Robot Demo Featuring the EDKPlus with .NET Micro Framework 4.0 and the Traxster II

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

May 2010

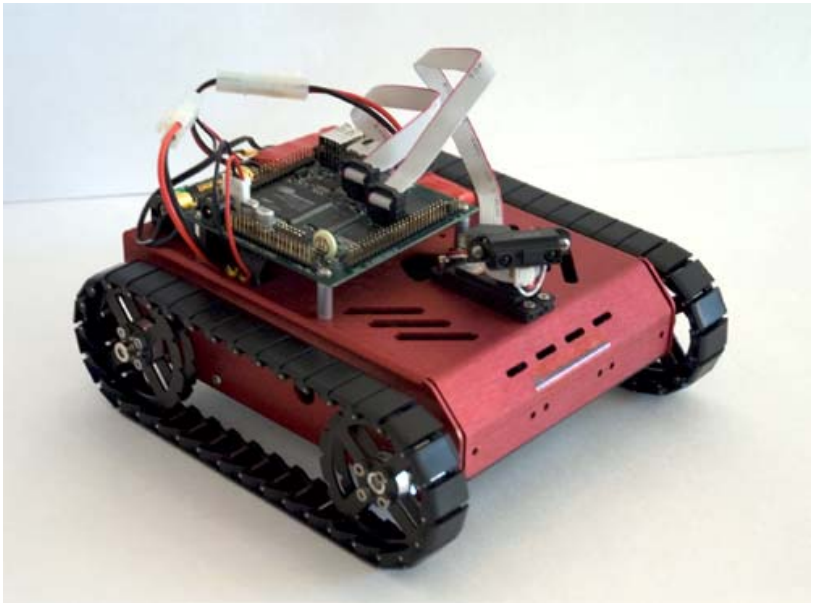
Introduction

The EDKPlus with .NET Micro Framework V4.0 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

A couple of years ago as a demonstration vehicle, we posted an article about the EDKPlus and the original Traxster I from RoboticsConnection.com. Since then, we have been working on an update. RoboticsConnection developed the Traxster II, which provides a much more flexible platform for additional sensors and payload.

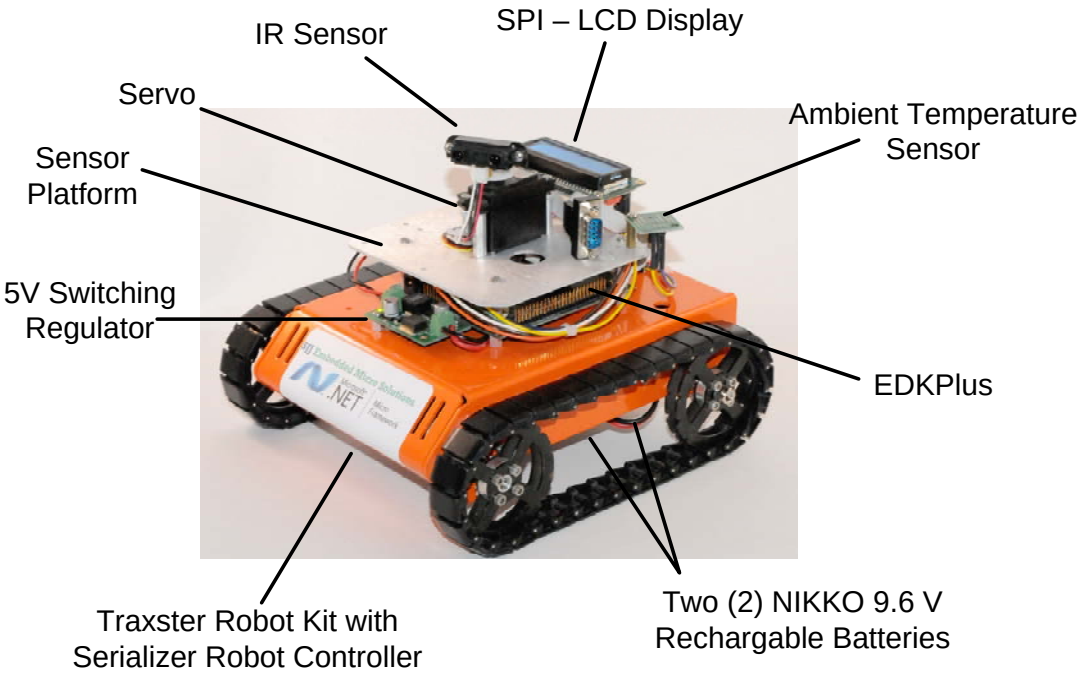
For those of you wondering, the Microsoft® .NET Micro Framework is still alive and well (www.netmf.com), and it exists as an open source project. With the release of .NET MF V4.0, Visual C# Express 2008, which is a free download, is now supported.

The original integration of the EDKplus with the Traxster I robot had a single IR sensor mounted on a servo. The robot's goal was to always move forward and avoid obstacles. Using the IR sensor, it scanned left, center, and right to see if there are any obstacles. If an obstacle was detected, the robot would stop scan again, and turn in a direction that was not blocked. If all directions were blocked, the robot would back up and scan again. If the backup happened more than 2 times, it would reverse direction and start the process in the opposite direction.



The Original Traxster I with EDKPlus on top

For the Traxster II, we are keeping with the same idea, but expanding support to include an ambient temperature sensor and a 2-line LCD display.



Traxster II provides a more robust / dynamic robotics platform

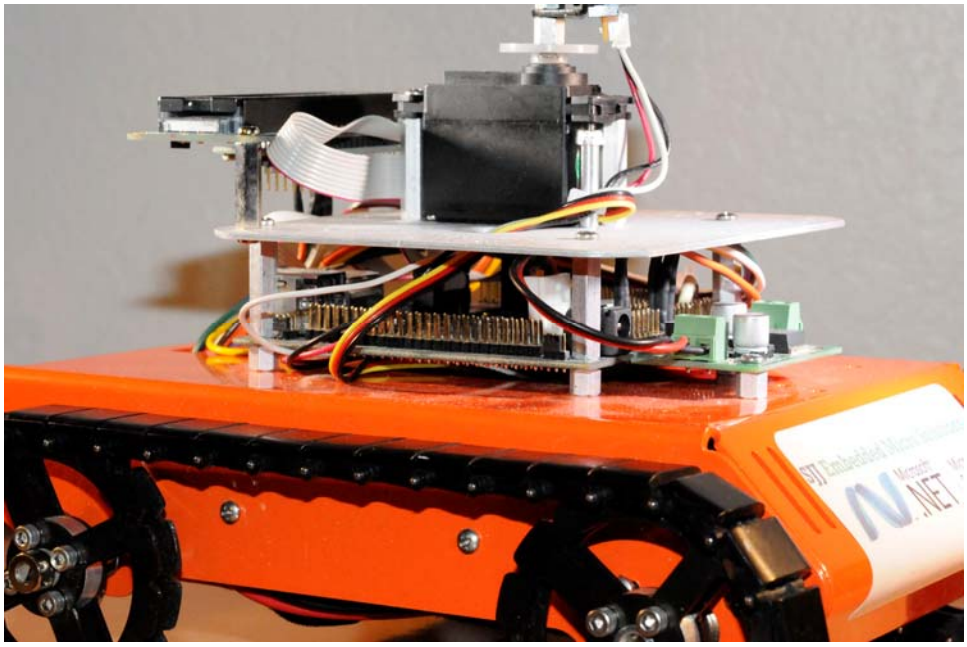
The LCD display provides an on board indicator of the robots current activities. When the robot stops for an obstacle, a temperature reading is made and the result is displayed on the display.

Robot Setup

The Traxster II was setup first following the instructions that came with the kit.

The EDKPlus (iPac-9302 Deluxe) has two COM ports. COM1 is always used to download and debug the application. COM2 is used by the .NET Micro Framework application for standard RS-232 communication. After building the Traxster II platform, we mounted the iPac-9302 on top of the Traxster II along with the 5V switching regulator. A serial ribbon pigtail was connected from the iPac's COM2 header to the Serial RS-232 Interface module on the Serializer board. The Serial interface module already has the signals set up so you can plug the RS-232 directly into the connector. No crossover or null modem interfaces are needed. A second pigtail 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. A second 9.6V battery feeds the voltage regulator which powers the iPac-9302. A floppy disk-style power connector was used to plug into the iPac-9302's floppy power connector. Both batteries are connected through individual switches: one switch for the iPac-9302 and the other for the Serializer.



Floppy disk-style power connector powers the EDKPlus from the voltage regulator.

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

The sensor platform was placed above the iPac-9302 using stand-offs. The servo with the IR sensor was placed in the middle. The LCD and the temperature sensor were mounted in the back using stand-offs. A hole had to be drilled into the sensor platform to mount the LCD.

The LCD display was connected to the iPac-9302 SPI port. Information about the SPI-LCD can be found in the EDK Developer's Guide. The temperature sensor was connected to the Serializer. The temperature sensor supports 5V power with up to a 5V output signal. For this example, connection to the Serializer was easier. The ambient temperature sensor could also be connected directly to one of the iPac-9302's ADC channels. This could be accomplished using either a different 3.3V ambient temperature sensor or using this same temperature sensor with a precision resistor divider to reduce the output signal range to 3.3 V.

The Main Program

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. Because of the back and forth communication, the program has to perform a read after write to clear out the buffer.

Setting up the serial port (COM2) is one of the first things that need to happen in the application. .NET Micro Framework V4.0 changes the way serial ports are setup to be more aligned to .NET Framework and improve portability of code between .NET Micro Framework platforms and Windows platforms. Setup and instantiation are performed at the same time like .NET Framework:

```
public SerialPort COM2Port = new SerialPort(SerialPorts.COM2,
(int)BaudRates.Baud19200, Parity.None, 8);
```

The baud rate to communicate with the Serializer is 9600.

We also set up the SPI port and implement the managed code SPI_LCDDriver from the EDKPlus:

```
static SPI.Configuration mySPIPortSettings = new
SPI.Configuration(Pins.GPIO_NONE, false, 0, 0, false, true, 50,
SPI.SPI_module.SPI1);
SPI mySPIPort = new SPI(mySPIPortSettings);
SPI_LCDDriver myLCDDriver = new SPI_LCDDriver();
```

The Green LED is also set up:

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

The Serializer 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. Constants are also needed for Stop, Servo motion, and the Sensor. Finally we will define the byte arrays for each command.

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

```
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 integer degrees within the 360° circle. 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];
```

The temperature sensor is connected to sensor port 4.

```
public static String tempSensor = "sensor 4\r";
public byte[] boutTempSensor = new byte[tempSensor.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 good to run. The word “Ready” is displayed on the LCD. For the “mogo” commands, the COM2port.Write method sends the data, but a COM2port.Read is required to clear the buffer of the returned “ACK”. If we leave the ACK in the buffer, reading of the sensor will attempt to read the ACK as a returned value from the sensor, resulting in an error. We call these dummy reads, since they don’t do anything, but you could perform error checking for NACKs. 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 all dummy reads.

```
public byte[] binSensorArray = new byte[16];
```

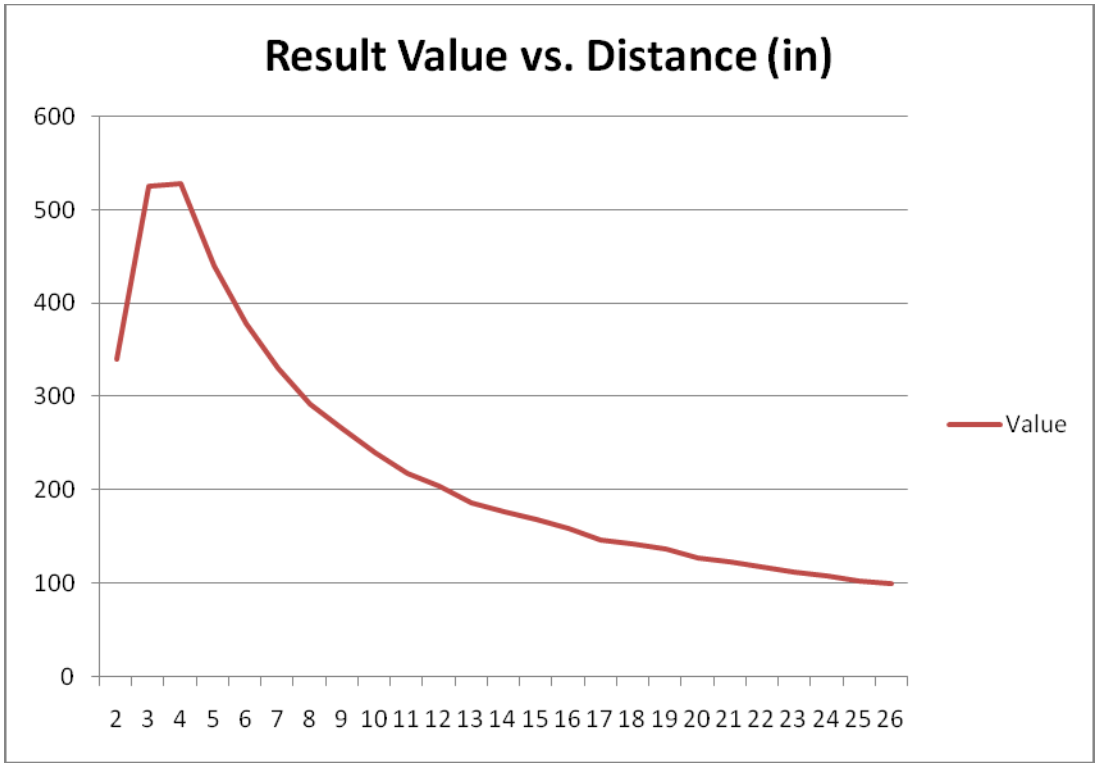
The rest of the code handles the basic operation that we want the robot to perform:

- The main While-loop keeps the robot moving forward and turns the servo to 3 different positions where a sensor reading can be taken.
- Obstruction() - If the sensors indicate an obstruction, the Obstruction method is found. The code is the same whether looking left, center, or right. A separate method simplifies the code. If the sensor indicates an obstruction, the motors are stopped. A call to the Scan() method is made, where the robot will turn to avoid the obstacle. The Green LED and the word “Moving” on the LCD will be the last calls made before returning to the main While-loop.
- Scan() method – if an obstacle is detected, the robot stops and runs the Scan method. The Servo is rotated from left, center, and right, where 3 IR readings are made. The robot will then turn based on the result from the 3 sensor readings. The amount of turn left or right is controlled by engaging the drive motors and using the thread.sleep(int) to delay before a Stop is called and execution is returned 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 for IR reading. 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 from an object, the lower the return value. With some testing you would see that these numbers change on an exponential

curve. The readings were made with a fully charged battery at around 25°C. With some testing, the best values are between 120 and 175. If the value is greater than the set value, a logic 'true' is returned from the method.



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 the object is from the sensor, the smaller the output becomes. One thing to watch out for is when objects are very close, because the tests show that the output value increases as distance decreases until it gets to a point where the output peaks and then decreases as the distance continues to decrease. 175 seemed like the best choice for the right obstacle distance to handle this drop off at close distances. There is about 1 inch from the front of the sensor to the front of the Traxster, so readings were taken starting at 2 inches.

- TempRead() – this new method reads the ambient temperature sensor. The StringToInt is used to convert the string data to a number that can be used to calculate the final temperature. Both °C and °F are displayed on the LCD.
- Reversechecking() – the final method handles 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 and just to spin around and start moving forward again.

Here is the whole code listing:

```
//
//TraxsterPlus2
//SJJ Embedded Micro Solutions
//Copyright (c) 2004 - 2010 SJJ Micro Solutions, LLC. All Rights Reserved
//

using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using System.Threading;
```

```
using Microsoft.SPOT.Hardware.SJJ;
using Microsoft.SPOT.Hardware.SJJ.SPI_LCD_Driver;
using System.IO.Ports;

namespace TraxsterPlus2
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(Resources.GetString(Resources.StringResources.String1));
            App myApp = new App();
            myApp.Run();
        }

        public class App
        {
            //configure the SPI port, the LCDDriver, the Serial port and the Green LED
port.
            static SPI.Configuration mySPIPortSettings = new
SPI.Configuration(Pins.GPIO_NONE, false, 0, 0, false, true, 50, SPI.SPI_module.SPI1);
            SPI mySPIPort = new SPI(mySPIPortSettings);
            SPI_LCDDriver myLCDDriver = new SPI_LCDDriver();

            public SerialPort COM2Port = new SerialPort(SerialPorts.COM2,
(int)BaudRates.Baud19200, Parity.None, 8);

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

            //Setup the Serializer commands
            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 static String tempSensor = "sensor 4\r";
            public byte[] boutTempSensor = new byte[tempSensor.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()
            {
                //Open the COM2 port
                COM2Port.ReadTimeout = 10;
                COM2Port.Open();

                //Encode the Serializer commands to byte arrays
                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);

            //Ready to go!
            mySPIPort.Write(myLCDDriver.LCDClear());
            Thread.Sleep(6);
            mySPIPort.Write(myLCDDriver.LCDLine1("Ready"));
            Thread.Sleep(6);

            //This is the main While-loop that keeps the robot moving forward
            while (true)
            {
                COM2Port.Write(boutForward, 0, boutForward.Length);
                Thread.Sleep(100);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);

                mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Left"));
                COM2Port.Write(boutServo_m45, 0, boutServo_m45.Length);
                Thread.Sleep(300);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                if (SensorIR())
                {

                    Obstruction();

                }

                COM2Port.Write(boutForward, 0, boutForward.Length);
                Thread.Sleep(100);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);

                mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Center"));
                COM2Port.Write(boutServo_C, 0, boutServo_C.Length);
                Thread.Sleep(300);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                if (SensorIR())
                {
                    Obstruction();

                }

                COM2Port.Write(boutForward, 0, boutForward.Length);
                Thread.Sleep(100);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);

                mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Right"));
                COM2Port.Write(boutServo_p45, 0, boutServo_p45.Length);
                Thread.Sleep(300);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                if (SensorIR())
                {

                    Obstruction();

                }
                Thread.Sleep(50);

            }


        }

        //An obstruction was found, stop the motors and scan to find
        // a clear path
        public void Obstruction()
        {
```

```
                myGreenLED.Write(true);
                mySPIPort.Write(myLCDDriver.LCDLine1("Obstruction Found"));
                Thread.Sleep(6);
                COM2Port.Write(boutStop, 0, boutStop.Length);
                Thread.Sleep(50);
                //Dumby read to clear buffer of ACK from
                COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                Scan();
                Thread.Sleep(1000);
                myGreenLED.Write(false);
                mySPIPort.Write(myLCDDriver.LCDClear());
                Thread.Sleep(6);
                mySPIPort.Write(myLCDDriver.LCDLine1("Moving"));
                Thread.Sleep(6);

            }

            //Read the IR sensor. If the resulting value is greater
            //then the sensitivity value return true otherwise return 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);

                int finalNum = StringToInt(binSensorArray);

                mySPIPort.Write(myLCDDriver.LCDClear());
                Thread.Sleep(6);
                mySPIPort.Write(myLCDDriver.LCDLine1("IR Value: " +
finalNum.ToString()));

                //The value can be changed for better sensitivity.
                if (finalNum > 135)
                {
                    return true;
                }
                else
                {
                    return false;
                }

            }

            //The Scan() method determines which direction the robot should turn.
            //3 IR readings are made - left, center, right. Based on the readings
            //The robot will make a turn to avoid the obstruction.
            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;

                TempRead();

                mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Left"));
                COM2Port.Write(boutServo_m45, 0, boutServo_m45.Length);
                Thread.Sleep(500);
```

```
//Dumby read to clear buffer of ACK from
COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
if (SensorIR())
{
    ScanLeft = true;
}

mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Center"));
COM2Port.Write(boutServo_C, 0, boutServo_C.Length);
Thread.Sleep(500);
//Dumby read to clear buffer of ACK from
COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
if (SensorIR())
{
    ScanCenter = true;
}


mySPIPort.Write(myLCDDriver.LCDLine2("Scanning: Right"));
COM2Port.Write(boutServo_p45, 0, boutServo_p45.Length);
Thread.Sleep(500);
//Dumby read to clear buffer of ACK from
COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
if (SensorIR())
{
    ScanRight = true;
}

if (ScanLeft == true && ScanCenter == true && ScanRight == true)
{

    //Obstruction ahead - turn around
    COM2Port.Write(boutReverse, 0, boutReverse.Length);
    Thread.Sleep(50);
    //Dumby read to clear buffer of ACK from
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
    Thread.Sleep(500);

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


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

    //turn right
    COM2Port.Write(boutRight, 0, boutRight.Length);
    Thread.Sleep(50);
    //Dumby read to clear buffer of ACK from
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
    Thread.Sleep(1000);

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

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

    //Obstruction ahead - reverse
    COM2Port.Write(boutReverse, 0, boutReverse.Length);
    Thread.Sleep(50);
    //Dumby read to clear buffer of ACK from
    COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
    Thread.Sleep(500);
```

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

        }
        else if (ScanLeft == false && ScanCenter == true && ScanRight == true)
        {
            //Turn Left
            COM2Port.Write(boutLeft, 0, boutLeft.Length);
            Thread.Sleep(50);
            //Dumby read to clear buffer of ACK from
            COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
            Thread.Sleep(1000);

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

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

            //Turn Left
            COM2Port.Write(boutLeft, 0, boutLeft.Length);
            Thread.Sleep(50);
            //Dumby read to clear buffer of ACK from
            COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
            Thread.Sleep(500);

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

        }
        else if (ScanLeft == true && ScanCenter == false && ScanRight == false)
        {
            //turn right
            COM2Port.Write(boutRight, 0, boutRight.Length);
            Thread.Sleep(50);
            //Dumby read to clear buffer of ACK from
            COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
            Thread.Sleep(500);

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

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

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

```
                    //Dumby read to clear buffer of ACK from
                    COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                    Thread.Sleep(500);

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



            }
            //-------------------------------------------------------------------------
----------------
            // Converts Strings to an Integer values
            // Used for both IR and Temp Readings
            // 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;
            }

            //Sends Temperature Reading to the LCD display
            public void TempRead()
            {

                Thread.Sleep(500);

                System.Text.UTF8Encoding Encoding = new System.Text.UTF8Encoding();
                boutTempSensor = Encoding.GetBytes(tempSensor);

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

                int finalNum = StringToInt(binSensorArray);

                float tempC = ((((float)finalNum * 5) / 1024) * 100) - 273;
                float tempF = (tempC * (float)1.8) + 32;

                mySPIPort.Write(myLCDDriver.LCDLine1("Temp: " + tempF.ToString() + "F / "
+ tempC.ToString() + "C"));

                Thread.Sleep(1000);
```

```
            }


            //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);
                    Thread.Sleep(50);
                    //Dumby read to clear buffer of ACK from
                    COM2Port.Read(binSensorArray, 0, binSensorArray.Length);
                    Thread.Sleep(1800);

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

                }


            }


        }


    }
}
```

The different motor and sensor commands could be wrapped up in a single managed code library, but the implementation above is basically straight forward. We will leave this to you to build on and create your own solution.

The EDKPlus and Traxster II's new implementation allows for more I/O like GPS, cameras, wireless connectivity, etc. to be added to create an application specific robot.

Windows is a registered trademark of Microsoft Corporation.