Interfacing with the Sick LMS-200

The SICK Laser Measurement Sensor (LMS) 200 is an extremely accurate distance measurement sensor that is quickly becoming a staple of the robotics community. The LMS 200 can easily be interfaced through RS-232 or RS-422, providing distance measurements over a 180 degree area up to 80 meters away. As this sensor grows in popularity, the ability to program code to interact with this sensor becomes an essential skill for all roboticists.

INTRODUCTION

The purpose of this tutorial is to teach you how to operate the SICK LMS-200 and construct a basic Visual Basic program to acquire data. This tutorial will tell you how the SICK works, how to set up the SICK hardware, how to operate the provided software and how to make a VB6 program to acquire data. This tutorial assumes you have basic hardware skills such as soldering and basic programming skills with Visual Basic 6.0.

The rest of the tutorial is presented as follows:

BACKGROUND

SICK is a german based company that specializes in industrial sensor technologies. SICK specifically specializes in optical solutions for automation, safety systems, automatic identification, and analysis and process measurement. SICK offers a variety of scanning laser

range finders, some of the more popular ones being the LMS 200, the LMS 291, and the PLS 101/201. The table below explains the differences between these sensors.

LMS 200

Part of SICK's Laser Measurement Sensor (LMS) series. In ideal conditions, the LMS 200 is capable of measuring out to 80m over a 180° arc. For an object with only 10% reflectivity (such as matt black cardboard), the LMS 200 can measure out to 10m. The sensor is best for indoor use as it can be dazzled by sunlight (causing it to give erroneous readings).

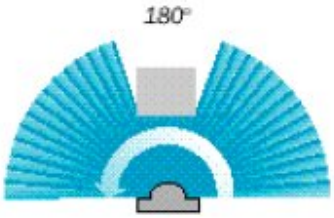
LMS 291

Another of SICK's LMS series, this sensor gained fame in the 2005 DARPA Grand Challenge. The LMS 291 is very similar to the LMS 200, with measuring range out to 80m over a 180° arc. However, the LMS 291 can see an object with 10% reflectivity out to 30m. The LMS 291 also has fog correction, allowing it to work in harsher conditions. The downside is that its measurements carry greater error than the LMS 200.

PLS 101/201

The Proximity Laser Scanner (PLS) series of SICK's are meant for the machine safety industry. Similar to the LMS sensors, the PLS 101 and 201 sweep a 180° arc out to 50m. However, the PLS sensors have built in hardware that allows them to control machinery. Through software, the user can define a protective field within the sensor's field of view. When anything enters the protective field, the sensor shuts down the machine.

As mentioned, this tutorial will focus on operating the LMS 200. Most information provided in this tutorial should be applicable to the other sensors, with some minor changes in code. All of the sensors operate by shining a laser off of a rotating mirror. As the mirror spins, the laser scans 180°, effectively creating a fan of laser light as shown in the picture below.



Any object that breaks this fan reflects laser light back to the sensor. The distance is calculated based on how long the laser takes to bounce back to the sensor. The specs for the LMS 200 are given below:

Model Name	LMS 200-30106
Part Number	1015850
Field of view:	180°
Angular resolution:	1 ... 0.25°
Response time:	13 ... 53 ms
Resolution:	10 mm
Systematic error:	+/- 15 mm
Statistical error (1 sigma):	5 mm
Laser class:	1
Enclosure rating:	IP 65
Ambient operating temperature:	0°C ... +50°C
Scanning range:	80 m
Data interface:	RS-232, RS-422
Data transmission rate:	9.6 / 19.2 / 38.4 / 500 kBps
Switching outputs:	3 x PNP
Supply voltage:	24 V DC +/- 15%
Power consumption:	20 W
Storage temperature:	-30°C ... +70 °C
Weight:	4.5 kg
Dimensions (L x W x H):	156 x 155 x 210 mm

MATERIALS

To complete this tutorial, you will need the following:

QTY	DESCRIPTION	PART #	COST	SUPPLIER
1	LMS 200	1015850	$5,771.43 (2006)	**www.sick.com**
1	Power Supply	174844	$18.95 (2006)	**www.jameco.com**

1	Serial Cable	199638	$4.49 (2006)	**www.jameco.com**
Computer running at least MS Windows 98 or XP with serial port and Visual Basic 6.0				

HARDWARE

To make the SICK operational, it must be wired for power and communication. On the back of the SICK are two black connectors. After unscrewing the socket head screws, the connector should appear as in the picture below:

On the bottom of these connectors you will see a 9-pin connector that looks like a serial port connector. The connector with the female end is for power. The connector with the male end is for communications. Unscrew the cap on the top of the connector. This is where the cable should be fed in. Unscrew the phillips head screws on the bottom of the connector. Feed the wire through the connector as shown below:

The cable can now be soldered to the back of the 9-pin connector. This tutorial will focus on commincating with the SICK via RS232 (which is standard for a PC serial port). As such, the power and serial cable should be wired as follows:

POWER CONNECTIONS

Pin No.	Signal designation	Input/ Output
1	**GND_EXT** (ground)	**– Supply**
2	Restart	-
3	**VCC_EXT** (24V DC +- 15%)	**+ Supply**
4	Not connected	-
5	OUT C (for field detection)/ weak signal	-
6	Not connected	-
7	Not connected	-
8	OUT B (for field detection)	-
9	OUT A (for field detection)	-

COMMUNICATIONS CONNECTIONS (for RS232)

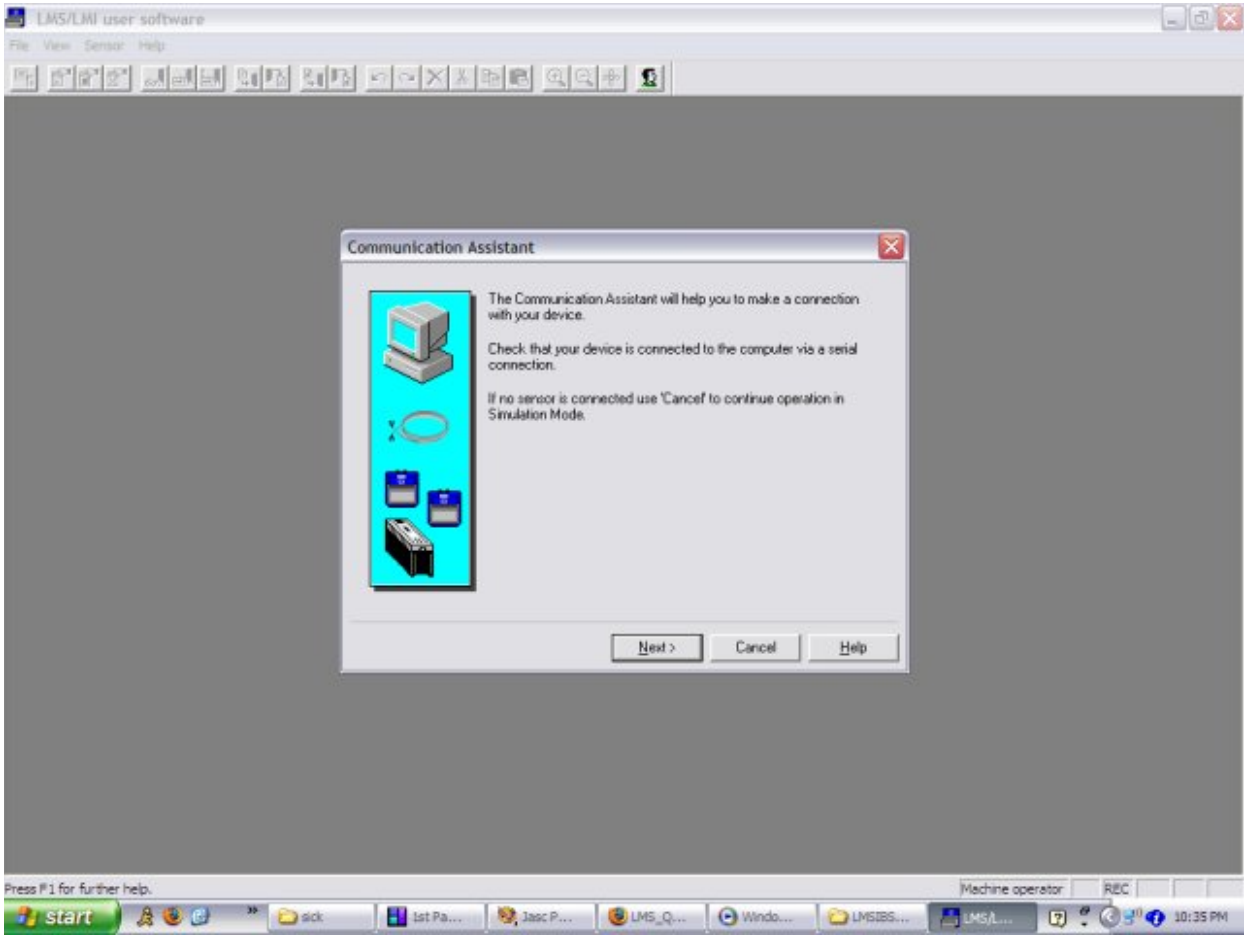
LMS			PC	
Signal designation	**PIN No.**		**PIN No.**	**Signal designation**
Not connected	1		1	NC
RxD	2		2	RxD
TXD	3		3	TXD
Not connected	4		4	Not connected
GND	5		5	GND
Not connected	6		6	Not connected
Not connected	7		7	Not connected
Not connected	8		8	Not connected
Not connected	9		9	Not connected

Note that the transmit and receive pins are crossed from the PC to the SICK. In other words, the transmit from the SICK goes into the receive of the PC, and vice versa. After soldering the cables to the 9-pin connectors, reassemble the black connectors and reattach them to the back of the SICK. You're now ready to start working with the SICK.

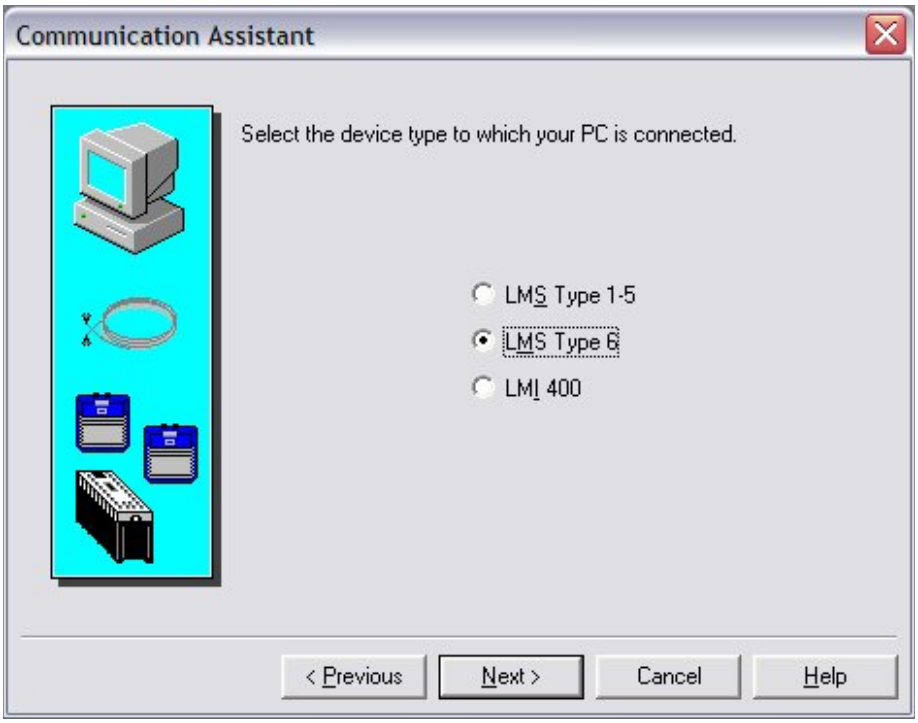
SOFTWARE

Your SICK should have come with a cd containing software and manuals for the LMS 200. Insert the cd into your cd drive and locate the file "INSTALL.EXE" (should be in a path such as D:\CD LMS2xx Version OB70\LMSIBS User Software_Benutzersoftware V5.11). Run this file to install the LMS software.

After installation, navigate to the LMS software and run the program (**Start Menu -> Programs -> SICK Applications -> LMS-LMI400 user software V 5.10**). A window should appear titled "Communication Assistant". If this window does not appear, go to **Sensor -> Interface Assistant**. Your screen should now look like this:

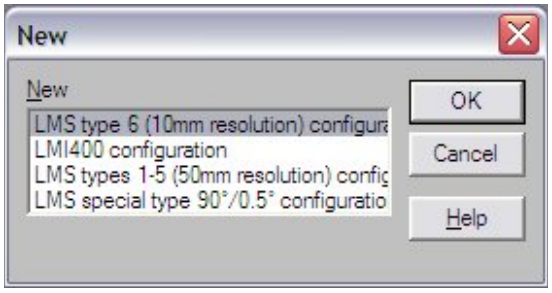


Click "Next". Select "LMS Type 6" from the radio buttons and click "Next".

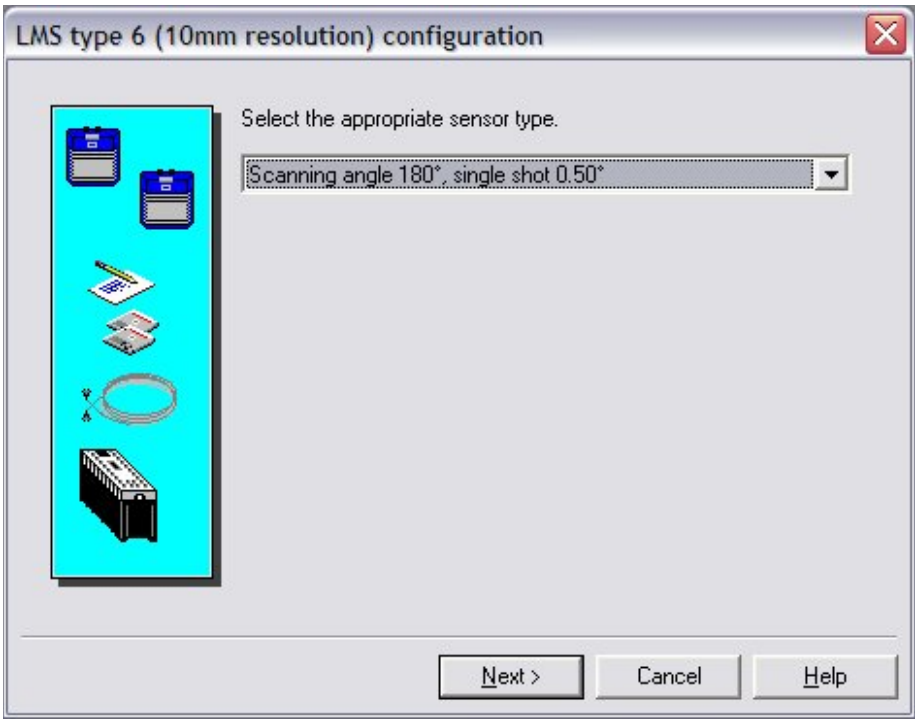


The software will prompt you to search for the correct buad rate and connection type. At this point, plug in the sensor. The sensor should display an orange and red light. After about 15 seconds, these lights should change to a green light. If this does not happen or if the sensor displays a different light pattern, refer to the manual for troubleshooting. After the green light appears, select "Next". The software should successfuly detect the LMS 200. If not, refer to the manual for troubleshooting.

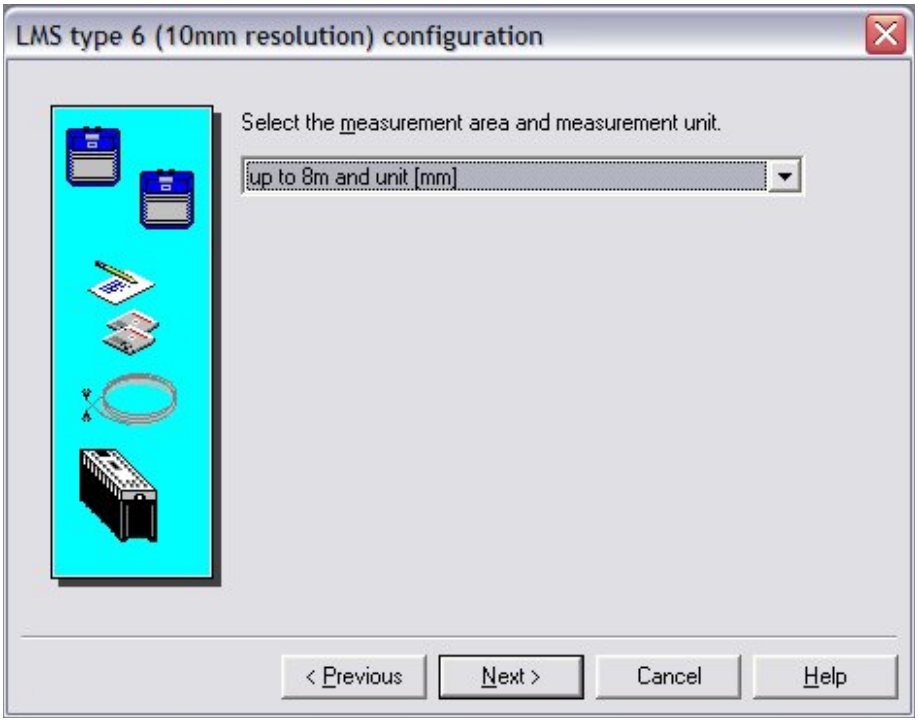
From the menu bar, select **File -> New**. From the window that appears, select "LMS type 6 (10mm resolution) configuration" and hit "OK".



In the next window you can select how the sensor is configured. From here you can change the angular range and resolution of a scan. Select "Scanning angle 180°, single shot 0.50°" and hit "Next".

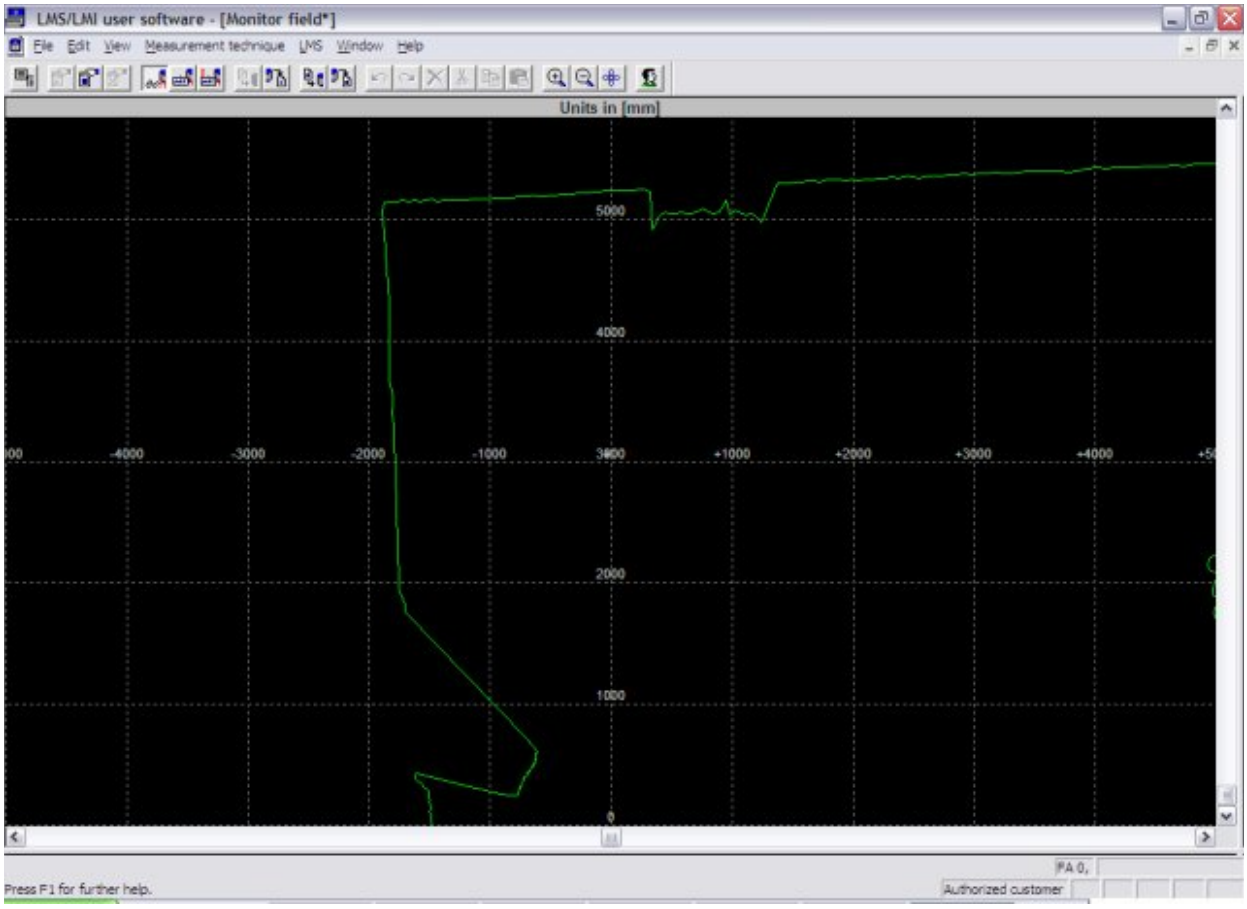


Next set the distance range and resolution. Continue with the default "up to 8m and unit [mm]". Select "Next".



The software will indicate that you have selected a new sensor configuration. Select "OK". A page will appear detailing the configuration settings. In the top left corner, select "Confirm". You will now be prompted to log in. For user, select "Authorized customer". The default password is

"SICK_LMS" (make sure all caps). The configuration should be successfuly sent. Now you can begin scanning with the sensor. At the top, hit the zoom out magnifying glass until you can see the 5000 mark on the grid. Hit F3 to begin scanning. You should see something like this:



The green line represents the scan line. Congratulations! You just performed your first scan with the SICK LMS 200.

PROGRAMMING

Before we begin programming, we need a crash course in communicating with the SICK. Detailed information about communicating with the SICK can be found in the "Quick Manual for LMS Communication Setup" which is linked here:

LMS_Quick_Manual_V1_1.pdf

The SICK receives commands as streams of bytes through the serial port. When transmitting data, it sends back streams of bytes corresponding to distance measurements at a given angle. A typical stream of data from the SICK might look like this:

2 128 214 2 176 105 65 79 23 76 23 77 23 .....

NOTE: These are the decimal values! Most programs (such as hyperterminal) display RS232 data as ascii characters. So the same string of data in hyperterminal might look like this:

%2?Ö%2°iAO.L.M%2 .....

It is therefore important to remeber that you must always be sending and receiving BINARY data and not ASCII characters.

To grab data from the SICK, you must first send a start string to tell the sensor to start sending data. This string is:

Hexadecimal Form: 02 00 02 00 20 24 34 08
Decimal Form: 2 0 2 0 32 36 52 8

After the start string has been successfully sent, the sensor will begin streaming RS232 data. The data comes in packets that take the following form:

STX	ADR	LenL Low byte	LenH High byte	CMD	Data LenL	DataL enH	Data 0° Low byte	Data 0° High byte	Data 1° Low byte	Data 1° High byte	..... (more data)	Status	CRC Low byte	CRC High byte

Designation	Data size (number of bits)	Remarks
STX	8	Start byte (STX = 02 hex)
ADR	8	Address of the subscriber (in this case the PC) addressed. Typically, the value is (81 hex).
Len	16	Length of the total LMS output data string. Number of following output data string bytes excluding the checksum (CRC = 2 bytes)
CMD	8	Command byte, in this case (B0 hex), which is the command for continuous data output.
DataLen	16	Number of measurement data bytes (depending on measurement mode, refer to section C.8.)
Data ...	n x 16	Measurement data values (2 bytes each) according to the measurement mode settings. Please refer to the note below)*
Status	8	Status byte Indication of system error, pollution etc. Refer to telegram listing for exact information.
CRC	16	CRC Checksum

For example, a packet of data might look like this:

```
2 128 214 2 176 105 65 79 23 76 23 77 23 ......
```

Notice that there are 7 bytes sent before data (highlighted in yellow) is sent. These 7 bytes are the packet header. It contains information about the data being sent. If you know what this header is supposed to be, you can use it to identify the beginning of the data. In our case, this header will be:

Hexadecimal Form:	02 80 D6 02 B0 69 41
Decimal Form:	2 128 214 2 176 105 65

The incoming data from a scan is sent sequentially as the sensor scans through 180°. In our case, since the sensor is set to scan 180° with resolution of 0.5°, the first data point sent will correspond to 0°, the next will correspond to 0.5°, the next to 1°, and so on. This means we should have a total of 361 data points. Each distance measurement is sent in the form of two bytes. The least signifficant byte is sent first, followed by the most signifficant byte. Since we're operating in mm mode, the unit for the measurements will be milimeters. For instance, if the measurement were 5m, the data would look like this (in decimal form):

```
136 19
```

To recover the measurement, perform the following operation:

```
136 + 19*256 = 5000 mm
```

Since each measurement is represented by 2 bytes, and there a total of 361 data points, the incoming data stream should have a total of 722 bytes. This is important for choosing the buffer size and for parsing the data.

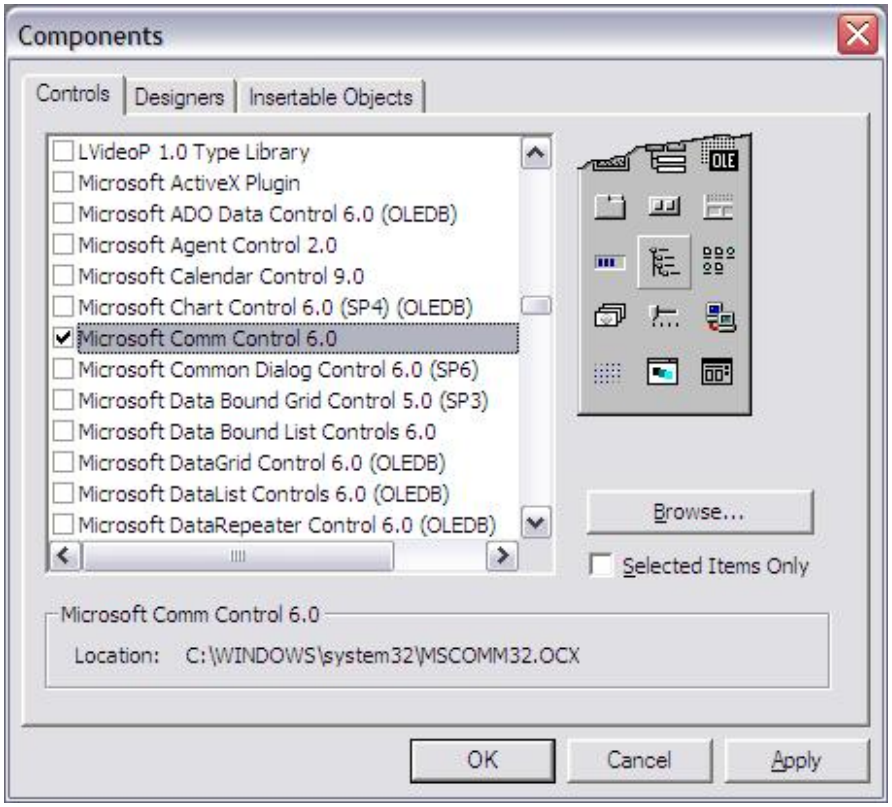
Finally, to stop the sensor from sending data, a stop string must be sent. This string is:

Hexadecimal Form:	02 00 02 00 20 25 35 08
Decimal Form:	2 0 2 0 32 37 53 8

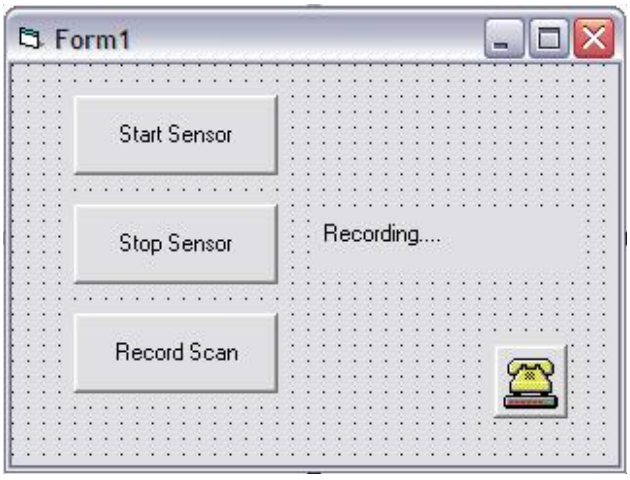
Now we're ready to program. This tutorial will detail the major parts of the code. The full code can be found here:

sickVBCode.zip

We'll create a program that will record a scanned line every time you press a button. Start up VB6 and when prompted, begin a new "Standard EXE". First we'll need a comm control for RS232 communications. From the menu bar, go to **Project -> Components**. This should pop up a window titled components with a bunch of check boxes. Scroll down and find "Microsoft Comm Control 6.0". Check the box and click "OK".



You should now see a small telephone icon on the left side of the screen. Click this to select it, then click and drag a box on the form. This will and a comm control called "MsComm1" to your project. Also add three buttons to your form: one to start the sensor, one to stop the sensor, and one to record a scan. Change their names to cmdStartSensor, cmdStopSensor and cmdRecordScan, respectively. Finally, add a label named "recordLabel" and change its caption to "Recording...". Your form should now look like this:



Now let's look at the code. The first part of code executed is the form load event. This is used to initialize variables and set up the comm port. In particular, lets look at the initialization of the comm port:

```
'Initialize RS232 communication
With MSComm1
   .CommPort = Val(port)
   .InputMode = comInputModeBinary
   .InputLen = 2000
   .Handshaking = 0 - comNone
   .RThreshold = 2000
   .RTSEnable = False
   .Settings = baud + ",N,8,1"
   .SThreshold = 1
   .PortOpen = True
   ' Leave all other settings as default values.
End With
```

There are two important things to note in this snippet of code. First the input mode is set to "comInputModeBinary". This means that the data coming in will be in decimal form as opposed to strings of ascii text. Second, the input length and receive threshold have been set to 2000. The receive threshold determines how many characters are received before the "comEvReceive" event is triggered. The input length determines how many characters are grabbed from the buffer when the "mscomm.input" method is invoked. In effect, when the buffer fills up with 2000 bytes, we empty the buffer into a local variable.

Recall that we are looking for 722 bytes. To ensure that we capture a complete data string, it is necessary that the buffer be at least twice the size of the incoming data string. Therefore, the buffer has been set to 2000.

After the form load, the majority of the code in located in "Private Sub MSComm1_OnComm()". This code is executed whenever a comm event occurs (in our case, whenever the buffer is full). Skipping past the code to grab the data from the buffer, lets look at the first loop:

```
'This loop looks for the header in the data
Do Until ((c > 7) Or (i > (UBound(buffData) - 1)))
   c = 1
   i = i + 1
   While (headerString(c) = buffData(i))
     c = c + 1
     i = i + 1
   Wend
Loop
```

Notice that there are two nested loops here. The Do loop is used to step through the buffer. If it reaches the end of the buffer, it exits. The while loop is used to compare the expected header to the buffer. If the entire header has been found, "c" will equal 8 and we exit the Do loop. Now lets look at where data is written to the file:

```
'Write the angle and distance measurement to the file
For i = i To i + 721
   'calculate the distance measurement from the lower and upper byte
   Print #1, (deg & vbTab & (CLng(buffData(i)) + CLng(buffData(i + 1)) * 256))
   deg = deg + 0.5
   i = i + 1
Next i
```

Skipping past code that labels the columns in the text file, lets look at the loop where the data is written. Notice that this loop iterates from the current position in the buffer to a position 721 bytes later. This ensures that we grab all of the data. Next, look at the print command.

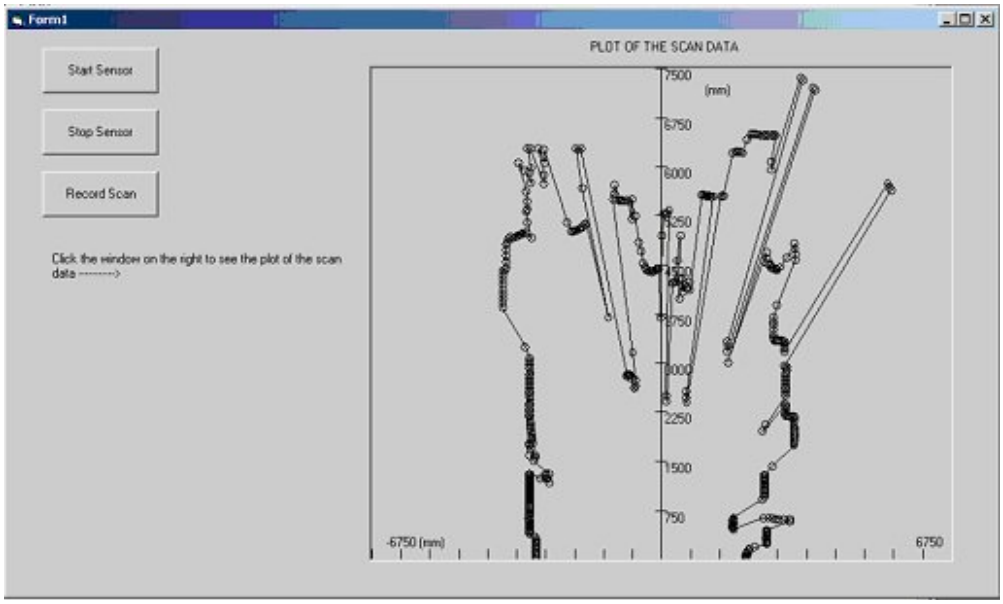
Specifically, look at the following code:

```
CLng(buffData(i)) + CLng(buffData(i + 1)) * 256
```

This is where the distance is actually calculated from the two bytes that comprise the distance data. The lower byte is added to the upper byte multiplied by 256. Finally, the counter for the current angle is incremented and the counter for the buffer is incremented. Note that incrementing the buffer counter inside the loop in effect makes it count up by 2. This is so that we grab 2 bytes at a time.

This code is the basics you need to grab data from the sensor. If you'd like to plot the data as well, that can be done with the following code:

sickPlottedVBCode.zip



Essentially, to plot the data the following snippet of code is added:

```
'Enter in location of FILE temp.txt (Location where data was written from a scan)
Open "C:\temp1.txt" For Input As #2
'Number of data points in one scan
n = 361
'Initialize starting values as zero
x(1) = 0
y(1) = 0
For k = 2 To n
   'Input the angle and the radius
   Input #2, angle(k), r(k)
   'Conversion to Cartesian Coordinates
   x(k) = r(k) * Cos(angle(k) * (3.14159 / 180))
   y(k) = r(k) * Sin(angle(k) * (3.14159 / 180))
   'Plot the data points as circles
   scanplot.Circle (x(k), y(k)), 100
   'Draw lines between data points
   scanplot.Line (x(k - 1), y(k - 1))-(x(k), y(k))
```

```
Next k
Close #2
```

Skipping past code that initializes the plot window, lets look at how the data from the sick laser is converted to Cartesian coordinates for plotting. The scan data is obtained from the file that it was written to as described earlier in this tutorial. The data is imported in the form of range (distance to a measurement in mm) and angle (in degrees). Plotting the data on a standard x-y position plot, requires mapping the range and angle data in Cartesian coordinates. This is accomplished using the formula x = range * cosine (angle in radians) and y = range * sine (angle in radians). Each data point is then plotted as a circle and connected with the previous data point by lines to generate a complete plot of the scan.

FINAL WORDS

After reviewing this tutorial you should be able to set up a SICK LMS 200 and write a simple program to store data from the sensor. The principles introduced here can be applied to different programs and programming languages. Now that you know how to get distance reading from the sensor, much more complicated codes can be created from field monitoring to collision avoidance for a robot. If you have questions about this tutorial you can email me at **Keithicus@drexel.edu**.