Robotics Society of Southern California

The Robot Builder

Volume Eleven Number Twelve December 1999

Notices

· Practical Mobile Robotics Class - 11:00AM - 12:00PM

· Business Meeting - 12:30 - 1:00

· General Meeting - 1:00 - 3:00

Distribution

If you would like to receive The Robot Builder via e-mail, contact the editor at:

apendragn@earthlink.net

Inside this Issue

Basic Ideas of Digital Signal Processing

By Arthur Ed LeBouthillier

It's no secret that computers are nearly everywhere. Although it's not always obvious, they're already embedded into everything from microwave ovens to toys. They're inside your car, your radio, your digital camera and your calculator. It seems that nearly everything made today has a processor in it. However, just as nearly every personal device that needs control or sequencing has been computerized over the last few decades, it is likely that embedded computers will be used to perform a new kind of processing in the coming decades: the processing of analog signals.

When computers first came out they were applied wherever some kind of sequencing, timing or control operation needed to be performed. That's why places like your microwave's timer seems like an obvious place to stick a microprocessor. Turning things on and off seems like an obvious application of something that "thinks" in terms of on and off (or binary) signals.

However, with the advent of cheap digital signal processors (DSP's), embedded computers are taking on new kinds of roles which depend on the processing of continuous, time-varying signals themselves. Before the advent of cheap digital signal processors, these kinds of operations were far too difficult for computers to perform.

Analog signals are different from the digital signals that computers use. Digital signals consist of sequences of 1's and 0's where there are discrete differences between two levels. Analog signals, however, may vary continuously over a given range. Therefore, whereas a digital signal may consist only of two voltage levels of 0 and 5 volts, an analog signal varies continuously between those limits. Analog signals can take on intermediate values such as 0.1 volts or 0.5 volts or 3 volts. Analog signals are more representative of such things as sound, which can vary smoothly between no sound to very loud sound.

Signal Processing

Let's concentrate on this idea of signal processing first. Your radio is a signal processor; it takes a signal which is transmitted from a radio transmitter, filters out unwanted signals, amplifies that signal, filters out more unwanted signals and finally amplifies the signal again so that it can drive the speakers. Two of the most obvious kinds of signal processing from this example are amplification and filtering. Amplification

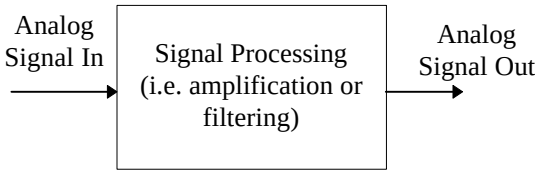

Figure 1 - Analog signal processing

increases the value of a signal and filtering removes unwanted kinds of signals. Mathematically, we can see a signal processor as solely a mathematical function. A signal of one sort goes through a function

See ***DSP***, Page 2

(a black box) to produce another, more desirable type of signal.

Until recently, this kind of processing of converting one kind of analog signal into another analog signal has been performed directly with circuitry. An engineer's job was to design the circuitry that performed this signal processing.

Digital Signal Processing

Digital Signal Processing (DSP) seeks to replace formerly analog processing of signals via electronic circuitry with software operations on computer-based processors. DSP's perform the process of taking an analog signal, doing some kind of processing on that signal (albeit with a computer) and then outputting some kind of analog signal.

As the figure 2 shows, a DSP takes an analog signal, converts it into a digital signal, operates on that digital representation in the computer and then outputs an analog signal through a digital to analog converter. The analog to digital converter takes a continuously varying signal and converts it to a stream of digital numbers representing the amplitude of the input signal at a given moment. The computer takes this stream of numbers, performs some mathematical functions on it and then outputs a stream of digital numbers to an digital to analog converter. The digital to analog converter takes the stream of digital numbers and converts them to a continuously varying analog signal. The overall operation is: Analog in, Digital Processing, Analog Out.

The reason that DSP's are becoming popular for performing the processing of analog signals is that the benefits of digital systems can be brought to the processing of analog signals. These benefits include easy re-programming, easy enhancements and upgrades, and, in some cases, far superior processing for lower costs.

Basic DSP operations

When we discussed a radio, we said that two of the most important operations were amplifying and filtering. As an example of why DSP's have become popular, we can understand the process of amplification as merely multiplying the signal by a factor. Therefore, if the value coming into the analog to digital converter were 0.2 volts and we wanted to amplify the value by 10, all we need do is multiply the number 0.2 by 10 to obtain 2.0. The processor would then output this value to the digital to analog converter to produce a signal which is 2.0 volts.

The other primary operation performed by DSP's is filtering. Filtering is a process of removing some unwanted frequency component from a signal. A low pass filter passes low frequencies and excludes high frequencies. A high pass filter passes high frequencies and excludes low frequencies. A bandpass filter excludes all high and low frequencies which are not within a narrow band of frequencies. All of these operations can be performed easily with a DSP because they are actually only an averaging process. Although a full understanding of how they work requires extensive mathematics, the fact is that they are only mathematical functions.

Because filters can be understood as mathematical functions, they can be tailored very precisely. DSP's can create filters which are far superior to

See ***DSP***, Page 3

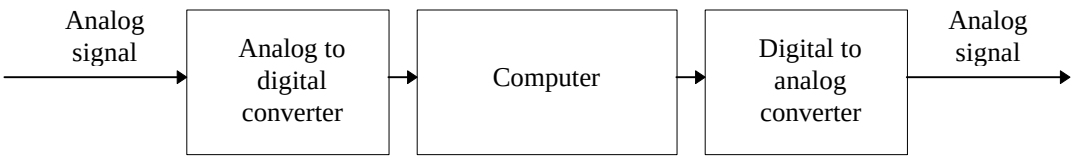

Figure 2 - The structure of a digital signal processor

Electronic circuit-based filters because their mathematical functions can be more precisely specified. Unlike electronic circuit-based filters, they are not affected by temperature and aging and as computers get faster, a small DSP can perform the function of dozens of filters.

In fact, it is possible to use DSP's to create filters that would be nearly impossible to create using electronic circuitry. An engineer can specify nearly any kind of filtering function and a single DSP can perform what would take hundreds or even thousands of transistors. Because these filters are only programs running on a DSP, they can be changed in a few microseconds and the whole system can be reconfigured to perform some other operation.

Where they're used

Because of the ability to perform very complicated processing of analog signals, DSP's have been and will be used in many different places. Filters have already been mentioned, but DSP's will be used in the next few years in speech understanding systems, voice compression, voice encryption, music compression (MP3), noise elimination, and video processing. They're used in engine emission control systems in your car and in your cellular phone. They'll be in high-density television (HDTV) sets.

One area that is receiving a lot of attention lately is that of motor control. Controlling motors optimally using DSP's is envisioned. By using a DSP to control a motor, it can be made to be extremely efficient. This is because the DSP can tailor its control over the motor to get the most optimum operation depending on the speed of the motor. This will minimize losses due to heating and will also make motors last longer. Just motor control alone is envisioned as a near-revolutionary area and it is envisioned that DSP's will be in virtually every motor in the next few years.

Another area that will utilize DSP's heavily will be speech recognition products. The basic process of speech recognition relies on being able to identify time-varying frequency trends that form the basic sounds of speech. In order to analyze speech with a computer, the incoming sound must be analyzed in terms of their basic frequencies and how those frequencies are related. This is often done with a process known as determining the spectrum. A spectrum takes a signal and breaks it into the frequency components that form it; it is as if there were a number of filters, one for each frequency. By analyzing trends and relations in the frequency components from these filters, a computer can characterize these basic sounds. By using pattern matching techniques on these sound elements, the word being spoken can be recognized. Being able to do this requires being able to convert a sound into its spectrum quickly. DSP's have been optimized to speed up the process of converting a sound into its spectrum.

DSP Optimizations

What makes a DSP different from a regular computer is the fact that its design has been optimized to perform the operations needed in signal processing equations. Multiplication is the first example. DSP's are designed with fast multiplying circuitry allowing a DSP to perform up to hundreds of times more multiplications in a period of time. They are also optimized so that repeated operations like multiplying a number by a constant and adding the result to a running sum are a single operations. These kinds of operations are extremely common in signal processing math. Other than these optimizations, though, DSP's are merely microcomputers which have been optimized to speed up certain kinds of math.

Summary

DSP's seek to replace the processing of analog signals by software running on a computer. In order to do this, they are optimized to do certain math operations quickly. Because they are becoming cheaper yet they offer superior capabilities in some applications, they are becoming more commonplace in places that were formerly processed with analog circuitry.

B.E.A.M. Robotics

by Arthur Ed LeBouthillier

B.E.A.M. robotics has arisen as a new approach to building robots. Its primary originator, Mark W. Tilden, has developed a robot design philosophy which is very different from traditional robot-building techniques, emphasizing a minimalist, evolutionary design approach.

Principles of B.E.A.M.

B.E.A.M. is an acronym for Biology, Electronics, Aesthetics, and Mechanics. It stands apart from most other robot building strategies because it emphasizes a no-computer, minimalist approach based on insect-like nervous systems. Rather than concentrating on trying to make a computer do the abstract thinking that living things like humans do, it instead calls for mimicking the neuron-level circuitry of the lowest levels of living neural systems. The emphasis is on trying to build reflexive, stimulus-response systems more like a vertebrate's spinal cord than its brain. As Brian Bush says in the B.E.A.M. FAQ, "the basic principles of BEAM rest on the fact to build smart machines one must first build a smart body."

Nervous Nets

The simplest processing elements in a B.E.A.M. robot are the Nervous (Nv) and Neural (Nu) neurons; an Nv is a motor driving element and an Nu is a signal processing element. According to Mark W. Tilden, founder of the B.E.A.M. approach, Nu's and Nv's are "… real-time non-linear analog control system emulating a low-level peripheral spinal system. Based on arrays of sequential RC (Resistor-Capacitor)-time-based pulse delay circuits in closed loops, a Nv net is any circuitry that can act as a media for sustaining independent control 'processes.'" The basic element of an Nv or Nu is a schmitt-triggered inverter with an RC timing circuit or similar circuit. Schmitt-triggered means that the inverter has a strong threshold before it fires; this helps to ensure a clean oscillating behavior in a circuit. Figure 1 shows a simple Nv or Nu circuit.

By combining these Nv's into simple feedback circuits, different kinds of sophisticated behaviors arise. One reason for the sophistication is the feedback caused on the circuit by motor loads. Since drive motors have a different resistance when a load is applied, they become both actuators and sensors. This feedback can be anticipated by the designer so that the feedback causes adaptive behavior.

Multicores

Nv's and Nu's are combined into larger control circuits called multicores. Depending on the number of Nv's and Nu's in the system, they are known as monocores, bicores, tricores, quadcores, quincores, hexcores, septcores, octocores and so on. These kinds of circuits become coordinating circuitry between several motor elements and thus bring

See *BEAM*, Page 5

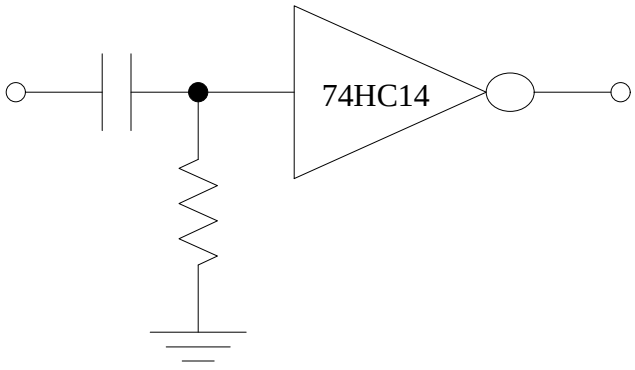

Figure 1 - A basic B.E.A.M. Nv neuron circuit

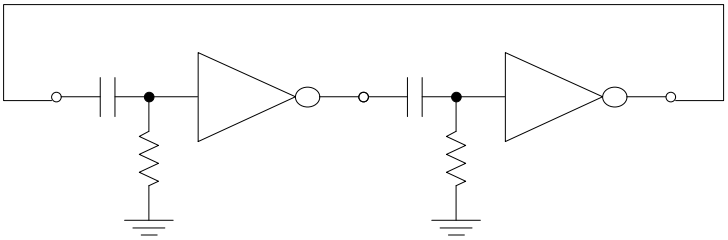

Figure 2 - The Bicore

about certain patterns of motor behavior. They can be combined in leg-control or wheel-control systems to produce capable moving mechanisms which react to their environment. One of the simplest multicores is the bicore. The bicore puts two Nv's together to form a closed-loop motor driving circuit. Motors could be attached to the output of each Nv.

Multicores are similar to control elements biologists have found in animal muscle control systems called Central Pattern Generators (CPG). A CPG is a neural circuit which produces coordinated muscle patterns. Biologists have identified numerous kinds of CPG's in various creatures. Lobsters, for example, have numerous CPG's that control the walking or resting posture. Like CPG's, multicores act as an adaptive sequencing control over the robot's legs or wheels.

Biomorphs

B.E.A.M. enthusiasts envision building robots which are near-living machines which they call biomorphs. As Mark Tilden and Brosl Hasslacher say in **Living Machines**:

> What is different about biomorphic machines from typical mobile platform designs is not their materials base but how they are organized. They use a dynamical, non-symbolic internal world representation and compliant, bi-directional, interactive response where the external world assumes a crucial role. In this they have much in common with biological forms which is not accidental; these machines are designed along biological paradigms rather than on first principle notions of how such machines should be organized.

Tilden and his associates have produced dozens of different biomorphs over the years. These range from very simple 2 neuron creatures to complex biomorphs with over a dozen neural elements. These robots are not programmed in the way that microprocessors are, but rather are designed using the Nv and Nu circuitry to have propensities towards certain kinds of behaviors. Often, the actual behaviors which spontaneously arise are far different from those anticipated by their designers.

Summary

B.E.A.M.is a robot design philosophy which stresses building life-like mechanisms using neural-like circuitry. Utilizing loops of pulse-forming devices, complete adaptive control systems are formed with very few components. This is similar to mechanisms found in creatures such as insects and lobsters. These machines evidence emergent behaviors which are sometimes sophisticated and often unpredictable. They can be fascinating to watch and learn from.

References

The BEAM FAQ - Brian O. Bush
http://people.ne.mediaone.net/bushbo/beam/FAQ.html

Nervous Networks - Brian O. Bush
http://people.ne.mediaone.net/bushbo/beam/nvnet.html

Advanced Nervous Networks - Brian O. Bush
http://people.ne.mediaone.net/bushbo/beam/advnvnet.html

Living Machines - Brosl Hasslacher and Mark W. Tilden

Basic C Programming (Part 1)

By Arthur Ed LeBouthillier

The C programming language is one of the most important programming languages you can learn. It is available for virtually every kind of computer around and is one of the more powerful languages. This series of articles will introduce some basic ideas of C programming.

C is a typed language

From your grade-school math, you'll remember hearing the idea of different kinds of sets of numbers. You probably barely remember hearing about natural numbers, counting numbers, whole numbers, real numbers and imaginary numbers. The C language also uses this idea of different types of numbers. Let's look at the first kind: ***integers***. If you remember the idea of whole numbers, they were numbers which didn't have a fractional portion after them. Therefore, 0, 1, 2, -1, and 100 are examples of whole numbers, whereas 0.5, 1.25, 2.9, -1.6 and 100.75 are not examples of whole numbers. C has its own terminology for these kinds of numbers; they are known as ***integers***. In C, an integer is a whole number without a decimal point like -1, 0, 1, 5 and -3. C represents what is called a typed language because you must always be aware of the type of value you are operating on. Going further, C makes a distinction between type ***int*** (short for integer) and type **float** which are numbers which do have fractional values (i.e. -6.2, 33.1, 600.5). You must learn to distinguish these kinds of numbers in order to use C properly. There are other types in C such as long integers and many others and you can even define your own types, but it is enough to understand that C requires you to be conscious of the type of number you are using.

C is a Functional Language

C has another idea that it borrows from math, that of functions. Writing a C program consists of defining functions which perform the operations you wish to perform. You're probably familiar with the idea of a function from grade school math. You'll probably remember the general function formula: **y = f(x)**. In this formula, ***y*** represents a variable which will be assigned some value, which is the product of applying the function ***f*** to the variable ***x***. We don't know the value of **f(x)**, though, because it is not defined. In math terms, we could define: **f(x)** = 5x+3. This definition of **f(x)** means that whenever we evaluate a function such as **y = f(5)**, we evaluate the formula for f(x) by replacing the general variable, ***x***, with the value, 5 and then compute the result.

When you write a program in C, you also define functions, although C has its own peculiar way of writing down functions. If we were to define the function ***f*** in C, it would look like this:

```
float f(float x)
{
        return 5 * x + 3;
}
```

Remembering the idea from the last section of typing, we now see how C uses it. In this function definition for ***f***, we see the words ***float f(float x)***; this is how we tell C that we are going to define a function ***f*** which returns a type ***float*** and which takes an argument, ***x***, also of type float. Having defined this function, we could then call it from within a program by typing:

```
y = f(5.0);
```

The result of this would be that the ***float*** number 5.0 would be assigned to the variable ***x*** in the function ***f***, the answer would be calculated and then the result would be assigned to the variable ***y***. One other thing to notice about this line of code is the presence of the semi-colon at the end of the line. Statements in C are separated by a semicolon and most often appear at the end of a line in order to tell the system where the end of the statement is.

Conclusion

That's all for now, we'll look at some more ideas about C programming next month.

Robotics Society of Southern California

President	Randy Eubanks
Vice President	Henry Arnold
Secretary	Arthur Ed LeBouthillier
Treasurer	Henry Arnold
Past President	Jess Jackson
Member-at-Large	Tom Carrol
Member-at-Large	Pete Cresswell
Member-at-Large	Jerry Burton
Faire Coordinator	Joe McCord
Newsletter Editor	Arthur Ed LeBouthillier

The Robot Builder (TRB) is published monthly by the Robotics Society of Southern California. Membership in the Society is $20.00 per annum and includes a subscription to this newsletter.

Membership applications should be directed to:

Robotics Society of Southern California
Post Office Box 26044
Santa Ana, CA 92799-6044

Manuscripts, drawings and other materials submitted for publication that are to be returned must be accompanied by a stamped, self-addressed envelope or container. However, RSSC is not responsible for unsolicited material.

We accept a wide variety of electronic formats but if you are not sure, submit material in ascii or on paper. Electronic copy should be sent to:

apendragn@earthlink.net

Arthur Ed LeBouthillier - editor

The Robotics Society of Southern California was founded in 1989 as a non-profit experimental robotics group. The goal was to establish a cooperative association among related industries, educational institutions, professionals and particularly robot enthusiasts. Membership in the society is open to all with an interest in this exciting field.

The primary goal of the society is to promote public awareness of the field of experimental robotics and encourage the development of personal and home based robots.

We meet the 2nd Saturday of each month at California State University at Fullerton in the electrical engineering building room EE321, from 12:30 until 3:00.

The RSSC publishes this monthly newsletter, The Robot Builder, that discusses various Society activities, robot construction projects, and other information of interest to its members.

Membership/Renewal Application

Name _____

Address _____

City _____

Home Phone ( ) - Work Phone ( ) -

Annual Membership Dues: ($20) Check #
(includes subscription to The Robot Builder)

Return to: RSSC
POB 26044
Santa Ana CA 92799-6044

How did you hear about RSSC? _____

RSSC
POB 26044
Santa Ana CA 92799-6044

Please check your address label to be sure your subscription will not expire!