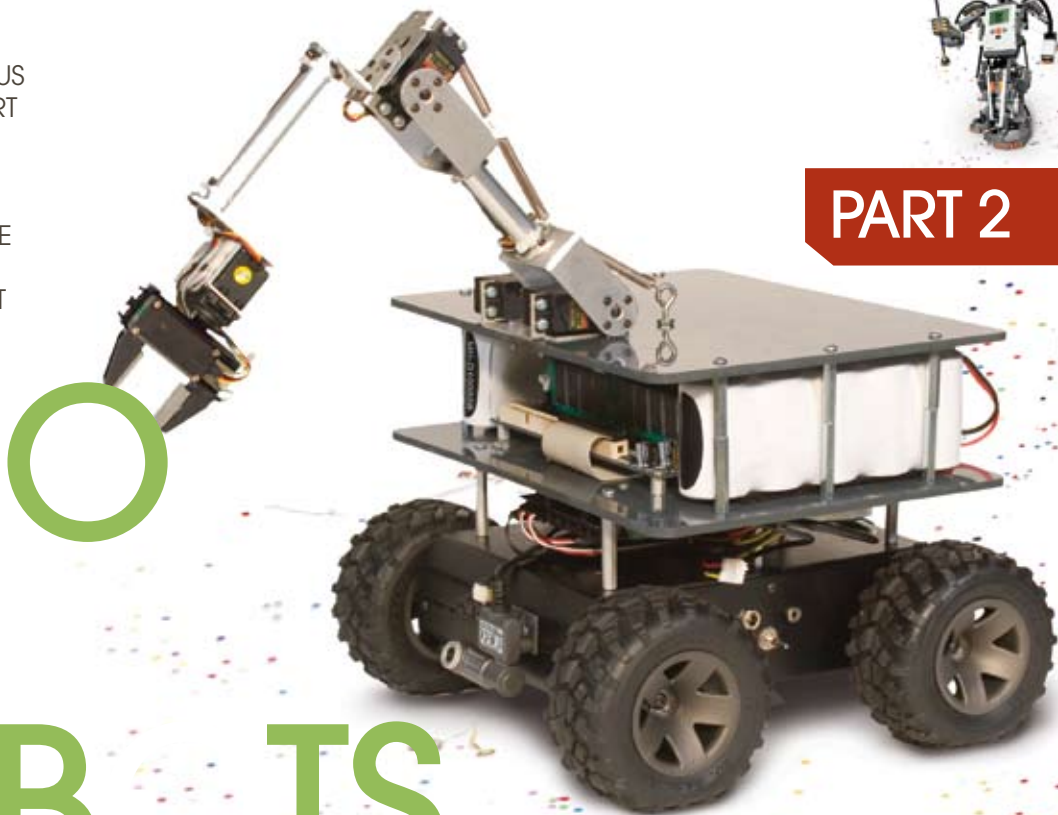


EVOLVING ROBOTIC-DEVELOPMENT PLATFORMS PRIMARILY FOCUS ON HOW TO JUMP-START DEVELOPERS, BUT THEY ALSO PROVIDE MUCH-NEEDED MECHANISMS TO REUSE THE SOFTWARE COMPONENTS FROM ONE ROBOTIC PROJECT TO ANOTHER.



## PART 2



# ROBOTS ON THE MARCH

BY ROBERT CRAVOTTA • TECHNICAL EDITOR

**T**he line between toys, games, and “real-world” applications is blurring. Technologies that originally found use in serious real-world applications continue to find their way into the much-larger-volume electronic-toy, gadget, and computer-based-game markets. Meanwhile, increasing opportunities exist for the flow of new engineering innovations from these entertainment devices into real-world applications. The consumer market currently accepts a significantly shorter production-support life cycle for many low-cost consumer-electronic products, such as entertainment devices, than the market accepts for higher priced products, such as automobiles, other vehicles, industrial and medical machinery, and large central-office equipment. The shorter support life cycle for these consumer-electronic products permits—or even demands—a higher degree of technology movement to identify what works.

To make industrial robots and semiautonomous systems, developers use technologies that are increasingly crossing the line from being industrial technolo-

gies to consumer and home-application technologies, such as electronic toys, gadgets, games, and other personal-entertainment devices. Unfortunately, like

PCs in the early '80s, software compatibility among today's robots still has a lot of room for improvement. The publicly available robotic-development platforms that have started emerging since last year are trying to address how to more quickly start robotic-design projects. They achieve this goal partly by providing a mechanism for developing software components the designers developed on one robot project and reusing them in another project. Since the publication of Part 1 of this hands-on project (**Reference 1**), I have become aware of two more publicly available robotic-development platforms—one from CoroWare and the other from Gostai (see **sidebar** “More platforms”).

Robotic-development platforms and their continued growth and evolution are essential components to enable scaling the complexity of today's and future projects to a manageable level that pre-

serves the productivity of designers. This hands-on project focuses mostly on the Lego NXT Mindstorms platform in conjunction with the National Instruments LabView environment. I also had some time to play with the Microsoft Robotics Studio.

## THE PROJECT

I based the list of hardware components for this project on a demonstration by Brady Duggan, a software engineer at National Instruments. Duggan demonstrated an unofficial reference design for an electronic “sheep dog.” The value of using a hardware configuration that someone had used in a similar project helped immensely with quickly getting up and running. The hardware setup consisted of a Texas Instruments TMS320VC33-based Speedy-33 DSP module from National Instruments that connected to a prototype board from HiTechnic. That board in turn connected to the Lego NXT controller that controls the drive motors for the simple platform of Lego pieces (**Figure 1**).

The Speedy-33 includes dual microphones approximately 5 in. apart, and the board supports sampling of the microphones at 48 kHz. LabView supports the same direct programming of the board, as do any of the many hardware components that LabView supports. The Speedy-33 board acted as the ears of the robot. Because I would need to learn much information about sound in a short time, I decided that the Speedy-33 would act only as a sensor and feed the data to the NXT. A follow-up iteration of the project will include implementing two-way communication between the two units so that the sound-sensing algorithm can incorporate information from the robot platform in discerning the sound signal’s location relative to the robot.

To simplify the complexity of this project, I chose an 880-Hz sound source that would remain stationary during a test run. I chose this frequency because experience with the reference algorithm had shown that the system had better success with higher frequency than with lower ones, such as 440 Hz. Looking for only one tone made it easier to use the DSP functions that the LabView DSP-module package includes. In essence, the algorithm cross-correlates the microphone signals with the target frequency

### AT A GLANCE

- ▶ A number of robotic-development platforms are available to designers.
- ▶ The development tools for robotics platforms are maturing but still have a way to go.
- ▶ Robotic-development environments allow designers to quickly iterate designs to test ideas.

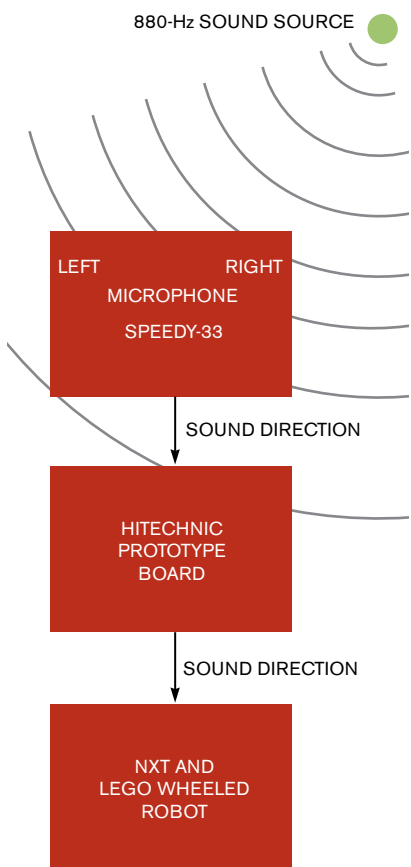
and determines the relative phase difference by comparing the sample position of the peak between each microphone. For future iterations of the project, the system should be able to detect an arbitrary predefined signal with the ultimate goal of detecting the phase difference from any arbitrary sound signal in a noisy environment using motion feedback from the robotic platform. To perform this project’s ultimate goal, the robotic platform must sense its inertial posi-

tioning so that, when the platform turns or moves, it can accurately convey that motion to the sensing algorithm. This feature requires the addition of gyro and acceleration sensors, such as those from HiTechnic, to the robotic platform.

The HiTechnic prototype board is basically a bridge between the Speedy-33 interface and the NXT interface so that you need create no new code on the NXT to interface the two components. The prototype board allows designers to build their own sensors and more easily interface them with the physical and logical interface that the NXT uses. The HiTechnic board presents the custom sensor to the NXT according to the NXT sensor protocol. For this project, I used the six digital ports to communicate to the NXT from the Speedy-33. The Speedy-33 came in a sealed container with ports for several of the supported peripherals. To use the six digital I/O ports, however, I had to remove the board from its case because there was no other way to access the digital I/O. This project required no direct programming of the HiTechnic prototype board, however.

In the interest of simplicity and time-saving, I implemented a one-way communication from the Speedy-33 to the NXT controller. I knew that building a two-way communication scheme could take time and that it would increase the need for troubleshooting sessions. The Speedy-33 would report the left/right direction of the sound whenever it detected the target sound. The NXT control program had to know whether the Speedy-33 had updated the entry, so I dedicated two of the six digital pins as counters and the other four pins to denote a position from left to right across 16 positions for the sound source with the other four pins. This approach allowed the Speedy-33 to send an update only when it heard the target sound, and it put the onus on a new sound-position update had occurred.

As with the Speedy-33, I used LabView to program the NXT. However, a subtle but important difference exists between programming for each of these targets. LabView does not formally support NXT as it does the normal hardware components in the LabView family. To build NXT code with LabView, you must use the programming constructs as NXT tool-kit add-ons. Even normal program constructs, such as loops and



**Figure 1** The hardware setup consists of a Speedy-33 DSP module from National Instruments, which connects to a prototype board from HiTechnic, which in turn connects to the Lego NXT controller.

comparisons, must come from the add-on tool kit rather than from the normal location. This limitation still allows you to open and pin down the NXT-specific tool-kit menus so they are easy to reach

without accidentally pulling up the other tools for other targets.

The NXT houses a 32-bit ARM ([www.arm.com](http://www.arm.com)) processor that provides plenty of processing capability for this system.

Because the Speedy-33 sensing algorithm would update the direction information only when it heard the target sound, the NXT could display status information or remain idle between direction updates.

## MORE PLATFORMS

Since the first part of this article appeared, I became aware of two additional robotic-development platforms—CoroBot from CoroWare and URBI (universal real-time-behavior interface) from Gostai. CoroBot integrates a four-wheel skid-steering platform with a color camera; IR distance sensors; and a 1.2-GHz PC-class processor running Windows XP, Xubuntu Linux, or both (Figure A). Designers can drill holes into the product's plastic top plate for permanent mounting, and it can accept various adhesives, such as those on Velcro strips, for temporary mounting. The openness of the system eases access to its various components but limits its use to indoor environments. It weighs 12 lbs and can accept payloads as large as 5 lbs.

Nine models of the CoroBot platform, starting at prices of \$2799, are available to developers. Software development for the platform is available through Microsoft Robotics Studio for models with preinstalled Windows XP or through Player for models with preinstalled Xubuntu Linux. The platform is available with a dual-boot option and an optional four-DOF (degrees-of-freedom) arm with a gripper sensor. Models with the arm have 24 available servo ports, and armless models have 30 available servo ports. No C or C++ libraries currently support the platform, but the company says that it is reviewing PlusPack for Microsoft Robotics Studio to support future development.

Gostai is positioning its product, the URBI scripted-interface language, as a universal robotic platform for software modules. It works over a client/server architecture to remotely control a robot or any complex system.

URBI presents a universal mechanism to control a robot, add functions by plugging in software components, and develop fully interactive and complex robotic applications in a portable way. The platform works with various robotic systems; operating systems; and programming languages, such as C++, Java, and Matlab.

Gostai based the object-oriented URBI on a prototype approach that allows developers to define objects in pure URBI or directly plug C++ classes, or "UObjects," into the kernel to add these classes to the language as native URBI classes. You can even unplug UObjects from the kernel and run them as remote autonomous applications, taking the IP (Internet Protocol) address of the URBI engine as a parameter.



Figure A The CoroBot platform is available in nine configurations.

A key consideration in the URBI language is the integration of parallelism and events into the core of the language semantics. The URBI language supports four types of temporal constraints between commands. One is that Task B must execute after Task A. A second specifies that Task B must start when Task A ends, whereas the first constraint allows a temporal gap between the two tasks. The third constraint is that tasks A and B must start at the same time, meaning that, if one of the tasks is unavailable, the other will wait until the other constraint becomes available before executing. The fourth constraint is that Task B must start at the same time or after Task A begins, but it must start before Task A completes.

Because URBI is a parallel language, it can handle concurrent accesses with more than just mutex (mutual-exclusion) techniques to ensure that only one piece of code is using a resource at a time. URBI supports seven blend modes that specify how the system should handle conflicting and simultaneous assignments. Examples of these blend modes are the add and mix modes, which add or average the calculation of the conflicting assignments to the resulting value. The queue mode implements a classic mutex mechanism.

To better support parallelism, the notion of time is part of the URBI-language semantics. For example, a simple assignment in URBI can target a variable to reach a value in a given time or at a given speed or to set a sinusoidal oscillation. These noninstantaneous assignments can execute in parallel with other assignments. As an example, consider the assignment `neck.val=10 time:450ms&leg.val=-45 speed:7.5 &tail.val=14 sin:4s ampli:45;` This assignment uses "time," "speed," "sin," and "ampli" to modify the way the assignment completes. In this example, the value of "neck.val" will reach 10 in 450 msec. Other supported modifiers include "phase," "getphase," and "smooth."

URBI natively expresses parallel-event handling because several events can occur in parallel and trigger some code execution that could run in parallel and overlap. In practice, the simplest way to react to an event in URBI is to use the "at" command, which looks like an "if" statement in that it performs a command when the test becomes true. However, unlike an "if" statement, the "at" command remains in the background to trigger again and does not terminate. Another such tool is the "whenever" statement, which loops the execution of the command as long as the test is true. This statement is similar to a "while" statement except that it remains in the background when the test is false. The language can also emit events with or without parameters.

With the sensing algorithm, the closer the sound reaches equidistance between the two microphones, the harder it becomes to discern the direction of the sound. This phenomenon is due in part to the fact that the source sound hits each microphone with a smaller time difference relative to the sampling rate. For this project, this situation was acceptable. This phenomenon also means that, because the robot has been turning, the direction of the sound gets closer to the center of both of the microphones. So, the motor movements should get smaller as the detected sound's direction becomes more equidistant between the two microphones. Otherwise, the robot, whose movement is coarse, could end up bouncing between two positions.

## BUILDING THE SOFTWARE

Working with the LabView development environment took a little getting used to. My experience with programming has been predominantly text-based coding with languages such as C and assembly. Working through the tutorials helped significantly, especially as they helped me to become familiar with the location of and means of accessing tool resources. National Instruments has been improving the LabView development environment for more than 20 years, and many add-on tools extend the environment for domain-specific applications. LabView's virtual-instrumentation tools for data collection, display, and analysis are easy to use, and you can set up sophisticated displays for data analysis and troubleshooting—one of the strengths of the LabView environment.

During the 1990s, I used an early version of LabView for a tunable laser-control system. I did some of the programming with visual-language tools, but I did much of it in C because I found it hard to make the transition to a fully visual programming model. After much reflection on my current project, I can describe why the transition is difficult for me. Over the years, I have adopted coding styles that impart intent and design information “in the white space” of the code. In other words, the indenting of the code, the location of blank lines, and the splitting of long or complex command sequences across lines of code all impart valuable information to a reader who is familiar with how the software developer made these decisions. I have

**MORE AT EDN.COM**

⊕ For a related blog post about robotic-development resources, go to [www.edn.com/080207df1b1](http://www.edn.com/080207df1b1).

⊕ For a related article about mechatronics, go to [www.edn.com/article/CA6491142](http://www.edn.com/article/CA6491142).

⊕ For a related article about smarter vehicles, go to [www.edn.com/article/CA6339246](http://www.edn.com/article/CA6339246).

neither developed an analogous means to impart information in the white space of a visual-programming model, nor am aware of an industry approach for this strategy, although I have not looked for one. In doing this project, I can see how a developer could use left-and-right and up-and-down flows to impart intent and other information in the white space.

## USING A VISUAL-PROGRAMMING MODEL CAN MAKE IT EASIER TO IMPART INFORMATION ABOUT PARALLELISM THAT WOULD BE MORE DIFFICULT TO DO WITH TEXT-BASED CODING.

Using a visual-programming model, such in LabView and Microsoft Robotics Studio, can make it easier to impart information about parallelism that would be more difficult to do with text-based coding. You can position the sequencing constructs so that you can see that they can operate at the same time, and you can more easily see whether they are sharing any resources. Both of these environments allow you to mix visual programming with text-based coding by encapsulating the text-based code in blocks that work within the visual environment. An example from the Robotics Studio tutorials raised a concern about visual programming. The example showed how to implement a previous example—implemented with a variable and a loop—without using these struc-

tures. I imagine that my inexperience with visual programming hid the loop from my eyes as I looked at the rewritten code, but I get nervous when a loop is not inherently obvious just by looking at the structure of the code.

I would like to have simulated the robotics with the LabView environment, and, although you can link LabView with the MathWorks' Simulink environment, I couldn't try that approach for this project. On the other hand, I was able to download the Microsoft Robotics Studio and immediately start simulating a robot. Unfortunately, according to Kyle Johns, senior developer at Microsoft, the simulation environment provides a physical and visual model for every object in the environment but currently lacks support for simulating sound, which I needed for this project. To be fair, Microsoft's environment targets robotics, and I used only the predefined robots from the available manifest. However, it was nice to place a robot in an environment and see what it did and what features in the environment the robot could see using an intuitive highlighting method. I am not sure how much work it takes to set up a robot manifest so that you can simulate it, but a number of basic configurations exist for many of the supported robotic platforms, so you can start working with them right away. It will be interesting to see whether these two environments eventually complement each other and work together.

The visual-programming tools for Robotics Studio are less mature than the LabView environment, but the tools worked fine. I noticed an interesting problem with the Robotics Studio when it came to executing some code. One of the tutorials shows how to do looping and convert text to speech. It was fun to hear the system count. However, it was disconcerting when the program would sometimes mix up the order of the numbers if I performed a context switch during program execution. In other words, the message passing exhibited a last-in, first-out behavior, so that, if the system happened to be busy enough to receive an overrun message, it could miss the message and catch up to the earlier message later in an out-of-order fashion. This quirk may be the nature of the text-to-speech block, but it was an unexpected behavior. This type of behavior could cause troubleshooting sessions

if the out-of-order execution is less obvious than it was in the code I was using.

Another example of the maturity of the development-environment interface of Robotics Studio appeared when I performed context switching to another program in the middle of a spawned dialogue box. I sometimes had trouble getting back to the dialogue box if it was under the parent window, and the parent window would lock up waiting for the spawned dialogue to complete. The dialogue box did not show up on the task bar in Windows XP, but I eventually figured out that I could manually select it using the Alt and Tab keys.

This project is just the first step in a series of projects that I hope will build on each other and introduce more complex-

---

## FOR MORE INFORMATION

**CoroWare**  
www.coroware.com

**Gostai**  
www.gostai.com

**HiTechnic Products**  
www.hitechnic.com

**iRobot**  
www.irobot.com

**Lego**  
www.lego.com

**MathWorks**  
www.mathworks.com

**Microsoft**  
www.microsoft.com

**National Instruments**  
www.ni.com

**Texas Instruments**  
www.ti.com

ity to reach the ultimate goal of discerning an arbitrary sound in a noisy environment with a binaural-sensing system. In addition to the goal and value of the project, working with the development platforms provided an opportunity to demonstrate that resources are available to developers to aid in developing complex robotic-control systems. A common expressed goal is that developers should be able to design to a common hardware specification and then be able to use that specification across a variety of robotic platforms through runtime binding without having to redesign it.

I am excited about what is currently available and expect to see a flurry of ac-

tivity with all of these development platforms over the next few years as they do a better job of jump-starting new robotic projects and enabling developers to reuse software and hardware components from previous projects. I am especially excited that some of these development environments are treating these systems as a set of distributed systems that can interact with each other. This feature will be a key enabling capability as designers build systems that consist of multiple robots working collaboratively. **EDN**

---

## ACKNOWLEDGMENT

*Special thanks to software engineer Brady Duggan and media-relations specialist Tiffany Morrison of National Instruments for their assistance and support in acquiring and working with the LabView development tools, the electronic-“sheep-dog” reference design, and the Lego NXT and supporting hardware components for this project.*

---

## REFERENCES

1 Cravotta, Robert, “Robots on the march,” *EDN*, Dec 3, 2007, pg 44, [www.edn.com/article/CA6505566](http://www.edn.com/article/CA6505566).

