Non-holonomic Robotic Control Using the BasicX-24p Microcontroller

John M. Kuperavage, CIT
Multi-Dimensional Integration
jkuperavage@mdiadvantage.com

John R. Wright, Jr., Ph.D., CSIT
Department of Industry and Technology
Millersville University of Pennsylvania
john.wright@millersville.edu

Abstract

The study of microprocessors and microcontrollers is considered to be a staple topic of study in many electronics, control systems, mechatronics and automation systems technical programs. Processors and controllers continually improve at an exponential rate. Instructors of courses in microprocessors and microcontrollers need constant exposure to the latest technologies available to ensure that their students remain on the cutting edge of our technological capability. The presentation will focus on how the BasicX (BX-24p) microcontroller may be utilized to control non-holonomic autonomous mobile robots. This includes robotic components such as: motors, end effectors and sensors. The BX-24p is an industry grade microcontroller that is very robust and powerful (i.e. 83,000 instructions/second with multitasking capability). It is also compatible with the BASIC Stamp microcontroller carrier boards, the BASIC Stamp's components and other components commonly found in technology classrooms. Throughout this paper, many helpful BasicX control functions are explained and discussed that were utilized for the development of two mobile robots that participated in the 13th Annual Trinity College Home Firefighting Robot Contest.

Introduction

Teaching a microcontroller course in an Industrial Technology curriculum can be quite challenging due to the speed of advancing technologies in the world of electronics. One way to keep pace with the ever-changing technologies is to develop a course around small low powered components that still give students the experience of developing microprocessor-controlled systems. This allows instructors to replace the equipment with up-to-date devices more frequently than with large, high-priced industrial grade equipment. Students are able to acquire a hands-on experience on a small scale and then transfer that knowledge later to more high-end industrial equipment.

One such small-scale system that is often used as an introductory microprocessor/microcontroller experience is the BASIC Stamp II (BS2) microcontroller by Parallax. [1] Often accompanying the BS2 is an educational carrier board called the Board of Education (Boe) and a robotic chassis giving the whole system the nick name "Boe-Bot." Parallax also makes a variety of sensors and add-on components, which in most cases have all the necessary circuitry prepackaged with the sensor. [2] These low priced devices (Boe-Bot with the BS2 and numerous sensors) allow students the opportunity to develop robotic systems using microprocessors in an environment that is safe, easy to interface, cost effective and enjoyable.

Although the lessons learned using the BS2 are easily transferable to more advanced systems, a pitfall of using an educational microcontroller is the limitations as to what you can do with functionality. Recently, Parallax upgraded their BS2 microcontroller so that its components are now industry grade, but the PBASIC language/functionality is still limited. For example, multitasking is not possible with PBASIC. Having said this, PBASIC is very easy to use (no separate compiler, many canned functions, etc.). This issue of functionality has been resolved through the development of the BasicX (BX-24p) microcontroller. This microcontroller from NetMedia, Inc. is not only industry grade, but it is backward compatible with all BS2 components including the Boe-Bot. The BX-24p is also a "Pin-For-Pin drop in replacement" [3] for the BS2 (Figure 1).

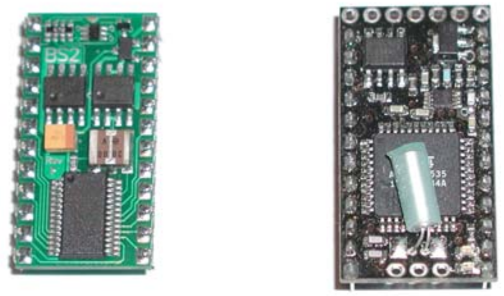

Figure 1: BS2 chip (Left), BX-24p chip (Right)

These two important characteristics allow a user to simply replace the BS2 with the BX-24p without any other modifications (such as additional electrical interfacing) to operate any component that may be operated by the BS2. This gives students the benefit of utilizing a much more realistic, high-end system (industry grade programming language) at the same price as the educational systems with all of the user-friendly components (Table 1) that Parallax offers. Since the components are easily interfaced with the microcontroller, students can devote considerably more time developing code and building the actual system than doing laborious electrical interfacing/breadboarding work.

Table 1: Comparison of the BS2 and the BX-24p

	BS2	BX-24p
Instructions per Sec.	4,000	83,000
RAM (Bytes)	32	400
EEPROM (Kilobytes)	2	32
I/O Pins	18	21
Input pins w/Analog capability	None	8
Price	$49.00	$49.95

The language of the BX-24p is called BasicX and is almost 100% compatible with the popular languages QBasic and Visual Basic. It also has the programming structure of other popular languages such as C, C++, Java, and Perl. [4] This allows students to learn these languages and techniques and then implement them in the classroom without having standard industrial equipment. Once students move beyond higher education, they will have knowledge of commonly used languages and techniques and not just theory.

All of the capabilities of the BX-24p make it a good choice to control systems that need high-powered performance at a low cost. This was such the case when an advanced microprocessors course (ITEC 467, Microprocessor Electronics) at Millersville University decided to take on the daunting challenge of creating a pair of fully autonomous robotic vehicles for the 13th Annual Trinity College Firefighting Home Robot Contest in Hartford, Connecticut. The goal of the competition was to build a mobile robot that would autonomously transverse a model house, find a candle representing a fire, and extinguish the flame in the shortest amount of time. [5] The class was split into two teams, each developing distinctly different mobile robots that utilized the BX-24p (Figure 2). Each team member helped research and develop portions of the robotic system including: motion control, sensor integration, end effectors (extinguishing devices) and integration into one complete automated device. The BX-24p can be utilized to control different functions of such autonomous mobile robots. This paper will focus on many of the common programming techniques used to control these fire fighting robots and other types of mobile robots as well. Specific functions will be discussed including sample code to operate those functions.

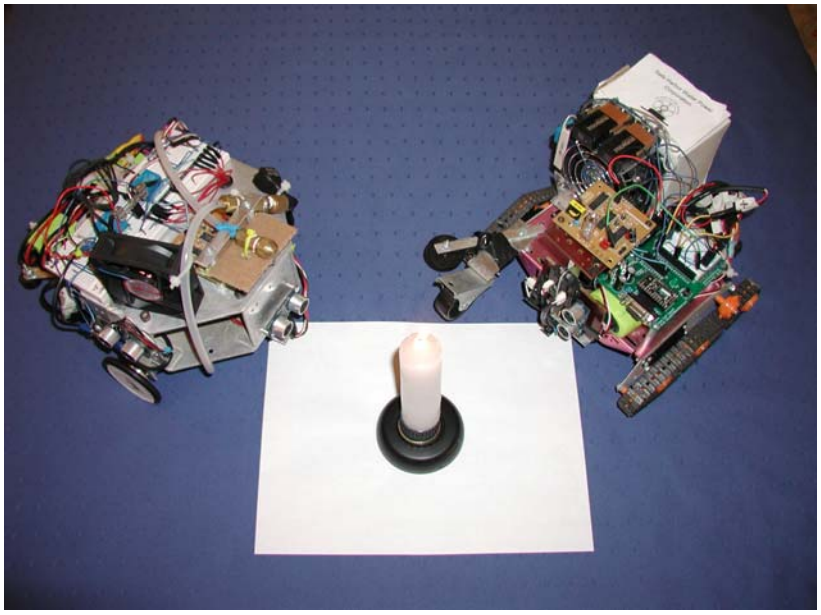

Figure 2: Non-holonomic autonomous mobile robots built to compete in the Home Firefighting Robot Contest

Motor Control

Many of the difficulties of controlling mobile robots involves having a motion controller that is small enough to be on the mobile robot, powerful enough to control the device and simple enough to be programmed with ease. In this case, the BX-24p comes to the rescue. Robotic motion control can be realized using a Motor Mind C carrier board (MMC_BS2) that is designed for use with the BS2, but since the BX-24p is backward compatible with the BS2, integration from one to the other can be easily done. Some of the features of this carrier board include a place to insert a microcontroller chip (BX-24p), places to solder wires for I/O and a place to insert a motor controller. The motor controller typically used with the MMC_BS2 is the Motor Mind C (MMC) which is capable of driving two 12V DC motors (Figure 3). [6] The DC motors and the 12V DC power supply are wired directly to the MMC_BS2. Almost all of the user control is done through programming in the BasicX language which greatly reduces other types of integration. To move a robotic vehicle, the program can drive each motor at different speeds in either direction allowing any combination of forward, reverse and turning motions at any speed desired.

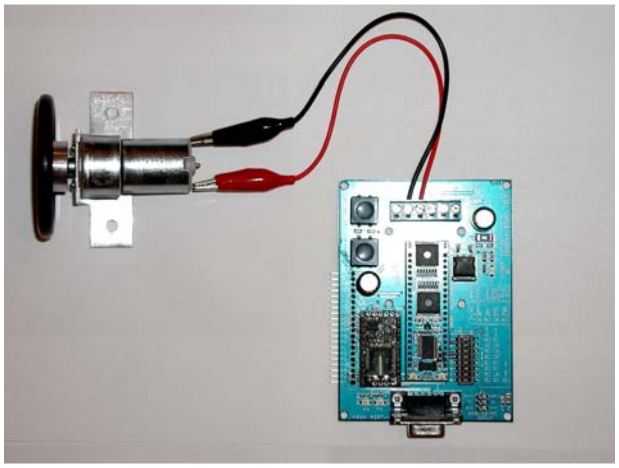

Figure 3: MMC with a 12V DC motor

To drive a motor using the BX-24p and the MMC, one must understand how the MMC motor controller works. The controller uses a hexadecimal number to set the speed of a motor. For example “0000” means no motion, “03FF” stands for 100% forward and “FC01” stands for 100% reverse. This means that any percentage of full motion in either direction can be achieved by taking that percentage of the maximum hexadecimal value. For example, to find 28% of full speed first take 28% * 1023 (Decimal 1023 = Hexadecimal 03ff). Then take that product (286) and convert it back to the hexadecimal value (011E). 011E represents the hexadecimal value of 28% of full speed. This speed along with the setup for the controller is placed in a queue in the program and sent from the BX-24p to the MMC as a command packet via a serial communication. This turns on the motor at the desired percentage of full load. Once on, the motor will remain on at that percentage until another serial communication is sent to change the speed.

The first critical step in allowing the BX-24p to control a motor using the MMC is to setup the hardware communications. The DIP switches on the MMC_BS2 should all be set to low accept switch 6. This sets the type of control signal the motor controller will receive. The FM communication pin (serial output) is pin 13 on the BX-24p while the TM communications pin (serial input) is pin 12. Because the BX-24p I/O pin out is different then that of the BS2, the I/O pins on the MMC_BS2 carrier board are 5 less than the corresponding BX-24p pins. In other words, programming serial output through pin 13 on the BX-24p would be hardwired to pin 8 on the MMC-BS2. Likewise programming a serial input through pin 12 on the BX-24p would be hardwired to pin 7 on the MMC_BS2. Pin 7 and 8 on the MMC_BS2 should never be used for any purpose other than communications. Knowledge of this issue will be assumed throughout the rest of this paper for all I/O references. Finally, the BUAD rate of the monitor port on the BasicX software should be set to 19,200.

When writing the actual program (Figure 4) the first things that need to be done are declaration of the arrays, opening the queues and clearing the queues. The output queue (OQ), input queue (IQ) and data packet string (data) are all set up as byte arrays. OQ and IQ are set to the size 17 (8 bytes of data and 9 bytes of overhead) while data is set to the size of 8 (8 different bytes of data). The OQ and IQ queues are first opened and then cleared to allow data to be sent and received through them. After the arrays are setup the program’s communications must then be established. Using the DefineCom3 command the input pin (12), output pin (13) and communications parameters must be declared. The pattern of bits in the communications parameter determines the type of logic (inverter or non-inverted), parity (odd, even or none) and the number of data bits (7 or 8). The *OpenCom* command determines the communications port to be used, the BUAD rate of data transmission and input/output queues. This setup is done to allow a serial string of data to be sent to the MMC.

The serial string of data sent to the MMC consists of 8 different bytes of data (256-bit Hexadecimal values) placed into an 8-byte array. The first data byte is the command setup. This allows the program to set such commands as: read, write, clear fault, and change frequency. The command used to control the motor is SetDC (represented by &HD0) that is used to set the Duty Cycle of an H-Bridge. [7] Next is the MMC address followed by the length of the message. The forth and fifth bytes represent the Hi and Lo registers for the speed of motor one. For example a speed of “011E” would contain “01” in byte four and “1E” in byte five. This is also the case for the sixth and seventh bytes when controlling a second motor. Byte eight is a stop bit. This checks the sum of the previous seven data bytes and if they are equal it indicates an accurate data transmission and the end of the command packet. After the data string is sent the *PutQueue* command is called which used the OQ queue to send the data array of size eight. After a half-second pause, communications port three is closed as well as the queues IQ and OQ.

```
‘Array declaration
Dim OQ(1 to 17) as byte
Dim IQ(1 to 17) as byte
Dim data(1 to 8) as byte

‘Open and clear the queues
Call OpenQueue(OQ, 17)
Call OpenQueue(IQ, 17)
Call ClearQueue(OQ)
Call ClearQueue(IQ)

‘Communications declaration
Call DefineCom3(12, 13, bx0000_1000)
Call OpenCom(3, 9600, IQ, OQ)

‘Serial data string setup
Data(1) = &HD0 ‘Command setup
Data(2) = &H01 'MMC address
Data(3) = &H04 'Length of message
Data(4) = &HFE 'Hi register motor 1
Data(5) = &H00  ‘Low register motor 1
Data(6) = &HFE 'Hi register motor 2
Data(7) = &H00 ‘Low register motor 2
Data(8) = data(1) + data(2) + data(3) + data(4) + data(5) + data(6) + data(7)

‘Send the data string and close the com port
Call PutQueue(OQ, data, 8)
Call Delay(0.5)
Call CloseCom(3, IQ, OQ)
```

Figure 4: Code to operate the MMC motor controller

Another type of motor that is typically included on a mobile robot is a servo motor. A servo motor could be either continuous (unlimited amount of rotation) or standard (rotation through a certain number a degrees). Continuous servos can be used as a tank style drive system on very small mobile robots by connecting a wheel to the shaft of the motor. This is very simple way of controlling the motion of the robot but is limited in the amount of power and speed. If a continuous servo (or any type of) motor is used as a rear wheel drive system, then a standard servo is often used to steer the front wheels from one extreme to another. This is the case in many RC car applications.

Both continuous and standard servos can be used for end effectors and robotic arms to do numerous tasks such as open and close grippers (Figure 5), rotate a particular tool, or move a robotic arm to a different position. Almost any type of movement, when on a small scale, can be controlled through the use of small servo motors.

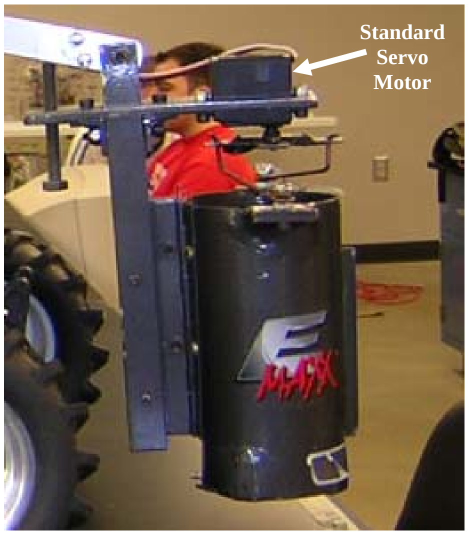

Figure 5: Servo controlled gripper

Small RC car type servos, like the ones used by many small mobile robots, are controlled by sending a pulse for a set time duration. Sending these pulses to a continuous servo will move the motor a continuously at a speed dependent on the time duration. For example, with the servos used for the firefighting robots the range of speed was: .0020s full speed one direction, .0010s full speed the opposite direction and .0015 stopped. You can set a continuous servo to any speed by adjusting the pulse width. A standard servo is slightly different. When a pulse is sent to a standard servo it will move to a position determined by the pulse width. Again using the servos implemented on the fire fighting robots: .0020s maximum rotation one direction, .0010s maximum rotation the opposite direction and .0015 neutral position. Sending multiple pulses (such as through a loop command) to a continuous servo will keep the servo at a constant speed, while sending multiple pulses to a standard servo will keep it at a constant position.

The code control these two types of servo motors is identical (Figure 6). A *PulseOut* command is called to set the motor's pulse width. The first parameter of the command is the pin number, followed by the pulse width and then type of pulse (high or low). A delay is needed to allow the servo to have time to perform its action. .02 seconds is the fastest this particular servo motor can update. When placed in a loop the motor will then receive the desired amount of pulses. This code is the same for a standard servo and a continuous servo with the exception of differences in the servo function.

```
 Dim x as integer

 'Code for spinning a servo motor
For x = 1 to 6
   Call PulseOut (8, 0.0020, 1)
   Delay (0.02)
 Next
```

Figure 6: Code to control a continuous or standard servo motor

Sensors

Sensors allow our robots to interact with their environment. They are essential for acquiring knowledge of the environment and for creating artificial intelligence. Interfacing with these devices is typically done through hardware and software setup. Common sensor classifications usually include digital and analog types. This section of the paper will identify and explain the proper setup procedures for some typical mobile robotic sensors such as IR, SONAR and light detection. The BX-24p has both analog and digital I/O capability. Pins 8-15 may be used for either analog or digital types of devices while pins 0-7 are reserved for just digital devices. It is advisable to avoid mixing digital and analog sensors on pins 8-15 without extensive testing because of a potential loading-down of the power source leading to potential false signals on the analog inputs. Typically, analog sensors have three connection pins: a voltage supply, a signal, and a ground reference (Figure 7 and 8). To implement these sensors, one simply needs to wire the correct pins and select the proper code syntax to accomplish the desired result. Figure 7 illustrates how a 3-wire sensor may be connected to the MMC_BS2 carrier board.

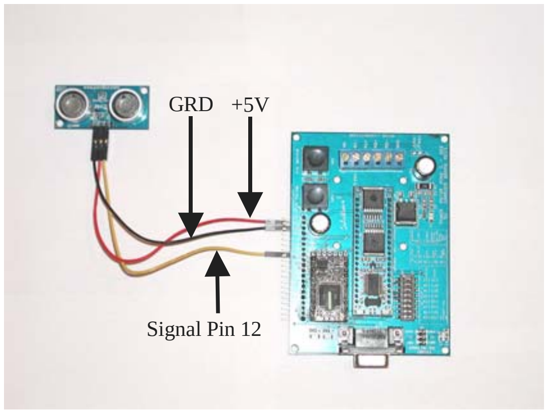

Figure 7: 3-wire sensor interfacing example

One common type of sensor used on many mobile robots is Infrared (IR). Depending on the wavelength, IR can be used for heat detection or for proximity detection. IR used for proximity detection is actual near-infrared which is about 880nm in wavelength. [8] To detect an object the IR sensor sends a beam of near-infrared night and based on the strength of the IR signal returned will determine how far away the object is. The code for object detection using an analog infrared sensor is quite simple. A variable (ADVal) is set equal to the function *GetADC*. What this does is stores an analog voltage as a 10 bit value in the variable ADVal (Figure 8). That variable can then be observed, via the *Debug.Print* command that pushes the output to the computer monitor, as in the example below or processed to make a decision in some other part of the code. The Sharp GP2D12 infrared sensor was utilized on one of the Millersville University robots entered in the competition (Figure 9).

```
Dim ADVal as Integer

'Loop to continually get the analog value and display it
Do
   ADVal = GetADC(18)
   Debug.Print CStr(ADVal)
   Call Delay(0.1)
Loop
```

Figure 8: Code to process an analog IR sensor

Figure 9: Sharp GP2D12 Sensor [9]

Another commonly used sensor is on a mobile robot is SONAR. SONAR, as opposed to IR, sends a frequency out in a conical fashion rather than a single beam of light (IR). Another key difference between an infrared and soar sensor is that the SONAR requires a pulse to be initiated while an infrared sensor sends out its beam of light continuously. SONAR sensor technology is analog in nature, yet is occasional utilized with additional circuitry to be digital (same is also true with IR). The hardwiring of the device is accomplished in the same manner as mentioned previously with IR.

Typical code to use SONAR involves taking the signal received from the sensor and converting it to a distance (Figure 10). The first step in this program is to define the variables (Width, Feet and Inches). A *Do loop* is then called to keep checking the sensor. Within that loop a *PulseOut* command is called to send an ~3 microsecond pulse to output 17. A *PulseIn* command is called next to record the signal with the variable, Width. That variable (Width) is then converted to feet and inches. The value of the SONAR signal, in inches, is then displayed on the screen via the *Debug.Print* command. Lastly a small 0.1 second delay is initiated with the *Delay* command to leave room between the sonar pulses. Figure 11 shows an easy to interface SONAR sensor utilized by the robots developed and described in this paper.

```
Option Explicit

Public Sub Main()
Dim Width as Single, Feet as Single, Inches as Single

Do
   Call PulseOut(17, 3.0, 1)
   Call PulseIn(17, 1, Width)
   Feet = Width * 550.0 'assuming 1100 feet per sec
   Inches = Feet * 12.0
   Debug.Print CStr(Inches)
   Call Delay(0.1)
Loop

End Sub
```

Figure 10: Code to read a SONAR sensor and convert the data into a distance

Figure 11: PING SONAR sensor [10]

Another type of sensor that was used by the firefighting robots was the Hamamatsu UVTRON (Figure 12). This sensor is a digital device used for UV light identification, and can detect a flame from across the room. It is typically purchased with an interface board that triggers a digital output (on or off) via the use of relay circuitry. There are two functions used to control the UVTRON (Figure 13). The first, *PutPin*, will set the desired pin to high when it is called, while the second, *GetPin* function, will read the result off of the same pin and store it in a variable (Flame). That variable can be displayed or processed for further use in the program.

Figure 12: Hamamatsu UVTRON Sensor and Interface Board [11]

```
Dim Flame as byte

Do
  Call PutPin(8,1)
  Flame = GetPin(8)
  Debug.Print "UV is " & Cstr(Flame)
Loop
```

Figure 13: Code to control the Hamamatsu UVTRON sensor

Multitasking

A valuable feature of the BX-24p microcontroller is that it possesses the ability to multitask. Multitasking is like have multiple processors working at the same time. Although this is not actually the case, a programmer can obtain very fast switching capability between tasks to improve a mobile robotic agent's responsiveness to its environment. The multitasking function within the BasicX language is accomplished by defining task stacks within the memory (RAM) available and deploying a command called *Sleep* to trigger the jump to the next task in the program. The *Sleep* function is normally used for timing, but when a programmer defines and sets up multiple tasks in a program, it is used in conjunction to achieve multitasking. [12]

In this example of code (Figure 14) multitasking takes place between two PING SONAR sensors. Two stacks (Stack1 and Stack2) need to be set up to allocate a specific number of bytes of RAM for the tasks. This is done with the *Dim* command. The tasks utilized, SonarTask1 and SonarTask2, are created by the *CallTask* command.

CallTask *Name_of_Task, Stack Memory Size Allocation*

A *Do Loop* command utilized in the main part of the program with the *Sleep* command is required to allocate time to the multitasking function. Additional *Do Loops* are utilized in the subprograms as well to continually poll the sensors. Each time the *Sleep* command is executed, the program switches to the next sequential task. In the example code below, two LEDs were wired to the MMC_BS2 output pins 0 and 1 to indicate when an object is present (within ~8 inches).

```
Option Explicit

Dim Stack1(1 To 24) As Byte
Dim Stack2(1 To 24) As Byte
Dim Width1 as Single
Dim Width2 as Single
'---------------------------------------------------
Public Sub Main()
CallTask "SonarTask1", Stack1
Call Sleep(0.01)
CallTask "SonarTask2", Stack2
Call Sleep(0.01)

Do
  Call Sleep(120.0)
Loop
End Sub

'-------------------------------------------------------------------------------------
Private Sub SonarTask1()
Do
  Call PulseOut(17, 3.0, 1)
  Call PulseIn(17, 1, Width1)
    If Width1 <= 0.0012 Then
      Call Putpin (5,0) 'turns on LED pin 0, MMC_BS2
    Else
      Call Putpin (5,1) 'turns off LED pin 0, MMC_BS2
    End If
  Call PutPin(17, 1)
  Call Sleep(0.1)
Loop
End Sub
'-------------------------------------------------------------------------------------
Private Sub SonarTask2()
Do
  Call PulseOut(18, 3.0, 1)
  Call PulseIn(18, 1, Width2)
    If Width2 <= 0.0012 Then
      Call Putpin (6,0) 'turns on LED pin 1, MMC_BS2
    Else
      Call Putpin (6,1) 'turns off LED pin 1, MMC_BS2
    End If
  Call PutPin(18, 1)
  Call Sleep(0.1)
Loop
End Sub
```

Figure 14: Code example of multitasking two SONAR sensors

It is important to note that the memory allocated to the stack size is dependent of the application. This is usually optimized by a series of trials carried out to try and minimize the allocation size without degrading the robot's functionality. The BX-24p microcontroller only allows the total assignment of task RAM to equal approximately 400 bytes. Although multitasking offers a potential ability to poll sensors, etc. in a very responsive fashion, the memory allocation of the stock BX24p microcontroller is limited. Additional memory may need to be added to explore the full capability of this function. In addition to the technical gains that multitasking may offer, it also allows a group of programmers to subdivide a robotic agents' code functions to maximize programming productivity. Tasks can be written individually by multiple programmers to be later consolidated together into a master program.

Summary

As the technology of today’s microcontrollers and processors continue to change and improve at an exponential rate, we must also evolve how we teach this complex subject matter. Industrial Technology graduates focusing in automation and or electronics/control technologies must be exposed to cutting edge technology to enhance their education in today’s global marketplace. Through the utilization of the BasicX language and the BX-24p, students will gain exposure to an extremely powerful controller technology that is industry grade and relatively easy to interface and utilize with many off-the-shelf sensor and drive components. Common programming techniques for motor control, sensor interfacing, and multitasking were described in this paper to assist those that might wish to develop a mobile robot around the BX-24p microcontroller and the BasicX language. Students utilizing these products gain the opportunity to develop programming skills that are transferable to today’s industrial languages like Visual Basic and C++. Just add imagination and a little creativity and watch your BasicX robots come alive!

Bibliography

[1] Parallax, What's a Microcontroller, n.p.: n.p., 1997.

[2] World-Wide Web URL Http://www.parallax.com. Last Accessed June 3rd, 2006.

[3] World-Wide Web URL Http://www.basicx.com. Last Accessed June 3rd, 2006

[4] Chris D. Odom, BasicX and Robotics: The Art of Making Machines think, Trenton, N.J.: Robodyssey Systems, LLC, 2005.

[5] World-Wide Web URL Http://www.trincoll.edu/events/robot/Rules/default.asp. Last Accessed June 3rd, 2006

[6] World-Wide Web URL Http://www.solutions-cubed.com/solutions%20cubed/MMC2003.htm. Last Accessed June 3rd, 2006

[7] Solutions Cubed, Motor Mind C Dual or Single DC Motor Controller Data Sheet, Rev. 4 ed., n.p.: Solutions Cubed, 2003.

[8] J. L. Jones, B. A. Seiger, and A. M. Flynn, Mobile Robots: Inspiration to Implementation, 2nd ed., Natick, Mass.: A. K. Peters, 1999.

[9] World-Wide Web URL Http://www.acroname.com/robotics/parts/R48-IR12.html. Last Accessed June 3rd, 2006

[10] World-Wide Web URL http://www.parallax.com/detail.asp?product_id=28015. Last Accessed June 3rd, 2006

[11]   World-Wide Web URL Http://www.acroname.com/robotics/parts/R67-UVTRON.html. Last Accessed June 3rd, 2006

[12] NetMedia, Basic Express Operating System Reference (PDF file) Version 2.1.

Biographies

John M. Kuperavage, CIT is a Control Systems Integrator at Multi-Dimensional Integration in Shrewsbury, Pennsylvania. He currently holds a B.S.I.T, Control Systems (2005), having graduated with honors, from Millersville University of Pennsylvania. John's work allows him to deal with multiple applications including programming PLCs, vision systems, OITs, and safety control systems. His interests include robotics, PLC programming, and furthering his education with graduate studies.

John R. Wright, Jr., Ph.D., CSIT is an Associate Professor of Automation and Electronics Technologies in the Department of Industry and Technology at Millersville University of Pennsylvania. Dr. Wright holds the following degrees: B.S.I.T., Electrical Systems (1993), M.S., Industrial Management (1995), and a Ph.D., Industrial Education and Technology (1998). He has held several positions in research and development prior to his career in academia including serving as a technician, researcher, technical manager, commercial product development manager and post-doctoral fellow. His research and teaching interests include semi-autonomous and autonomous mobile robotic development and control, industrial robotics, programmable logic controllers and systems integration.