights the PLAY and EDIT LEDs while changing the Buzzer Index value of the AX-S1 as a function of the IR output from ADC Port 1 while maintaining the last Buzzer Time (default is 0.3s).
3. Mode 3 lights the PLAY and MON LEDs and modifies the Buzzer Time of the AX-S1 as a function of the IR output from the ADC Port 1 while maintaining the last Buzzer Index value.
4. Mode 4 lights only the EDIT LED and controls the speed of SERVO 1 (in position control mode) which oscillates between two positions (default: 20 and 1000). The speed of SERVO 1 is a function of the IR data from the center IR sensor of the AX-S1.
5. Mode 5 lights only the MON LED and controls the speed and direction of rotation of SERVO 2 in continuous rotation mode. The speed and direction of SERVO 2 are dependent on the values of the left and right IR sensors of the AX-S1. When the value of the left sensor is greater than the right, SERVO 2 rotates counter-clockwise with a speed equal to (left-right)*4. When the value of the right sensor is greater than the left, SERVO 2 rotates clockwise with a speed equal to (right-left)*4.

In short, we had found the embedded C libraries from Robotis to be effective and useful.

IV. Evaluation of LabView Applications

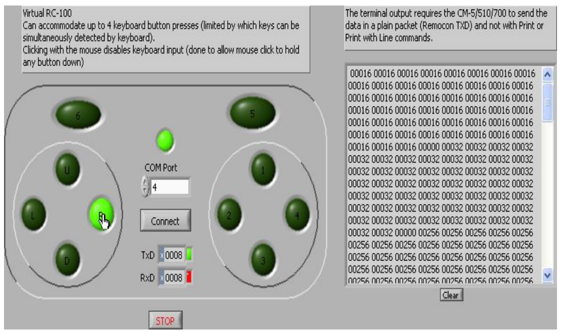
Robotis only provided two example LabView VIs related to reading and writing to the AX-12+ servos, but had provided web instructions on how to import their "dynamixel.dll". Thus we used the same procedure to import their "zigbee.dll" (and "zigbee.h") into LabView 2010 and surprisingly it worked well, thus we managed to create two new VIs: a virtual wireless remote controller running on the PC and a mobile wireless sensor network with the PC acting as a base station collecting data from 3 carbots.

A. *Virtual RC-100 Remote Controller*

Figure 3. Physical RC-100.

The actual RC-100 is the physical remote controller for the CM-5/510/700 controllers using one-to-one ZigBee protocols (see Fig. 3). The virtual RC-100 ran inside LabView on the PC and relied on the USB2Dynamixel/Zig2Serial/ZIG-100 combination module, as shown in Fig. 2, for its ZigBee capabilities. Fig. 4 is a snapshot of its front panel.

Once the user entered the correct COM port and clicked CONNECT, and when actual communication was established the ten buttons could be used to send commands to the actual CM-XXX controller in the same way as the physical RC-100. The Virtual RC-100 additionally could receive and display ZigBee packets sent back from the CM-XXX controllers as shown in the terminal output window on the right side of the front panel.



Virtual RC-100
Can accommodate up to 4 keyboard button presses (limited by which keys can be simultaneously detected by keyboard).
Clicking with the mouse disables keyboard input (done to allow mouse click to hold any button down)

The terminal output requires the CM-5/510/700 to send the data in a plain packet (Remocon TXD) and not with Print or Print with Line commands.

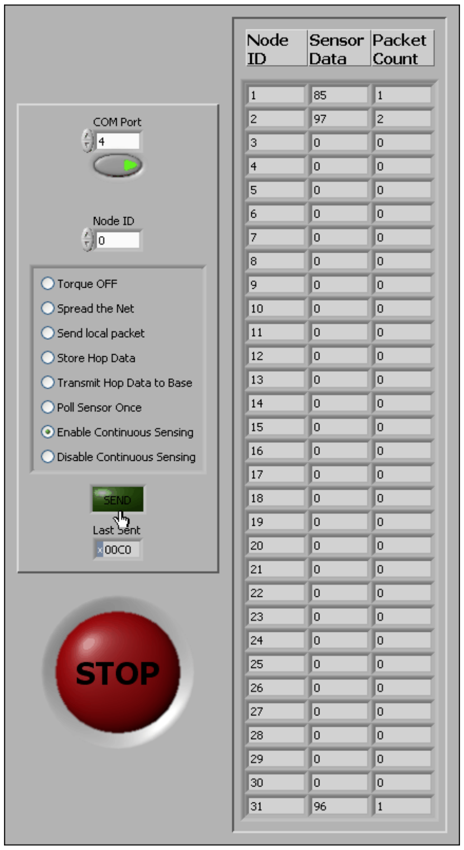
```
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00016 00016 00016 00016 00016
00016 00016 00016 00000 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00032 00032 00032 00032 00032 00032
00032 00032 00000 00256 00256 00256 00256 00256
00256 00256 00256 00256 00256 00256 00256 00256
00256 00256 00256 00256 00256 00256 00256 00256
00256 00256 00256 00256 00256 00256 00256 00256
00256 00256 00256 00256 00256 00256 00256 00256
```

COM Port 4
Connect
TxD 0008
RxD 0008
Clear
STOP

Figure 4. Virtual RC-100.

B. *Mobile Wireless Sensor Network*

Our goal was to create a ZigBee based mobile wireless sensor network whereas students can practice one-to-one and broadcast techniques (all ZigBee hardware actually set to broadcast mode). The PC would act a base station (see VI front panel in Fig. 5) which was designed to connect up to 31 carbots acting as mobile sensor nodes but so far we had only tested this VI with 3 carbots. Robotis recommended only a very modest baud rate of 57,600 bps for its ZigBee hardware but this was good enough for our instructional purposes.



COM Port 4

Node ID 0

( ) Torque OFF
( ) Spread the Net
( ) Send local packet
( ) Store Hop Data
( ) Transmit Hop Data to Base
( ) Poll Sensor Once
[✓ Enable Continuous Sensing]
( ) Disable Continuous Sensing

SEND

Last Sent 00C0

STOP

Node ID	Sensor Data	Packet Count
1	85	1
2	97	2
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0	0
17	0	0
18	0	0
19	0	0
20	0	0
21	0	0
22	0	0
23	0	0
24	0	0
25	0	0
26	0	0
27	0	0
28	0	0
29	0	0
30	0	0
31	96	1

Figure 5. Front panel of Mobile Sensor Net.

To operate this VI, the user entered the correct COM port and clicked on the CONNECT button. Once actual communications got established, the user would enter a node ID (1-31 for specific individual carbot or 0 for reaching all carbots). Next the user would click on a wanted command among the list of 8 available radio buttons on the mid-left panel and click the SEND button. For example. "Torque OFF" (command-1) would stop the carbot(s), "Spread the Net" (command-2) would send the carbot(s) outward from the base, etc… The STOP button of course terminated all processes and reset all buttons. The 3-column panel on the right displayed information within the latest messages sent by each carbot and received by the base station where:

- "Node ID" was the carbot ID where the message came from.
- "Sensor Data" was the reading [0-255] from the center NIR sensor of the AX-S1 sensor mounted on this particular carbot.
- "Packet Count" was the number of hops that this message had to make from that particular carbot (via the other carbots when needed) to the base station. For example, in Fig. 5, carbot 2 happened to be 1 hop further away from the base station than carbot 1 or carbot 31.

Robotis allows 16 bits to be used for each actual message encapsulated in a ZigBee packet of 6 bytes total. Our chosen design of these 16 bits is described below:

- The highest 3 bits (15 to 13) represented the Packet Count, i.e. the number of times the message had been relayed by another node (equal to 0 when sent from originating node).
- The middle 5 bits (12 to 8) represented the Packet ID, i.e. the Node ID of the sending node. If Packet ID = 0, then this packet was a command packet from the base station. If its value was between 1 and 31, then it was an ordinary data packet (i.e. NIR sensor data).
- The lowest 8 bits (7 to 0) represented the Data Byte. If this was an ordinary data packet, it has a value between 0 and 255 as provided by the particular AX-S1 sensor. If this was a command packet then:
  - Bits 7 through 5 represented the command number (1 to 8 as shown in Fig. 5 - to be defined in more details in a later section).
  - And bits 4 through 0 represented the Node ID of the intended receiving node. If zero, then this command (if allowed) is to be applied to all nodes.

As previously shown in Fig. 5, the base station could send out 8 types of command:

- Command-1, "Torque OFF", turned off all servos of selected carbot(s) via their Node ID.
- Command-2, "Spread the Net", spread out selected carbot(s) further away from the base station. Selected carbot(s) would move forward ~4s then backward for ~4s unless it received another new packet containing Command-1. If a new Command-2 packet was received during the forward motion, the carbot(s) reset the ~4s timer to continue moving forward; however if this new Command-2 was received during the backward motion, the carbot stopped in its tracks. Command-2 was designed to keep the carbot moving towards the edges of the ZigBee range by exceeding the range and then backtracking slightly until within range again of the base station or of another carbot.
- Command-3, "Send Local (i.e. **local**ization) Packet", ordered a node to wait ~15ms before it transmitted a new packet for Command-4.
- Command-4, "Store Hop Data", ordered a node to store the Packet Count (bits 15 to 13) and Node ID of the sending node (bits 4 to 0). This provided the relative location (in node hops) between a sending node and a receiving node.
- Command-5, "Transmit Hop Data to Base", ordered a node to send its stored location data to the base station.
- Command-6, "Poll Sensor Once", ordered a node to poll its NIR sensor and to send out this data only once to the base station.
- Command-7, "Enable Continuous Sensing", ordered a particular node or all nodes to enter an automatic sensor transmission mode. Affected node(s) would transmit their sensor data continuously without being polled by Command-5 type and these nodes would create and maintain a queue to time their transmissions to prevent conflicts.
- Command-8, "Disable Continuous Sensing", ordered a node or nodes to exit the automatic sensor transmission mode.

Lastly, each carbot ran a specific TASK program that self-assigned its Node ID and performed other tasks such as receiving, decoding, shaping and sending ZigBee packets and also controlling its servo motors as needed.

So far we had tested this system with only 3 carbots, and the basic process was for the base station to sequentially send out Commands 3, 4 and 5 to locate nodes relative to each other and to the base. Currently this VI could only display the last received or relayed information with the highest packet count. As the carbots spread out from the base station, as expected our simple design of the message relay system experienced a large drop in data rate and increasing lost packets, but we believed that we had achieved our goal of demonstrating to the students the complexity of wireless communications and most importantly that ones can use the Bioloid robotic systems to start learning about wireless sensor networks also.

In short, we also had found the LabView tools from Robotis to be effective and useful and we had much expanded their applications into the area of wireless sensor networks.

V. Multi-Controller Coordination and Robot Motion Capture

In this project we wanted to explore the control and synchronization issues arising from having multiple controllers on the same physical robot with link-based locomotion. The chosen twin-GERWALK design (see Fig. 6) was inspired from Boston Dynamics' Big Dog.

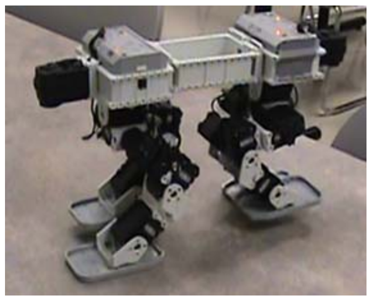


Figure 6. Twin-GERWALK robot platform.

Because of the inherent symmetry in this design, we readily arrived at a conceptual solution whereas the user would use the RC-100 to remotely control the front GERWALK acting as Master which would then send the appropriate commands to the rear GERWALK acting as Slave so that both robot motions were harmonized. For example, if the Master went "forward and right", the Slave would have to go "backward and left". Although Robotis already provided a very powerful and easy to use Motion Editor (ME) tool to create such robot motion pages, we found that Robotis allowed only 1 instance of the ME tool to be running at any one time in Windows. In other words, only 1 CM controller could be worked on at any one time and we had 2 CM controllers that we had to design "harmonizing" motions for. Thus we were forced to design a new bare-bone motion capture tool that consisted of 2 components:

1. The first component was a C++ program running on the PC side acting as a data logger recording servo position and servo speed data captured by a given CM controller and sent to the PC via ZigBee. This data logger could handle up to 4 CM controllers (via 4 separate COM ports) at the same time using our own extension code to the standard Robotis ZigBee library (which can handle only 1 COM port), and with 1 CM controller already served via the ME tool, potentially we could work on a robot that required 5 CM controllers collaborating together.

2. The second component was a TASK program running on each CM controller that was not connected to the ME tool. This TASK program used the UP, DOWN, LEFT and RIGHT buttons to execute following scripted actions:

    a. Pressing the UP button would lock in the current robot pose and transmit its data (i.e. current positions of all servos involved) to the PC.

    b. Pressing the DOWN button would unlock all servos (i.e. Torque Off) and allowed the repositioning of the robot links being worked on.

    c. Pressing the LEFT button would set the robot into its initial pose (i.e. run Motion Page 1, a Ready position).

    d. Press the RIGHT button to receive a pose from the PC Data Logger. This will cause the PC Logger to prompt the user to enter wanted servo goal positions for the robot to shift into. If the PC Logger received faulty user input, it would send a failure packet to the specific CM controller to exit the process. If the PC Logger is not active and if the CM controller was waiting for a pose to finish, pressing the DOWN button would exit the process.

Using this bare-bone motion capture tool, specific gait solutions were generated for various motions such as going forward or backward, turning left or right, and going up stairs steps (see Fig. 7). Next using the ME tool, these gait solutions were manually re-entered and saved into one motion file that was used for both Master and Slave controllers of the twin-GERWALK platform.

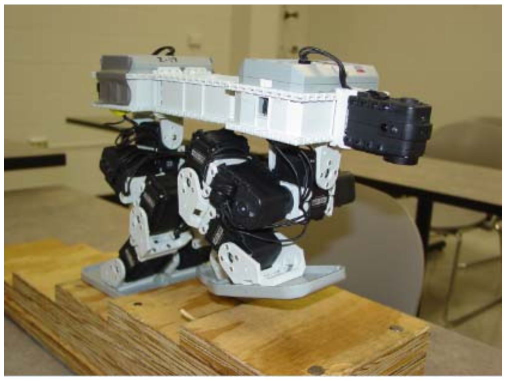


Figure 7. Twin-GERWALK going up stairs steps.

Next, specific TASK files were created for the Master and Slave controllers respectively with the goal of synchronizing their actions as a response to a remote control signal from an RC-100 as operated by a user:

- The Master received commands from the RC-100 and determined what motion both the Master and Slave should take, respectively. Next it transmitted the next motion to the Slave, it then executed its own motion. When it finished, it expected a confirmation of movement from the Slave as they should have both completed the motions simultaneously. If this confirmation packet was not received, it would try to reconfirm the motion (but no other corrective action is taken). As the Master and Slave were in close physical proximity (see Fig. 7), ZigBee functioned perfectly in this application.

- The Slave received its commands only from the Master and after completing a commanded motion, it transmitted a confirmation packet to the Master ensuring that it performed the correct motion.

For the reader, please contact the first author for further detailed source codes and visit this web site for video clips of our projects (http://www.robotis.com/xe/tips_en).

VI. Conclusions

We had found that the Bioloid systems, hardware and software wise, had been useful and effective in helping us design new instructional projects for our senior-level first course in embedded robotics. Robotis had provided robust embedded C and LabView tools that we were able to adapt and extend for our needs to create a mobile wireless sensor network whereas the PC acted as the base station to 3 carbots which could send back in real-time their NIR sensor data and the number of node hops needed for a data packet to get back to base.

In the area of harmonizing multiple controllers on a single robotic platform, Robotis did not have tools readily usable for us, but we managed to develop some rudimentary tools to capture robot motion, help us design appropriate gait solutions, and to get us going towards the exciting area of self-reconfigurable robots [6].

Although this work used exclusively Bioloid hardware and software products, our goal was to show that if we could add the "communications" dimension to a first-course engineering robotics course, students would have a good opportunity to go beyond the basic embedded-programming/sensor-interfacing issues and to get into a practical overview of multi-controllers design and robot motion capture issues.

REFERENCES

[1] T. Padir, M.A. Gennert, G. Fischer, W.R. Michalson, and E.C. Cobb, "Implementation of an undergraduate robotics engineering curriculum", Computers in Education Journal, vol. I, no. 3, pages 92-101, 2010.

[2] M. Boutell, C. Berry, D. Fisher and S. Chenoweth, "A multidisciplinary robotics minor", Computers in Education Journal, vol. I, no. 3, pages 102-111, 2010.

[3] B. Bishop, J. Esposito, and J. Piepmeier, "Moving without wheels: educational experiments in robot design and locomotion", Computers in Education Journal, vol. I, no. 3, pages 41-49, 2010.

[4] I. Mezei, V. Malbasa, and I. Stojmenovic, "Robot to robot: communication aspects of coordination in robot wireless networks", IEEE Robotics & Automation Magazine, vol. 17, no. 4, pages 63-69, 2010.

[5] N. Correll and A. Martinoli, "Multirobot inspection of industrial machinery: from distributed coverage algorithms to experiments with miniature robotic swarms", IEEE Robotics & Automation Magazine, vol. 16, no. 1, pages 103-112, 2010.

[6] K. Stoy, D. Brandt, and D.J. Christensen, Self-reconfigurable Robots:An Introduction. Cambridge, MS, USA, MIT Press, 2010, pp. 157-169.