

Microsoft ROBOTICS

STUDIO

A Technical Overview for Hobbyists and Running the BOE-Bot under MSRS

Microsoft Robotics Studio is demonstrated at the RoboBusiness 2006 trade show, www.robobevent.com.



by George Mitsuoka

Microsoft Robotics Studio (MSRS) may be the most important robotics-related product yet introduced in the 21st Century. But there seems to be some confusion about how MSRS changes the robotics landscape and, in particular, how it will affect robot hobbyists and hobby robotics. In this article, I'll try to explain some of the technical aspects of MSRS from this perspective.

What is Microsoft Robotics Studio? MSRS is a set of software tools, a runtime framework and samples designed to aid in the development of robotics software and applications. These tools can be accessed from Microsoft's Visual Studio integrated development environment (IDE)—the same environment used to develop most applications for Windows-based personal computers, or they can be accessed directly without Visual Studio. This instantly gives robotics software developers the option of accessing some of the most sophisticated, mature, and proven development tools ever created. It also means that developers are not tied into a single program-

controller in one of these small robots. Instead, a small program, written in the robot's native programming environment runs on the robot and provides a set of services such as "read left bumper," "set left motor speed," or "play tone." An "orchestrator," written in MSRS and running within an "MSRS Runtime environment" communicates with these services to control the robot. In order for this to work, a communication link has to exist between the

The ER2 prototype home robot was designed by Evolution Robotics, www.evolution.com, to showcase autonomous navigation and pattern recognition through interactive activities like chess, Pictionary and book reading.

programming language as they might be with other robot-specific tools. MSRS users can choose from C#, VB.Net, and other languages. In addition, Microsoft has developed a visual programming language (VPL) specifically for Robotics Studio.

CONTROLLING HOBBY ROBOTS

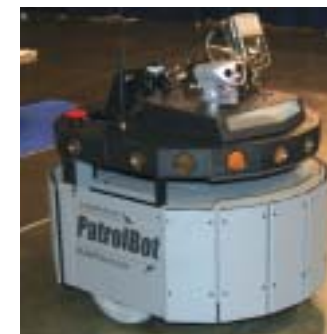
Can MSRS be used to control a hobby or educational robot such as LEGO Mindstorms, Parallax BOE-Bot, or other microcontroller-based robot? Yes, but the key word here is "control." MSRS isn't used to create code that will execute directly on the micro-



PHOTOS BY GEORGE MITSUOKA, KEN BERRY, LEM FLIGHT AND WALTER SIDAS

Runtime and the robot. For Mindstorms NXT and Parallax BOE-Bot, this can be done via a Bluetooth wireless link.

This seems to be a very different and more complex approach to programming robots, so what are the advantages? The MSRS software approach is different from that usually used in microcontroller based robotics, but whether it is any more complex depends on the problem you are trying to solve. Completely autonomous microcontroller based robots can be simple to program for simple tasks, but as tasks become more complex, programmers quickly hit the limits of the microcontroller's performance and memory resources. It then becomes very



The mobile PatrolBot from www.MobileRobots.com.

complex to manage the available resources and squeeze as much performance out of the microcontroller as possible. At some point, even relatively simple problems become too complex for the limits of the microcontroller. For these reasons, most microcontroller based robots simply don't exhibit very complex behaviors. In contrast, an MSRS Runtime on a modern PC may have virtually unlimited computational and memory resources available to it, and the applications using it can be much more complex.

A program created for a microcontroller based robot will usually only run on that specific robot or an identical one. But an orchestrator created in MSRS can control any robot that provides the necessary services. For example, you could create a maze-solving program in MSRS and test it initially with a BOE-Bot that provides bumper and motor services. The same code would then work with Mindstorms and Roomba robots that provide equivalent services. This level of abstraction from the real hardware lets developers solve problems once, instead of continuously



A miniature industrial robot from Robots and Design, Ltd., <http://www.rnd.re.kr>.

A MOMENT IN HISTORY

In the late 1970s and early 1980s, the computer industry was just starting to take off. Big businesses, government, and universities could afford expensive mainframe and mini-computers and the custom software and support they required. Consumers and hobbyists could choose products from a variety of companies including Apple Computer, Atari, Commodore, and Radio Shack. But these consumer-level products fragmented a relatively small market and truly useful software applications were difficult to find.

Then, in 1981, IBM introduced the IBM PC and the rest is history. As the leader in the computer industry, IBM brought much needed legitimacy to the personal computer market and created a de-facto standard around which hardware and software developers could focus their efforts. Today, after years of building on standardized technology rather than reinventing it, the technological offspring of the IBM PC are affordable, ubiquitous, and more powerful than the fastest computers of the 1980s. Casual users can custom build PCs from plug-and-play components and software applications are readily available to perform almost any conceivable task. In addition, the widespread use of personal computers has accelerated the development and adoption of tangential technologies including PDAs, digital cameras, MP3 players, and the internet.

Today, the robotics industry is just starting to take off. Big businesses, government and universities can buy or build expensive custom robots for specialized applications. Consumers and hobbyists can choose products from a variety of companies including Parallax, Wowee, LEGO, and iRobot. But these consumer-level products fragment a relatively small market and truly useful personal robots are still difficult to find.

Enter Microsoft and Microsoft Robotics Studio. As an industry leader and technology powerhouse, Microsoft brings much needed legitimacy to the personal robotics market and has created a standard platform around which hardware and software developers can focus their efforts. Now, instead of repeatedly reinventing robots from the ground-up, we can build on the efforts of others who use the MSRS standard. With standardization and mass-production, robots will become affordable, ubiquitous, and very useful. Casual users will be able to custom build robots from plug-and-play components and software will be widely available to allow them to perform almost any conceivable task.

A wild prediction? Maybe. But a quarter-century after what has proven to be the watershed moment in personal computer history, we may have just witnessed the personal robotics equivalent.

reinventing the wheel when writing robotics software.

Now, these are great features, but controlling these small, microcontroller based robots is not really what MSRS is all about. These robots are limited by their physical characteristics, not just the software they run, and if their physical capabilities are good enough for the problem you are solving, there's a good chance their native software tools

will be good enough as well. The best reason to use MSRS with these robots is to learn basic robotics, learn how to use MSRS, then go on to solve much more advanced problems.



Paro the therapeutic robot harp seal is a stuffed animal robot that was developed by Japan's National Institute of Advanced Industrial Science and Technology. Photo from RoboNexus 2005.

MSRS RUNTIME

Suppose you want to create a truly autonomous robot, one that's not tethered to a PC? Can you still use MSRS? Absolutely. Do you need a PC on board your robot? Perhaps surprisingly the answer is no, but you need a robot with enough onboard resources to run the MSRS Runtime.

What is this 'runtime'? The MSRS runtime is perhaps the most critical component in Microsoft Robotics Studio. It is based on Microsoft's .NET Common Language Runtime (CLR) and provides a cross-platform execution environment that combines features of a microprocessor and operating system in one package. Programs written in any of the MSRS-compatible languages are

compiled, but not to the native machine language of a particular processor. Instead MSRS compilers generate program executables in an intermediate language (IL). When a program is launched, the MSRS runtime compiles the IL into the native machine language of the host processor and executes it. The runtime also provides sophisticated operating system (OS) features such as concurrency control, coordination, scheduling, and I/O services.

Because the intermediate language is not processor-specific, programs developed in MSRS can be executed on any processor/operating system which hosts the runtime. Currently these include x86 and x64 PCs running



LEGO MINDSTORMS RCX (top) and NXT (bottom) hobby/educational robotics design systems.



Windows and in the future mobile platforms running Windows CE. It should be possible to port or recreate the Runtime on other 32-bit or 64-bit processors and operating systems.

Does this mean that microcontrollers won't be used in robots developed with MSRS? Absolutely not. Microcontrollers will be used more than ever. But instead of trying to control everything in a robot with a single microcontroller, robots will be made with many microcontrollers providing an array of services and orchestrated by one or more processors hosting an MSRS Runtime. When personal computers were first developed there was a single CPU that controlled everything. Today, a PC consists of many microcontrollers (each of which may be more powerful than the first PC CPUs) each dedicated to controlling specific peripherals such as disk drives, mice, sound, and graphics while offloading the main CPU for more complex tasks. Likewise, robots will have microcontrollers dedicated to individual or groups of sensors, actuators, and motors. The functions of these devices will be made available by the microcontrollers as services.



MSRS SERVICES

The term "service" comes from a software engineering paradigm known as Service Oriented Architecture (SOA) which is often used when creating enterprise-scale applications that communicate over the internet. An MSRS service is very lightweight and can be thought of as a kind of subroutine in a program. However, there are two key differences between MSRS services and subroutines in traditional programs:

■ MSRS services are executed concurrently and in isolation from each other, even when running on the same PC, and the same OS process, providing a very robust runtime environment.

■ MSRS services are specifically designed to be accessed remotely, usually over a network, and often use the same HTTP protocol used by web browsers to access web pages over the internet. This allows robotics applications to easily be composed of services running either on the same PC or on different PCs across the Net.

SOA has many advantages including the ability to create and test individual services in isolation. Once a service has been created and tested, it can usually be integrated into a solution without fear that the integration will cause the service to fail. SOA also leads naturally to the use of multiprocessing, which will be critical to the development of sophisticated robot applications. Also, because services within MSRS use the HTTP protocol, it is easy to create a test page for a service that can be accessed via a web browser.

This is one key difference of the MSRS service model compared to the more traditional web services model: MSRS services have to conform to an HTTP-like protocol that always ties in their current state with the behavior of the service. This protocol, called DSSP (Decentralized Software Services Protocol) provides transparency on how a service operates, debugging and aids in scaling across nodes.

ROBOT OF THE FUTURE

What will the electrical hardware of the robot of the future look like when using Microsoft Robotics Studio? Here's my prediction: The LANBot. The central piece of electronics in a robot is not going to be a PC architecture computer as some have predicted. Instead, the centerpiece will be a high-speed network router with a wireless bridge that allows it to connect to an external network. Various robot components such as arms, drive systems, and sensors with their own dedicated microcontrollers will provide control services via a network interface that connects to the router. Many low-power components will only need a single cable connection that provides both power and network signals. High-power components such as drive systems and arms may use an additional power connector. This one or two cable unified wiring system will greatly reduce the complexity of wiring a robot. Once everything is hooked up, what you have is a robot with all components connected via a local area network (LAN) that can be bridged wirelessly to an external network. I call this a LANBot.

Software development for a LANBot using MSRS will be simplified.

A software developer can write, test, and debug code locally on his PC. He simply needs to point his orchestration code running on his PC's local runtime to the remote services running on the LANBot. Service signals will flow transparently across the wireless network to and from the LANBot components as if his PC were sitting on the robot. There will be no need for a time-consuming and difficult-to-debug download and test cycle to a



The Robonaut is a humanoid robot that mimics the actions of an astronaut wearing a special suit.



The Spirit and Opportunity class Mars Rover.



Kyosho's MANOI humanoid robot is an extremely capable Robo-One category robot that may be setting a new standard on the competition scene.

controller on the robot. Once software development is complete, the robot can continue to be controlled via the wireless link by an external runtime. Or a small computer optimized to run the MSRS Runtime without the overhead of a traditional PC can be loaded with the orchestration code, installed on the robot, and connected to the LAN.

MSRS is ready to use now with existing robotics platforms. However, the true potential of MSRS will become apparent in a few years (or sooner), when robotic companies begin selling plug-and-play components with very sophisticated capabilities. For example, Honda could take technology developed for ASIMO and create a biped base (or several with different weight capaci-

ties) that knows how to balance, walk, and climb stairs. Its services would include "move forward," "turn left," "climb stairs," etc., and it would be simple to control using an MSRS orchestrator and network connection. You could take this base and combine it with a vision system from Evolution Robotics, and a bartending arm of your own design to create a personal mobile bartender. By wirelessly connecting to the internet, your bartender will have instant access to recipes for all the mixed drinks in the world. The possibilities are endless and exciting.

Honda's Asimo may be the most advanced biped robot in existence.



As reviewed elsewhere in this issue (see page 44), the popular BOE-Bot kit from Parallax provides a great introduction to microcontroller-based robotics. What I really like about the BOE-Bot kit is how the 'Robotics with the BOE-Bot' manual incrementally walks you through the multifaceted nature of building and programming a small robot. Robotics requires the creation and integration of mechanical, electronic, and software systems and the BOE-Bot manual covers these varied aspects in a clear, easy-to-follow style that is unparalleled in its quality and level of detail. After completing the exercises, you have a good understanding of everything involved in building a small robot, and feel you can continue on to design and integrate your own additional components.

A lot of the credit goes to Andy Lindsay, a product engineer at Parallax and author of the BOE-Bot manual and other educational texts from Parallax. He really does a great job of explaining complex ideas and doesn't assume that the reader already knows a lot about robotics, electronics, or programming. Now, Andy has brought his clear and thorough style to the task of helping students and hobbyists learn how to use MSRS with Parallax's new "BOE-Bot kit for Microsoft Robotics Studio" (MSRS BOE-Bot).

BLUETOOTH WIRELESS

The MSRS BOE-Bot is the popular BOE-Bot kit packaged with a Bluetooth module and additional software and documentation to allow you to use it with Microsoft Robotics Studio. You can program the BOE-Bot with its native development tool, the BASIC Stamp Editor, and also control it remotely from a PC running MSRS using the Bluetooth module. This kit may be the best way to learn the basics of microcontroller-based robotics along with the powerful MSRS software tools that will allow you to create the truly useful robots of the future. These skills really complement each other well. Complex robots will require the use of both microcontrollers to control individual hardware components, and high-level software to coordinate the components, solve higher-order problems, and provide better intelligence.

While building and testing the BOE-Bot and later using it with MSRS, I found it to be an interesting study in contrast. Thanks, in large part, to great documentation, the BOE-Bot and BASIC Stamp Editor are fairly easy to

PARALLAX BOE-BOT KIT FOR MICROSOFT ROBOTICS STUDIO



learn and comprehend, but also limited in capability. On the other hand, Microsoft Robotics Studio is complex and powerful. What you can do in PBASIC is limited by the memory and performance of the BASIC Stamp: there are only a few thousand bytes of memory and the BASIC Stamp can only execute a few thousand instructions per second. However, the performance and memory available to an MSRS orchestrator running on a modern PC is virtually unlimited.

As much as Microsoft would like everyone to believe that it is easy to get started with MSRS, I think people without prior Windows programming experience will find the initial learning curve to be somewhat steep. Programming a BOE-Bot with the BASIC Stamp Editor usually involves just editing a single file of PBASIC code and clicking the "run" button to download and execute it. Under MSRS, you need to load a PBASIC program on the BOE-Bot that communicates with the PC over Bluetooth and sends and receives sensor and motor data. You also need a "service" on the PC that handles the PC side of the communication and provides the BOE-Bot control interface, and you need an "orchestrator" that actually controls what the robot does in response to its sensors. On top of all this you may also need a graphical user interface (GUI) to show sensor state and/or allow remote control. MSRS comes with extensive sample code and tutorials, so you don't have to write all of this yourself, but it is still a lot to comprehend.

RUNNING THE BOE-BOT UNDER MSRS

Fortunately, Andy Lindsay has taken on the challenge of making MSRS approachable using the BOE-Bot. Working with Microsoft engineers, he has created a custom protocol, the necessary service, and PBASIC code to allow MSRS to communicate with a BOE-Bot, read its sensors, and control its motors. By itself this is an interesting feat, as the single-threaded BOE-Bot has to handle asynchronous messages coming from the PC without the benefit of large buffers. More importantly, Andy has written a tutorial that guides the reader through the steps necessary to get a BOE-Bot running under MSRS control. By carefully following the steps in his tutorial, I was able to quickly demonstrate both remote control of the BOE-Bot from a GUI running on my PC, as well as autonomous navigation

being controlled from MSRS.

I did run into a few glitches along the way and some unexplained behaviors. They were minor, but did allow me to explore one of the more interesting aspects of software development with MSRS: debugging. When programming the BOE-Bot in PBASIC with the BASIC Stamp Editor, debugging options are limited. If the BOE-Bot is connected to your PC with a serial cable, you can use DEBUG statements to print data to a debugging window; you can also use LED or speaker indicators on the BOE-Bot itself. That's about all you have to work with. In contrast, when debugging code running on your PC in MSRS, you have all the features of a modern visual debugger available to you. You can view source code, set breakpoints, single step through code, examine the call stack, watch variables, and so forth. So, once the basic services provided by the BOE-Bot are stable, it becomes much easier to develop, test, and debug complex behaviors.

One of the great things about the MSRS BOE-Bot kit is that after you've unpacked everything from the box, that isn't the end of what you get. Parallax is continuously updating the supporting web site with new documentation and software. Over the next few months, we can expect updates that add new sensors and capabilities to the BOE-Bot and extend the services available in MSRS. I also expect that Andy will provide more tutorials and documentation and make this new and powerful tool even more accessible.

I hope you have found this article useful and that it has peaked your interest in Microsoft Robotics Studio. Please send your feedback to me at georgem@botmag.com. We at *Robot* will continue to cover developments in Microsoft Robotics Studio as they occur. Stay tuned! @

Links

Microsoft Robotics, www.microsoft.com/robotics
Parallax, www.parallax.com, (888) 512-1024

For more information, please see our source guide on pg. 97.