by Steve Norris

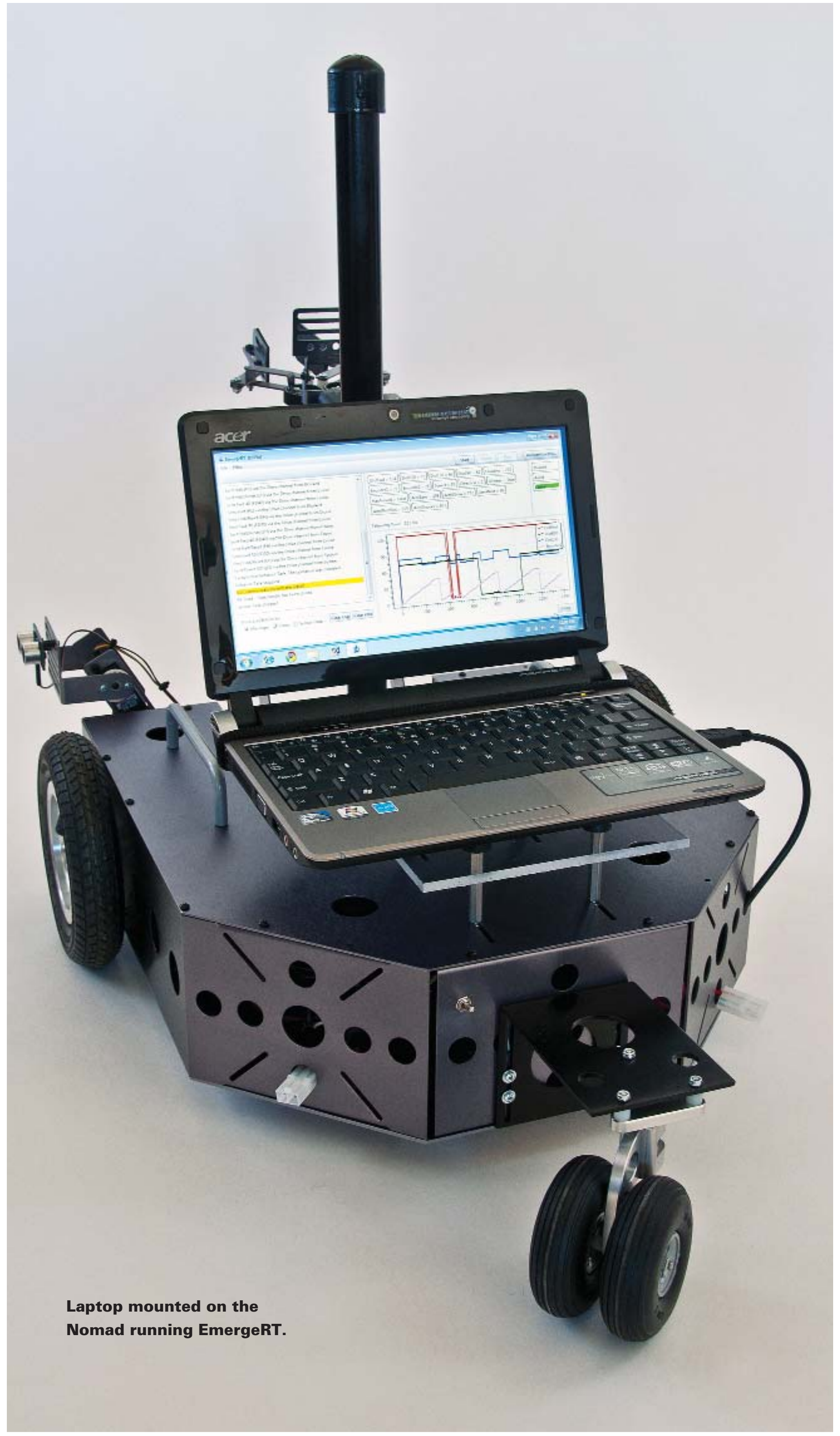
The CrustCrawler Nomad HD Rover PART 2

Exploring emergent behavior

In the last issue I reviewed the new CrustCrawler Nomad HD Rover and presented my implementation using the Parallax Propeller chip as the primary controller. In this article we will expand on the original platform and give the Nomad even greater capabilities. My original plan was to add additional sensors for beacon and line navigation as well as a video camera. But for now I will not go down that path; instead I will take a detour and expand the Nomad's computational abilities by giving it a bigger "brain." And with this bigger brain we will explore the fascinating world of emergent behavior using behavior-based programming.

Despite evidence to the contrary I do not spend *all* my time building robots. During the day I'm a mild-mannered software engineer developing health care applications based on the Microsoft Windows platform. The primary language I use is C# (pronounced C-sharp) which is one of the multitudes of programming languages (including C++, Visual Basic, J#, F#, etc) available in the Microsoft .NET platform.

The .NET platform includes a Common Language Runtime (CLR) which provides an abstraction layer over the Windows operating system, a large collection of pre-built base class libraries for common low-level programming tasks and a powerful development environment called Visual Studio. Given my familiarity with this platform it seemed only natural (and inevitable) that I use these tools to build a robot application running on a Windows-based PC.


Laptop mounted on the Nomad running EmergeRT.

PHOTOS BY STEVE NORRIS

THE CRUSTCRAWLER NOMAD HD ROVER

If you missed the last issue here is a quick review of the Nomad HD Rover. The Nomad is constructed from laser cut .063 gauge (sides and top plate) and .090 gauge (bottom plate) 5052 Type II anodized aluminum in gun barrel gray. Fully configured the Nomad weighs in at 12.6 pounds and has two grab handles mounted on the top deck for easy pickup. The lower inside and outer upper deck both measure 18 inches long and 14 inches wide with a body depth of 4 inches. As you will see, this leaves plenty of room for electronics, batteries, and can easily accommodate a small laptop computer. One unique feature of the Nomad is its "Robotic Arm Deck" which will accept any of CrustCrawler's robotic arms including the SG5-UT, SG6-UT, and AX-12 Smart Arm.

You can purchase the Nomad in a number of configurations bundled with the Parallax Motor Mount Kit, the Parallax Caster Kit and robotic arm of your choice. The kits do not include microcontrollers, motor controllers, or sensors. These all will need to be purchased separately depending on your configuration.

EMERGENT BEHAVIOR

As I mentioned before, the primary reason for moving the Nomad to a PC platform is to explore a concept called emergent behavior. Emergence is the result of an interconnected system of simple entities that self-organizes into an intelligent higher-level behavior that is greater than the sum of its parts. In these systems agents residing on a lower level produce behavior that lies on a level above. Although we are not generally aware of it, we see emergence in our everyday lives: Ants organize into colonies, birds organize into flocks, and people organize into neighborhoods and cities. None of these organizations have any central control but instead their structure emerges from the simple interaction of their respective participants.

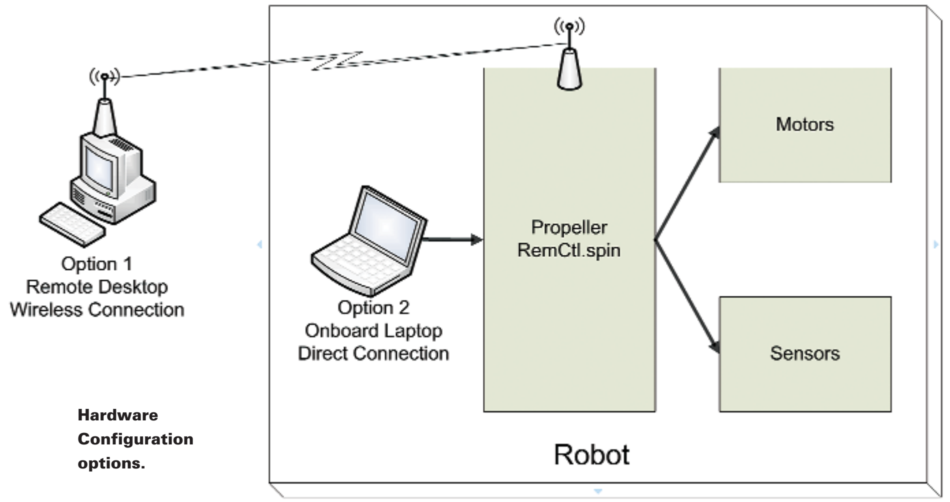
Robotic and software engineers alike are now discovering the power of emergent behavior. Using behavior-based programming robots can be built that exploit the emergence model that results in seemingly highly intelligent behavior.

The direct connection to the Nomad using a USB port.

BEHAVIOR-BASED ARCHITECTURE

The behavior-based software architecture uses a collection of behavior modules that get input from the robot's sensors. The behavior processes this sensor data and determines if it should trigger and send a request to the robot's actuators. Each behavior is called sequentially based on its priority. If a behavior is triggered it will place a request to the robot's actuators into a central queue. After all the behaviors have been executed an arbitration process then decides which behavior request in the queue will be sent on to the robot's actuators. The decision is based on the behavior's priority. The highest priority behavior requesting access to the actuators will win and override all other lower priority requests.

As an example let's say we have created a blocked behavior and assigned it the highest priority. This behavior would probably use a forward facing ultrasonic sensor as its input and, based on the measured distance, determine if the path ahead is blocked. If so it would trigger and request a simple



Hardware Configuration options.

A bungee cord holds the laptop in place.

"move back 12 inches" instruction. The blocked behavior has been given the highest priority and will override all the other behaviors and have its move request sent on to the motor drive.

HARDWARE CONFIGURATION

Looking at the hardware configuration diagram you can see that I use one of two basic hardware configurations. The first option uses a remote PC connected to the robot via a wireless link. This configuration allows the use of a powerful desktop PC without having to physically mount it on the robot. The wireless link is implemented using a pair of XBee transceivers. The PC is connected to one of the XBee modules using the Parallax XBee USB Adapter Board and a USB A to mini B cable. The adapter board provides an easy interface to configure the XBee module and it converts the XBee 2mm pin spacing to more useable 0.100-inch pin spacing.

As an added bonus the board also has four cool status indicator LEDs for Power, RSSI, Associate and Mode. The other half of the XBee pair is installed within the Nomad and is directly wired to the Propeller Proto Board using the Parallax XBee Adapter Board. Like the USB version this adapter converts the pin spacing but does not have the USB hardware.

The second hardware configuration option involves mounting a laptop directly on the back of the Nomad. Considering the size and cost of small laptops (sometimes referred to as Netbooks) this makes this option quite viable. I've been using the Acer Aspire One which comes with Windows 7 Starter, an Intel Atom processor, Wi-Fi, and a built in webcam and costs around $280. I connected the laptop to the Propeller Proto Board using one of laptop's three USB ports and a Parallax Prop Plug. As an alternative I could have used the USB version of the Propeller Proto Board which has the Prop Plug built into it. The primary advantage of using this "laptop onboard" configuration is the elimination of the XBee transceivers although there is the added expense of weight, power and some usability due to the small keyboard and screen.

I mounted the laptop to the top of the Nomad using four standoffs and a small sheet of Plexiglas. I install several inverted rubber feet to the sheet to hold the laptop steady and absorb any shock. A short bungee cord straps the laptop down onto the rubber feet and Plexiglas sheet.

In both options the Propeller is programmed with the same remote control application that receives and processes commands and returns sensor telemetry. It uses a simple serial communications protocol that I designed to be easy to implement and use. All packets start with a three character header and end with an ASCII Return character. Telemetry data returned from the robot is formatted in a similar manner.

THE EMERGE FRAMEWORK

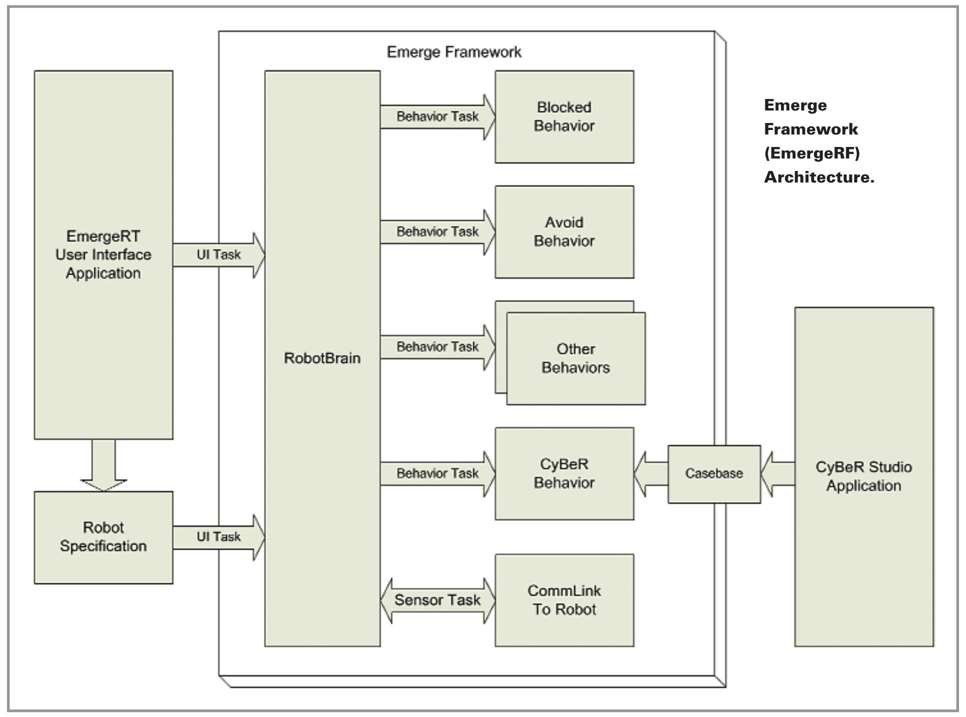
Now that we understand the behavior-based architecture and the hardware configuration let's look at the Emerge Framework shown in the accompanying diagram. The Emerge Framework (EmergeRF) is a .NET software library that implements (in C#) the infrastructure needed to construct behavior-based robotic applications. The library contains all the class libraries and methods needed to support the acquisition of telemetry data, the execution of behaviors, and the arbitration of requests to be sent to the robot's actuators (motors). By using this framework, even a novice .NET programmer can easily build and test behaviors. If you are not familiar with C# or .NET programming and would like to learn more look at the list of recommended books at the end of this article.

The top level of the framework is the RobotBrain class. The Emerge Runtime (more on the Runtime later) creates an instance of this class which then loads all the implemented behaviors. All behavior implementations must be derived from the base Behavior class. The behaviors for a particular robot are compiled as a set and placed into a single .NET assembly. The behaviors are loaded at runtime and their location is specified in a RobotSpecification class that you setup for each robot you wish to support.

There are three major and concurrent tasks (threads) that run in the framework. The Sensor Task handles the requesting and receiving of sensor data from the robot. The RobotSpecification object contains the definition of how this data is formatted since it will most likely vary with each robot. There are currently two different communications links that the Sensor Task could use and both are derived from the CommLink class. The first is the serial communications link (SerialComm) is used for direct and XBee connections. The other (and not implemented yet) is used for TCP based communications.

The Behavior Task handles the execution of the loaded behavior set and then arbitrates the request queue. The behaviors are executed in priority order and are passed the current sensor readings, the request and name of the last winning behavior. The behavior returns its request (if any) which is placed in the request queue. This makes arbitration process trivial; simply grab the first request in the queue. Note that all the behaviors are executed even if a higher priority behavior has made a request. This allows all the behaviors equal time and the ability to update their state based on the current sensor readings.

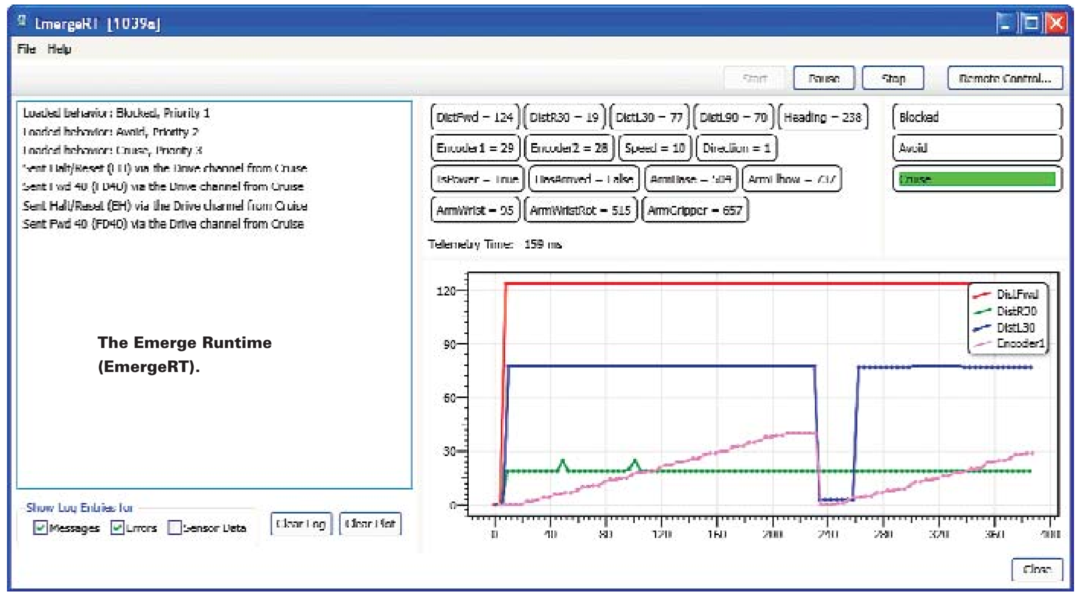
The last of the tasks is the User Interface (UI) Task. This task handles the user interface in the runtime environment which we will discuss next.



Emerge Framework (EmergeRF) Architecture.

THE EMERGE RUNTIME

The Emerge Runtime is both the user interface for the robot and the environment in which the framework runs. From the File menu a user selects the robot configuration that they wish to use. This file is used to initialize a RobotSpecification object which is then passed to the RobotBrain instance. The user interface interacts with the RobotBrain object and displays "brain" activity in one of four panels. The left most panel is the activity log. Behaviors have the ability to post messages to this running log which can be used to display internal status or error information. The top center panel displays the current sensors reading received over the communications link. In conjunction the bottom panel is a plotter which can graph any or all the sensors values received. The top right panel displays the names of all the loaded behaviors and indicates (in green) the last arbitrated winning behavior.



Loaded behavior: Blocked, Priority 1
Loaded behavior: Avoid, Priority 2
Loaded behavior: Cruise, Priority 3
Sent Halt/Reset (EH) via the Drive channel from Cruise
Sent Fwd 40 (FD40) via the Drive channel from Cruise
Sent Halt/Reset (EH) via the Drive channel from Cruise
Sent Fwd 40 (FD40) via the Drive channel from Cruise

DistFwd – 124, DistR30 – 19, DistL30 – 77, DistL90 – 70, Heading – 238
Encoder1 = 29, Encoder2 = 28, Speed = 10, Direction = 1
IsPower – True, IsArmed – False, ArmBase – 504, ArmElbow – 617
ArmWrist = 95, ArmWristRot = 515, ArmGripper = 657

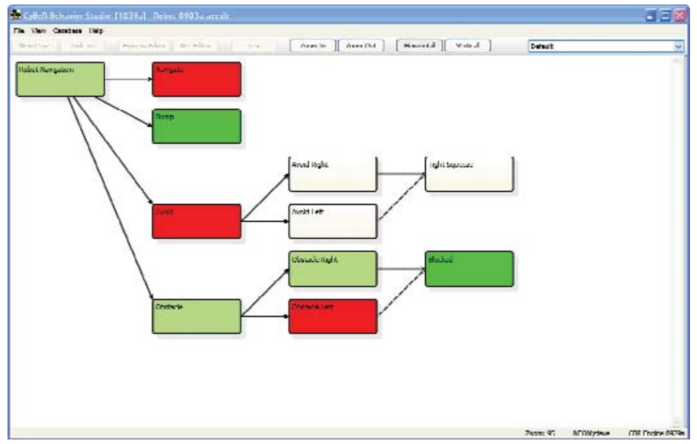
Telemetry Time: 159 ms

Blocked
Avoid
Cruise

☑ Messages ☑ Errors ☐ Sensor Data

The Emerge Runtime (EmergeRT).

The toolbar located at the top allows a user to start, stop and pause the framework. In addition there is a Remote Control button which disables the behaviors and allows manual control of the robot. This is very useful in emergency situations where the "emergent behavior" is not exactly what you wanted or expected.


Screen shot of the CyBeR Studio.

THE CYBER BEHAVIOR AND STUDIO

The CyBeR Behavior really deserves an article all by itself. Its name is derived from the underlying technology which is called Case-Based Reasoning (CBR). Case-Based Reasoning uses the concept of matching a case against a given situation. The CBR engine moves through the case tree trying to match cases against a set of provided feature values. In a robotics application the features are actually sensor readings. When a case is matched (triggered) an action associated with the case is sent to the arbitration process just like any other behavior.

Using the CyBeR Behavior allows non-programmer to build behaviors but not have to implement them in a programming language like C#. Instead the user defines the trigger and action of a behavior using a graphical design tool call the CyBeR Studio. The Studio allows cases to be created and linked to each other in a tree like structure. This structure determines the order in which they will be matched. Each case has a set of matching rules that are defined using a simple set of operators like "equal to" or "greater than". The Studio also supports the testing of behaviors by graphically displaying the execution of case matches on the screen. Colors are used to note which cases matched and which failed.

Mona loves to settle into a warm laptop!

THE FUTURE

I think that the CyBeR system is an interesting way to think about and construct behavior-based robotic applications. It is my plan to continue and grow both the Emerge Framework and CyBeR Studio. If there is enough interest, and I hope there is, I plan to start an Open Source project to share this platform with my fellow robotics enthusiasts.

READING LIST

Illustrated C# by Daniel Solis

Pro C# and the .NET Platform Fifth Edition by Bill Wagner ◎

Links
Acer Aspire One Netbook, http://us.acer.com

CrustCrawler, www.crustcrawler.com, (480) 577-5557

Microsoft Visual Studio, www.microsoft.com/express/downloads/

Parallax, www.parallax.com, (888) 512-1024

Steve Norris website, www.norrislabs.com

For more information, please see our source guide on page 89.