Bradley J. Wimpey
bjwimpey@yahoo.com
January 12, 2004

Bluetooth

The Bluetooth Protocol stack in Windows CE (winCE) 3.0 is from a company called Widcomm. A cost-effective (i.e. free) way to go about bluetooth development in winCE 3.0 is to download BTAccess from High Point Software (http://www.high-point.com/). The demo version of this software development kit is free, and one can then easily access the bluetooth stack via BTAccess libraries. One will need a third party solution for bluetooth development since software development capabilities did not become integrated with the OS until winCE 4.0 (.NET). We developed our program within Microsoft Visual C++ 3.0 using Microsoft Foundation Classes (MFC). The iPAQs we used are version 3970.

The communication on each iPAQ is set up like so:
The very first thing each iPAQ must do is establish a connection to the bluetooth stack. Once this is done, we can use other functions that access the bluetooth stack. The BTAccess software provides the CBtStack class, used to do just that.

```
CBtStack  *m_pStack;
m_pStack = new CBtStack();
eBTRC eRC = m_pStack->Connect(this);
```

Next, to add a little more autonomy to the bots, we adjusted the security settings so that authorization was not required to make a bluetooth connection between the two iPAQs.
We set the bAuthorizationRequired variable to false within the SVC_SECURITY_OPTIONS struct. This allowed us to not have to authorize each and every bluetooth connection between the iPAQs.

Next, each iPAQ searches for bluetooth capable devices in the local area (to find and connect to each other). Since we only have the two bluetooth iPAQs in the area, they only see each other. When iPAQ1 discovers iPAQ2, a bluetooth connection is made, but iPAQ2 will also have to search, discover iPAQ1, and make the same connection.

Now that the iPAQs are connected via bluetooth, we can initialize a serial connection. The serial connection is where we do our transmitting of data. When the two devices are both connected over a virtual serial connection, we must establish read and write ports for them to communicate.

Both iPAQs need to read and write, so we set up the virtual com ports COM7: and COM8: on each.

```
HANDLE hCommPort = CreateFile(_T("COM8:"),
                    GENERIC_READ | GENERIC_WRITE,
                    0,
                    NULL,
                    OPEN_EXISTING,
                    0,
```

NULL);
hCommPort would therefore be used on one iPAQ to write outgoing data.
HANDLE readPort was the same, except it used COM7:.
To write, we called function WriteFile on handle hCommPort. To read, we set up a read thread,

```
AfxBeginThread(ReadThread, this);
```

and the function

```
ReadThread(LPVOID lpParam)
```

where the reading occurred. We read in from COM7: by calling the function ReadFile on handle readPort. CreateFile, WriteFile, and ReadFile are available in the API for Windows CE OS versions 1.0 and later. The AfxBeginThread function is available from the Microsoft Foundation Classes library for Windows CE 2.0 and later.

Now that our connections are set up on both iPAQs, we can transmit our data.

The bluetooth communication was used to share orientation of the first robot with the second. This orientation was in the form of SHORT values representing the amount of inches to a given wall. This is how we prepared the values and sent them across bluetooth:
iPAQ1
We converted each of the two SHORT values, say x and y, to integers, and then we stored them in an integer array. The values were sent one at a time. So with our first value x, we converted the integer to tchar by calling function _itot. We then prepared the text to be sent over serial with function wcstombs. The data was then transmitted by calling WriteFile.
iPAQ2
When the read thread received data, it stored the data in a read buffer, szBuff. Then, all we had to do was:
int coordInt2= atoi(szBuff);
iPAQ2 then sent an acknowledgement that it received the data. The ACK was simply the integer 50000, since we would never get a value of 50000 as an actual, correct, orientation value.
iPAQ1
When the ACK value of 50000 was received in its read thread, iPAQ1 then knew it was OK to send the next orientation value.

This process is repeated a second time to completely send the two values to the second iPAQ. When both coordinate values are received, the second iPAQ initiates action on them by seeking to make its own coordinates match those it received.