Building a Robotics Platform using the EDKPlus and the RobitcsConnections.com's Traxster

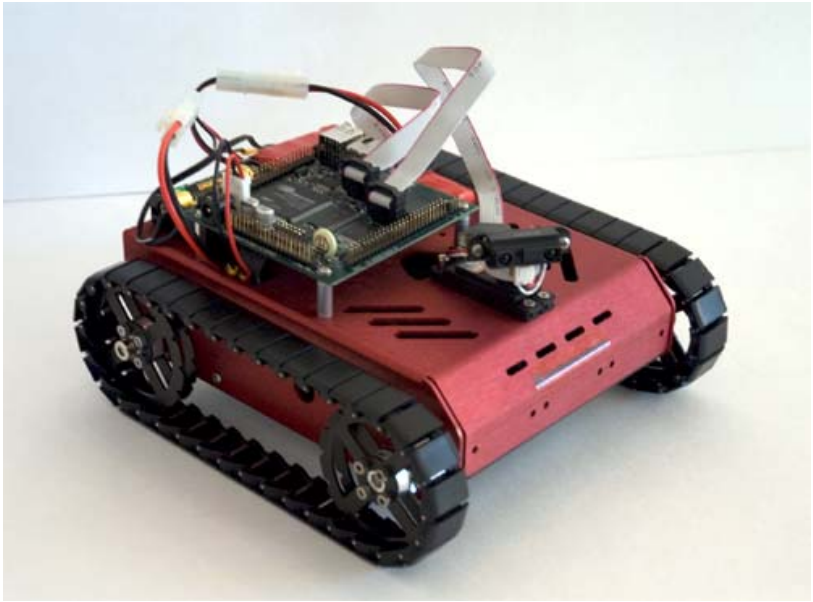
By Sean D. Liming and John R. Malin
SJJ Embedded Micro Solutions

March 2008

The Microsoft® .NET Micro Framework is a perfect development environment for robotics. With an off the shelf board and the right I/O, one could develop some very interesting robotic applications from sensor networks to autonomous motion robots. There have been many toy learning robots over the years, but useful robots are what many developers are looking to build. Motion robot's like the iRobot® Roomba® is a good example of robotics helping with daily life. The Mars' Rovers are great examples of scientific uses. To create these useful types of motion robots, one needs the right platform that handles motion, sensors, a variety of IO, has the processing power to run intelligent applications, and can be programmed using a common language like C#.

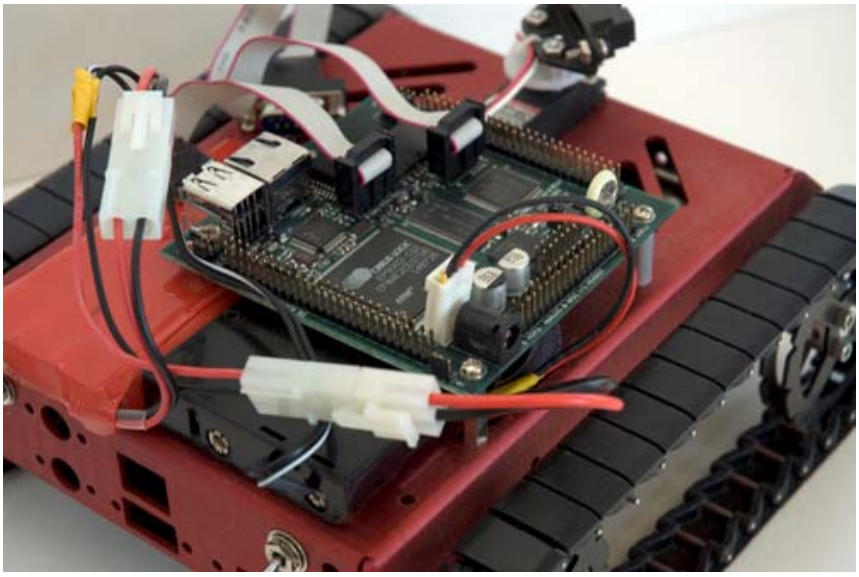
When developing the EDK, we ran across RoboticsConnection Serializer and robot kits. The Serializer board contains all the logic to control off the shelf DC motors, servos, and various analog sensors. The Serializer takes basic commands from a single serial interface to control a variety of devices, which is perfect use of the COM2 port in the EDKPlus. The Serializer's PC/104 form factor matches the EDKPlus' iPac-9302 deluxe platform so the platforms can be mounted together in a single system. The Serializer simplifies the development to control motion and obstacle sensors, which frees up the iPac9302 Deluxe's I/O (SPI, GPIO, SD/Card, and Ethernet) for other devices such as GPS, cameras, temperature sensors, robotic arm, etc.

As a basic introduction, we have developed an application using the Traxster I robot to demonstrate how the EDKPlus communicates with the Serializer. The robot will have a single IR sensor mounted on a servo. The robot's goal is to always move forward. While doing so, it will scan left center, and right to see if there are any obstacles. If an obstacle is detected, the robot will stop scan again, and turn in a direction that is not blocked. If all directions are blocked, back up and scan again. If the backup happens more than 2 times, reverse direction and keep going.



The iPac-9302 Deluxe has two COM ports. COM1 is always used for download and debug of application. COM2 is used for standard RS-232 communication in the .NET Micro Framework application. After building the Traxster platform, we mounted the iPac-9302 on top of the Traxster leaving room for the batteries underneath. A serial ribbon big tail was connected from the iPac's COM2 header to the Serial RS232 Interface module on the Serializer board. The Serial interface module already has the signals setup so you can plug the RS-232 directly into the connector. No cross over or Null modem interfaces needed. A second pig tail was connected to the iPac's COM1 header so we can download the application via a null modem connection from the PC.

The Serializer gets its power from a 9.6V battery. The same kind of battery used in many hobby car applications. To power the iPac-9302, we had to mount a 4-AA battery (6V) back via a switch on the back of the Traxster using a Serializer Wiring Harness. A floppy power connector was used to plug into the iPac-9302's floppy power connector.



***Warning**: you can only power the iPac-9302 via the banana plug or floppy power connector, but not at the same time. Applying power to both power sockets could result in injury and damage to the board.*

String commands are used to communicate with the Serializer. The protocol itself is very simple. Send a command, maybe some parameters, and a carriage return.

>command param1 param2<CR>

ACK|NACK<CR><LF>

Depending on the command, you may get a return value or some acknowledgement (ACK) that the command was received. If the command fails a no-acknowledgement (NACK) will be returned. You will have to address certain commands in your application to clear out the buffer.

Setting up the serial port (COM2) is one of the first things that need to happen in the application. First, the serial port configuration must be setup to communicate with the Serializer, which is baud rate of 9200 and hardware flow control turned on (true). The second step is to instantiate the COM2 port.

```
static SerialPort.Configuration COM2Port_Config =   new
SerialPort.Configuration(SerialPort.Serial.COM2,SerialPort.BaudRate.Baud19200, true);

public SerialPort COM2Port = new SerialPort(COM2Port_Config);
```

The commands are all strings, but byte arrays need to be sent using the .NET MF serial port methods. The next step is to create a set of constants for moving the robot forward, left, right, and reverse. Also constants are needed for Stop, Servo motion, and the Sensor. Finally we will define the byte arrays for each command.

For the setting up the robots motion, the 'mogo' command is used to drive the two motors 1 and 2. The speed for each motor is also setup at this point. With some experimentation with the program we set the speed to be about 10 to 15. A '\r' is for the carriage return.

```
public static String sForward = "mogo 1:10 2:10\r";
public static String sReverse = "mogo 1:-10 2:-10\r";
public static String sRight = "mogo 1:12 2:-12\r";
public static String sLeft = "mogo 1:-12 2:12\r";
public byte[] boutForward = new byte[sForward.Length];
public byte[] boutReverse = new byte[sReverse.Length];
public byte[] boutRight = new byte[sRight.Length];
public byte[] boutLeft = new byte[sLeft.Length];
```

A better way of programming these parameters would be to have the speed be set by the application instead of being hardcoded. One could create a library that offers this kind of flexibility.

The Sensor and Stop are simple one line commands. The Servo must rotate the sensor to look left, center, and right. Three servo commands are used to set the position of the servo. The servo can be set to any position in 360°. The starting point will be 0° (center) so -45° will be used to look left and 45° will be used to look right.

```
public static String sSTOP = "stop\r";
public static String sSensor = "sensor 0\r";
public static String sServo_m45 = "servo 1:-45\r";
public static String sServo_C = "servo 1:0\r";
public static String sServo_p45 = "servo 1:45\r";
public byte[] boutStop = new byte[sSTOP.Length];
public byte[] boutSensor = new byte[sSensor.Length];
public byte[] boutServo_m45 = new byte[sServo_m45.Length];
public byte[] boutServo_C = new byte[sServo_C.Length];
public byte[] boutServo_p45 = new byte[sServo_p45.Length];
```

When the application runs, all the commands are encoded to UTF8 constants. The Green LED is turned off to indicate that the application is go to run. For the 'mogo' commands, the COM2port.Write method sends the data, but a COM2port.Read is required to clear the buffer of the return 'ACK". If we leave the ACK in the buffer, the reading of the sensors will fail since the ACK will be read in and not the value from the sensor. The same dummy read goes for the Stop and Servo commands. Since we have to read sensor data from the Serializer, the binSensorArray is used for these dummy reads.

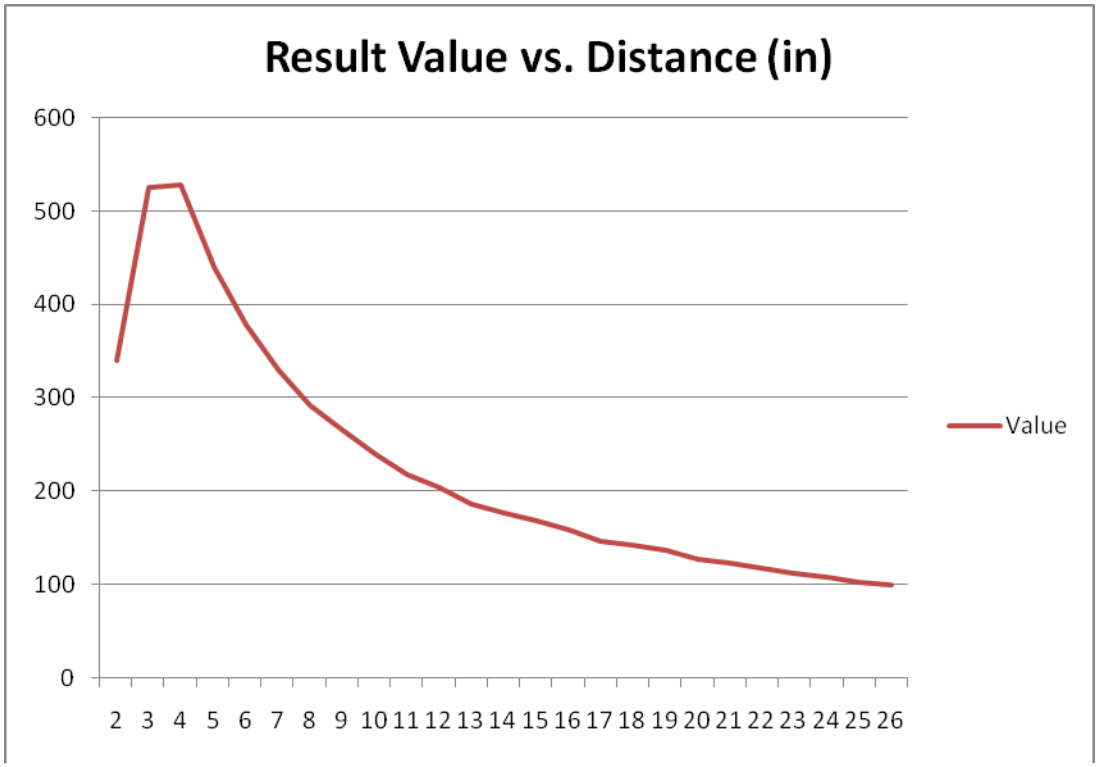
The rest of the code handles the basic operation that we want the robot to perform

- The main While-loop keeps the robot moving forward and turn the servo so a sensor reading can be taken.
- Scan() method – if an obstacle is detected, the robot stops and runs the Scan method. The Servo is rotated from left, center, and right. The robot will then turn based on the result from the sensor readings. The amount of turn left and right is controlled using the thread.sleep(int) to delay the return to the main While-loop. The table below has the logic used for the scan routine.

Motion	Left	Center	Right
Reverse	True	True	True
Right – long turn	True	True	False
Reverse	True	False	True
Right – short turn	True	False	False

Left – long turn	False	True	True
Reverse	False	True	False
Left – short turn	False	False	True
Forward	False	False	False

- SesnorIR() – Since reading the sensor is important to the application, a separate method is used. The StringToInt method is used to convert the output values from a byte array to a string that can be used for comparison. The Serializer performs the basic analog to digital conversion of the sensor. The closer the sensor is to an object the greater the return value. The farther the sensor is to an object, the lower the return value. With some testing you would see that these numbers change on an exponential curve. With some testing, 175 was determined to be the best value for indicating the sensor found an object and return a logic 'true'.



A separate program was created to test the sensor (Sharp GP2D12). Distance in inches is on the horizontal axis, and Serializer output is on the vertical axis. The farther out from the sensor the output becomes smaller. One thing to watch out for is when objects are very close because the tests show that the output value drops off. 175 seemed like the best choice for the right obstacle distance and handle this drop off close up. There is about 1 inch from the front of the sensor to the font of the Taxter so readings were taken starting at 2 inches.

- Reversechecking() – the final method hands a situation where the robot backs up more than twice. The robot could get stuck in a spot where it loops going back and forth, The Reversechecking method helps to determine that the robot is stuck so just spin around and start moving forward again.

Here is the whole code listing:

```
//
```

```
//Robot1
//SJJ Embedded Micro Solutions
//Copyright (c) 2004 - 2008 SJJ Micro Solutions, LLC. All Rights Reserved
//
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Hardware.SJJ;

namespace SJJ_MF_Console_Application
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(Resources.GetString(Resources.StringResources.String1));
            App myApp = new App();
            myApp.Run();
        }

    }


    public class App
    {
        static SerialPort.Configuration COM2Port_Config =   new
SerialPort.Configuration(SerialPort.Serial.COM2,SerialPort.BaudRate.Baud19200, true);
        public SerialPort COM2Port = new SerialPort(COM2Port_Config);


        public static String sForward = "mogo 1:10 2:10\r";
        public static String sReverse = "mogo 1:-10 2:-10\r";
        public static String sRight = "mogo 1:12 2:-12\r";
        public static String sLeft = "mogo 1:-12 2:12\r";
        public byte[] boutForward = new byte[sForward.Length];
        public byte[] boutReverse = new byte[sReverse.Length];
        public byte[] boutRight = new byte[sRight.Length];
        public byte[] boutLeft = new byte[sLeft.Length];

        public static String sSTOP = "stop\r";
        public static String sSensor = "sensor 0\r";
        public static String sServo_m45 = "servo 1:-45\r";
        public static String sServo_C = "servo 1:0\r";
        public static String sServo_p45 = "servo 1:45\r";
        public byte[] boutStop = new byte[sSTOP.Length];
        public byte[] boutSensor = new byte[sSensor.Length];
        public byte[] boutServo_m45 = new byte[sServo_m45.Length];
        public byte[] boutServo_C = new byte[sServo_C.Length];
        public byte[] boutServo_p45 = new byte[sServo_p45.Length];

        public int ReverseCheck = 0;

        public byte[] binSensorArray = new byte[16]; //256 was an arbitraty number. it
could have been 8 or 512



        public void Run()
        {
            System.Text.UTF8Encoding Encoding = new System.Text.UTF8Encoding();
            boutForward = Encoding.GetBytes(sForward);
            boutReverse = Encoding.GetBytes(sReverse);
            boutRight = Encoding.GetBytes(sRight);
            boutLeft = Encoding.GetBytes(sLeft);
            boutStop = Encoding.GetBytes(sSTOP);
            boutSensor = Encoding.GetBytes(sSensor);
            boutServo_m45 = Encoding.GetBytes(sServo_m45);
            boutServo_C = Encoding.GetBytes(sServo_C);
            boutServo_p45 = Encoding.GetBytes(sServo_p45);

            OutputPort myGreenLED;
```

```
myGreenLED = new OutputPort(Pins.GREEN_LED, false);

while (true)
{
    COM2Port.Write(boutForward, 0, boutForward.Length);
    Thread.Sleep(50);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);



    COM2Port.Write(boutServo_m45, 0, boutServo_m45.Length);
    Thread.Sleep(300);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
    if (SensorIR())
    {
        myGreenLED.Write(true);
        COM2Port.Write(boutStop, 0, boutStop.Length);
        Thread.Sleep(50);
        //Dummy read to clear buffer of ACK
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
        Scan();


        Thread.Sleep(1000);
        myGreenLED.Write(false);
    }

    COM2Port.Write(boutForward, 0, boutForward.Length);
    Thread.Sleep(50);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);


    COM2Port.Write(boutServo_C, 0, boutServo_C.Length);
    Thread.Sleep(300);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
    if (SensorIR())
    {
        myGreenLED.Write(true);
        COM2Port.Write(boutStop, 0, boutStop.Length);
        Thread.Sleep(50);
        //Dummy read to clear buffer of ACK
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
        Scan();
        Thread.Sleep(1000);


        myGreenLED.Write(false);
    }

    COM2Port.Write(boutForward, 0, boutForward.Length);
    Thread.Sleep(50);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);


    COM2Port.Write(boutServo_p45, 0, boutServo_p45.Length);
    Thread.Sleep(300);
    //Dummy read to clear buffer of ACK
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
    if (SensorIR())
    {
        myGreenLED.Write(true);
        COM2Port.Write(boutStop, 0, boutStop.Length);
        Thread.Sleep(50);
        //Dummy read to clear buffer of ACK
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
        Scan();

        Thread.Sleep(1000);
```

```
                myGreenLED.Write(false);

            }


        }

    }

    public bool SensorIR()
    {
        System.Text.UTF8Encoding Encoding = new System.Text.UTF8Encoding();
        boutSensor = Encoding.GetBytes(sSensor);

        COM2Port.Write(boutSensor, 0, boutSensor.Length);
        Thread.Sleep(50);
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);

        int finalNum = StringToInt(binSensorArray);

        //Change Sensor Sensitivity
        if (finalNum > 175)
        {
            return true;
        }
        else
        {
            return false;
        }

    }

    public void Scan()
    {
        System.Text.UTF8Encoding Encoding = new System.Text.UTF8Encoding();
        boutForward = Encoding.GetBytes(sForward);
        boutReverse = Encoding.GetBytes(sReverse);
        boutRight = Encoding.GetBytes(sRight);
        boutLeft = Encoding.GetBytes(sLeft);
        boutStop = Encoding.GetBytes(sSTOP);
        boutSensor = Encoding.GetBytes(sSensor);
        boutServo_m45 = Encoding.GetBytes(sServo_m45);
        boutServo_C = Encoding.GetBytes(sServo_C);
        boutServo_p45 = Encoding.GetBytes(sServo_p45);

        bool ScanLeft = false;
        bool ScanCenter = false;
        bool ScanRight = false;


        COM2Port.Write(boutServo_m45, 0, boutServo_m45.Length);
        Thread.Sleep(500);
        //Dummy read to clear buffer of ACK
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
        if (SensorIR())
        {
            ScanLeft = true;
        }


        COM2Port.Write(boutServo_C, 0, boutServo_C.Length);
        Thread.Sleep(500);
        //Dummy read to clear buffer of ACK
        COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
        if (SensorIR())
        {
            ScanCenter = true;
        }



        COM2Port.Write(boutServo_p45, 0, boutServo_p45.Length);
```

```
            Thread.Sleep(500);
            //Dummy read to clear buffer of ACK
            COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
            if (SensorIR())
            {
                ScanRight = true;
            }

            if (ScanLeft == true && ScanCenter == true && ScanRight == true)
            {
                COM2Port.Write(boutReverse, 0, boutReverse.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                ReverseCheck += 1;
                Reversechecking();
                Scan();


            }
            else if (ScanLeft == true && ScanCenter == true && ScanRight == false)
            {
                COM2Port.Write(boutRight, 0, boutRight.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(1000);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);

            }
            else if (ScanLeft == true && ScanCenter == false && ScanRight == true)
            {

                COM2Port.Write(boutReverse, 0, boutReverse.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                ReverseCheck += 1;
                Reversechecking();
                Scan();

            }
            else if (ScanLeft == false && ScanCenter == true && ScanRight == true)
            {
                COM2Port.Write(boutLeft, 0, boutLeft.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(1000);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);

            }
            else if (ScanLeft == false && ScanCenter == false && ScanRight == true)
            {
                COM2Port.Write(boutLeft, 0, boutLeft.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
```

```
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);

            }
            else if (ScanLeft == true && ScanCenter == false && ScanRight == false)
            {
                COM2Port.Write(boutRight, 0, boutRight.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);

            }
            else if (ScanLeft == false && ScanCenter == false && ScanRight == false)
            {
                COM2Port.Write(boutForward, 0, boutForward.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
            }
            else if (ScanLeft == false && ScanCenter == true && ScanRight == false)
            {
                COM2Port.Write(boutReverse, 0, boutReverse.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(500);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                ReverseCheck += 1;
                Reversechecking();
                Scan();
            }



        }
        //-------------------------------------------------------------------------
        // Assumption: number field in byte string has ASCII space, 0x20, after it:
        // <preampble characters><ASCII digits><ASCII space><trailer characters>
        // Number is always positive
        //------_-------------------------------------------------------------------

        public int StringToInt(byte[] AsciiString)
        {
            int iReturnVal = 0;
            int idx;
            int iPwrTen;   //Conversion multiplier

            for (idx = 0; AsciiString[idx] != 0x20; idx++)
            {
                ;   //Do nothing
            }

            idx--;  //Back up to units digit
            for (iPwrTen = 1; ((AsciiString[idx] >= 0x30) && (AsciiString[idx] <= 0x39));
idx--)
```

```
            {
                iReturnVal += (AsciiString[idx] & 0x0F) * iPwrTen;  //Mask off upper
ASCII nibble and convert to power of ten
                iPwrTen *= 10;
            }

            return iReturnVal;
        }


        //need to know if we backed up more than once and spin around to avoid
        //being stuck. The global ReverseCheck int is used
        public void Reversechecking()
        {

            System.Text.UTF8Encoding Encoding = new System.Text.UTF8Encoding();
            boutRight = Encoding.GetBytes(sRight);
            boutLeft = Encoding.GetBytes(sLeft);
            boutStop = Encoding.GetBytes(sSTOP);

            if (ReverseCheck >= 2)
            {

                //Spin around
                COM2Port.Write(boutRight, 0, boutRight.Length);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                Thread.Sleep(1800);

                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dummy read to clear buffer of ACK
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length, 20);
                ReverseCheck = 0;
            }

        }

    }
}
```

The code is not the prettiest we've put together, but it demonstrates basic serial port communications with another platform. Now let's re-do the application using a managed code driver (.NET library) for the Serializer.

```
//Robot3
//SJJ Embedded Micro Solutions
//Copyright (c) 2004 - 2008 SJJ Micro Solutions, LLC. All Rights Reserved

using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT.Hardware.SJJ;
using RoboticsConnection.SerializerLibMF;

namespace SJJ_MF_Console_Application
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(Resources.GetString(Resources.StringResources.String1));
            App myApp = new App();
            myApp.Run();
        }

    }
```

```
public class App
{
    private static PidMotorController pmc = new PidMotorController();
    private static ServoMotorController sc = new ServoMotorController();
    private static AnalogSensor sensor = new AnalogSensor();

    public int ReverseCheck = 0;

    public void Run()
    {

        OutputPort myGreenLED;
        myGreenLED = new OutputPort(Pins.GREEN_LED, false);

        int mySensorOut = 0;

        while (true)
        {

            pmc.TravelAtSpeed(10, 10);
            sc.SetPosition(1, -45);
            Thread.Sleep(500);
            mySensorOut = sensor.Reading(0);
            Thread.Sleep(100);
            if (mySensorOut > 100)
            {
                myGreenLED.Write(true);
                pmc.Stop();
                Scan();
                Thread.Sleep(500);
                myGreenLED.Write(false);

            }
            Thread.Sleep(500);
            pmc.TravelAtSpeed(10, 10);
            sc.SetPosition(1, 0);
            Thread.Sleep(500);
            mySensorOut = sensor.Reading(0);
            Thread.Sleep(100);
            if (mySensorOut > 100)
            {
                myGreenLED.Write(true);
                pmc.Stop();
                Scan();
                Thread.Sleep(500);
                myGreenLED.Write(false);

            }
            Thread.Sleep(500);
            pmc.TravelAtSpeed(10, 10);
            sc.SetPosition(1, 45);
            Thread.Sleep(500);
            mySensorOut = sensor.Reading(0);
            Thread.Sleep(100);
            if (mySensorOut > 100)
            {
                myGreenLED.Write(true);
                pmc.Stop();
                Scan();
                Thread.Sleep(500);
                myGreenLED.Write(false);

            }
            Thread.Sleep(500);

        }
    }

    public void Scan()
    {

        bool ScanLeft = false;
```

```
bool ScanCenter = false;
bool ScanRight = false;

Thread.Sleep(500);
sc.SetPosition(1, -45);
if (sensor.Reading(0) > 100)
{
    ScanLeft = true;
}

Thread.Sleep(500);
sc.SetPosition(1, 0);
if (sensor.Reading(0) > 100)
{
    ScanCenter = true;
}

Thread.Sleep(500);
sc.SetPosition(1, 45);
if (sensor.Reading(0) > 100)
{
    ScanRight = true;
}
Thread.Sleep(500);

if (ScanLeft == true && ScanCenter == true && ScanRight == true)
{
    pmc.TravelAtSpeed(-10, -10);
    Thread.Sleep(500);
    pmc.Stop();
    Thread.Sleep(50);
    ReverseCheck += 1;
    Reversechecking();
    Scan();
}
else if (ScanLeft == true && ScanCenter == true && ScanRight == false)
{
    pmc.TravelAtSpeed(12, -12);
    Thread.Sleep(1000);
    pmc.Stop();
    Thread.Sleep(50);
}
else if (ScanLeft == true && ScanCenter == false && ScanRight == true)
{
    pmc.TravelAtSpeed(-10, -10);
    Thread.Sleep(500);
    pmc.Stop();
    Thread.Sleep(50);
    ReverseCheck += 1;
    Reversechecking();
    Scan();
}
else if (ScanLeft == false && ScanCenter == true && ScanRight == true)
{
    pmc.TravelAtSpeed(-12, 12);
    Thread.Sleep(1000);
    pmc.Stop();
    Thread.Sleep(50);
}
else if (ScanLeft == false && ScanCenter == false && ScanRight == true)
{
    pmc.TravelAtSpeed(-12, 12);
    Thread.Sleep(500);
    pmc.Stop();
    Thread.Sleep(50);
}
else if (ScanLeft == true && ScanCenter == false && ScanRight == false)
{
    pmc.TravelAtSpeed(12, -12);
    Thread.Sleep(500);
    pmc.Stop();
    Thread.Sleep(50);
}
```

```
            else if (ScanLeft == false && ScanCenter == false && ScanRight == false)
            {
                pmc.TravelAtSpeed(10, 10);
                Thread.Sleep(500);
                pmc.Stop();
                Thread.Sleep(50);
            }
            else if (ScanLeft == false && ScanCenter == true && ScanRight == false)
            {
                pmc.TravelAtSpeed(-10, -10);
                Thread.Sleep(500);
                pmc.Stop();
                Thread.Sleep(50);
                ReverseCheck += 1;
                Reversechecking();
                Scan();
            }

        }


        public void Reversechecking()
        {

            if (ReverseCheck >= 2)
            {

                //Spin around
                pmc.TravelAtSpeed(-12, 12);
                Thread.Sleep(1800);
                pmc.Stop();
                ReverseCheck = 0;

            }

        }


    }
}
```

Developed by RoboticsConnection, the managed code library simplifies the commands, and makes the code much shorter and easier to debug. The library supports the different sensors and encoders so you can take full advantage of the Serializer capability. Being able to create reusable code is one of the many advantages of .NET programming.

To make the whole robot more robust more sensors and a different logic detection scheme could be added. With the ability to control the robotic motion and sensors via a single serial channel, the rest of the iPac-9302 Deluxe's IO (GPIOs, PWM, ADC, SPI, SD/MMC, Ethernet) is free for other experiment use. With a solid platform for motion and obstacle sensors, you can build-up a robot to target a specific task.

Windows is a registered trademark of Microsoft Corporation.