UW Campus Navigator: WiFi Navigation

Eric Work
Electrical Engineering Department
University of Washington

Introduction

When 802.11 wireless networking was first commercialized, the high prices for wireless networking equipment deterred most consumers. When IEEE 802.11a was first standardized consumers weren't ready and there was little commercial adoption. Shortly after IEEE 802.11b was standardized. This time commercial adoption was more prevalent, and manufactures were striving to build smaller and cheaper wireless network cards. Manufactures continued to follow this trend as new standards were released. Now with IEEE 802.11g available you can get 25 times the performance for about one-twentieth of the price[1]. The market for 802.11 radios is currently about 50 million units per year[1]. It is this trend that has caused a boom in the adoption of 802.11 wireless networking.

This rapid deployment of new wireless access points each year is a crucial part to the success of WiFi Navigation. You may be well aware that wireless networking allows users to roam freely with their mobile devices and effortlessly connect to networks from a distance. What you may not know is that wireless access points can be used as fixed beacons in a wireless positioning and navigational system.

The inspiration for WiFi Navigation has come from number of sources, first and foremost the Intel Placelab and their work on location-aware computing[2]. Intel has already done a significant amount of work on 802.11 wireless positioning. My primary interest in WiFi Navigation originated from a desire to incorporate Intel's technology into the UW Campus Navigator[3]. The UW Campus Navigator project was first started in July 2003 as a research exchange program between the University of Washington and the Chinese University of Hong Kong. The outcome of the project was a device a student could carry, which allowed them to search for restaurants on campus and navigate to their destination using a built-in map.

As a team member of the UW Campus Navigator project, I was in charge of the navigational aspects. Originally plans were drafted to incorporate WiFi Navigation into the final project. After having a number of difficulties with incompatibilities between Intel's work and our platform choice, we had to rely upon GPS. After the conclusion of the research exchange I continued working on the project and decided to instead use Intel's navigational theory and develop all the necessary code from scratch. This has been the inspiration for the development of WiFi Navigation on the Pocket PC.

Implementation Overview

The concept of WiFi Navigation is very simple despite the technicalities of the underlying hardware. The most important part to the system is a database located on the mobile device with a predetermined list of wireless access points in a defined area with corresponding longitude and latitude coordinates. When a mobile user begins navigating, the device will first collect a list of wireless access points within range of the radio.

This access point list will then be compared against the database to extract a set of coordinates for the access points which are in range. Using this coordinate set and an algorithm, the mobile device can estimate where it believes the user is located. Using standard mapping technology this position can be placed on the map so the user can then navigate their way to their destination. It should quickly become apparent that a high density of wireless access points is a desirable characteristic.

Accessing the Wireless Hardware, Obtaining Access Point Information

The most crucial part for WiFi Navigation is the ability to access the wireless hardware and obtain access point information. What is needed is a way to retrieve a list of access points within range of the wireless device. From each visible access points the minimum requirements are the MAC address and the signal strength if WiFi Navigation is ever to work. Once hardware access is established, obtaining all the desired information becomes easy. If accessing the wireless hardware remained unresolved, nothing more could have been done and the project would have come to a standstill.

Accessing the network driver in Pocket PC 2003 turned out to be a more time consuming task than expected. When I began, a tremendous amount of searching was done on the Internet looking for ways to get the required access to the underlying network driver. While searching, I quickly found a way to access an assortment of hardware information in Windows XP using Windows Management Instrumentation (WMI). In a number of example programs I was able to check the connection status of various installed network devices on my desktop computer. When looking into WMI for the Pocket PC I found that it had been developed, but was planned for release in the upcoming Window Mobile .NET. Somewhat set back by the disappointment I decided to continuing searching for other techniques.

While working on the UW Campus Navigator the previous quarter we experimented with a program called Pocket Warrior[4] which performed operations similar to those required for WiFi Navigation. The software was developed for Pocket PC 2002 but the source code was an excellent starting place for examples of accessing to the wireless hardware. Pocket Warrior I discovered was developed on top of the Network Driver Interface Specification (NDIS), which provides a common API for accessing low level network drivers, and is available in with almost every version of windows. The NDIS examples that were available would not compile for Pocket PC 2003. Now more familiar with the API, I continued searching for examples and references to NDIS on Pocket PC 2003.

After more searching I discovered NDIS User Mode I/O (NDISUIO) which is similar to NDIS but has been evolved to make hardware access easier and more unified. NDISUIO is a special driver which eliminates the need for connecting to each low level network driver individually and in addition also supports 802.11 network interfaces. By simply connecting to the NDISUIO driver and telling the driver which device to access, the rest of the connecting is done automatically. The MSDN libraries were a great source for learning the Windows API associated with the NDISUIO driver. After learning the API I developed a simple test program in Embedded Visual C++ (eVC++). Once I saw progress, I finished developing all the routines that might have been useful for WiFi Navigation. When all the wireless functionality was complete, my test program successfully displaying access point information on the Pocket PC.

With the hardware access now working, this functionality had to be incorporated into .NET to make integration with the UW Campus Navigator possible. To facilitate the integration all the functions had to first be put into a Dynamic Link Library (DLL). Once the DLL was created a .NET class had to be developed to encapsulate all of the function calls and data manipulation. There were difficulties ensuring that the functions were imported correctly into .NET, but nothing major.

The cause of some stalling in development at this stage was marshaling data between the eVC++ and .NET C#. Marshaling is the process of importing unmanaged data, like C++, into a managed environment, like C# where the language handles all the memory management. The full .NET Framework has support for marshalling structures where the field length may be variable. In the .NET Compact Framework this does not exist, so an alternative is duplicating the field and naming the extra fields an arbitrary name until you have the desired field length in term of base types. The more appropriate alternative for my implementation was dealing directly with the unmanaged pointers, and extracting an array of bytes then performing type conversions on the array. Once marshaling was correctly done the class was compiled as a class library which could be included into any .NET application.

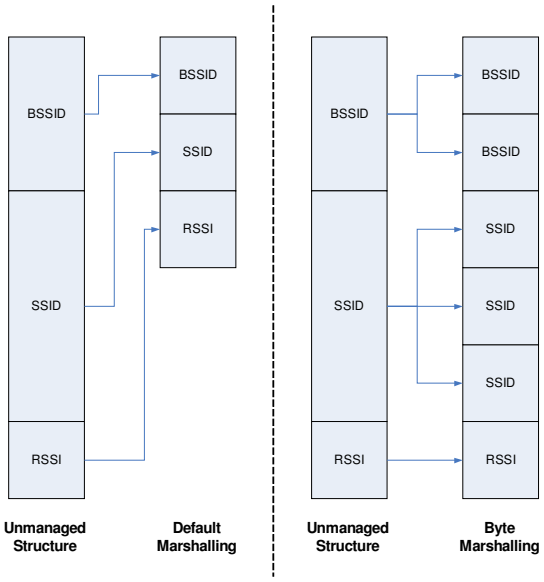

Figure 1: Column 1 depicts how the default marshalling leads to incorrect field lengths. Column 2 depicts using byte-by-byte marshalling to correct the field lengths.

Building the Database, GPS to WiFi Translation

The next major component to WiFi Navigation is an access point database. At minimum the WiFi database must store the MAC address, latitude, and longitude for each access point, and be stored locally on the Pocket PC. The UW Campus Navigator originally utilized a local SQL CE database for storing location names and their positions. The SQL CE database format was the natural choice, since pre-built libraries were already included with .NET, and working code was available from the previous project. In addition an entire SQL CE database is contained within a single file and can easily be replaced or moved for easy upgrades when new access points became available.

Once the decision was made to use SQL CE, functions were developed to load the database with access point information and run queries to retrieve location information. To quickly load

information into the database I developed a utility to parse comma separated value (CSV) files. Each CSV file contained a series of lines with a MAC address, latitude, and longitude separate by commas. For each line of the CSV file, a new database entry was created and the values from the CSV file were mapped into their corresponding fields. By using CSV files, existing text manipulation utilities could be utilized. Basic text utilities could be used to manually insert new entries, remove duplicate entries, or merge CSV files. By using this implementation approach the access point database was flexible and portable.

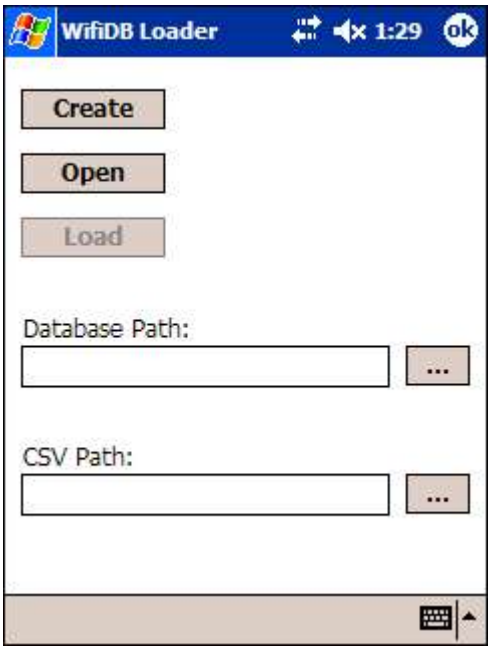

Figure 2: A screenshot of the utility that creates a new database and loads in data from a CSV file.

In order to build a database, a collection of CSV files first needed to be created, either through manual entry or through automation. A main goal of WiFi Navigation is taking advantage of the growing number of access points throughout the community. For this to be possible an automated approach had to be used. To facilitate the automated collection process I developed a utility to scan for wireless access points with the GPS enabled and assign latitude and longitude values to each unique MAC address found.

The collection algorithm used in my implementation was very simple, but performance could be improved by using a more sophisticated technique. To begin, the wireless hardware performs a scan and returns a list of the access points within range. For each item in this list the MAC address is compared to other previously saved entries to locate any duplicates. If there are no matches, a new entry is created in the database for the new MAC address and is assigned the current GPS location. If there is a match, the previously saved signal strength is compared to the new signal strength. If the new signal strength is greater than or equal to the saved signal strength, the GPS location for the saved entry is updated with the current GPS location. If instead the new signal strength is less then the old signal strength, then the list item is ignored.

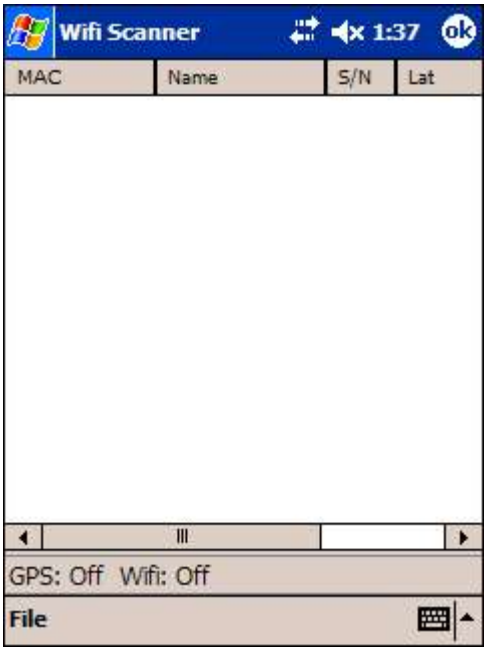

Figure 3: A screenshot of the utility that scans for access points and assigns each a GPS coordinate.

The objective of this simple algorithm is to assign each MAC address with an approximate real GPS location. The location where the highest signal strength was observed is estimated to be the closest location to the true location. To help organize the collection of CSV files, each time the scanning process is terminated the file includes a time stamp. Time stamping allowed me to separate a particular region into one CSV file that could later be identified. Once the collection of CSV files was complete the files were loaded into a new database that could be used by my WiFi Navigation system.

Hardware and Database Integration, WiFi to GPS Translation

The elementary concept of WiFi Navigation implies a way of extracting GPS information from a list of access points. The access point database was created by mapping WiFi access points to GPS locations. For WiFi Navigation to work the process must be reversed. The final step to completing the implementation is linking the access point information retrieved from the hardware, to the GPS information inside the database. By moving the functionality of the access point database into my .NET class library I was able to join these pieces together. By using a unified library, the WiFi Navigation functionality could be effortlessly included into any new project.

With the unified library complete, a positioning algorithm was then developed to actually perform the reverse translation to GPS coordinates. The algorithm used in my implementation was simplistic and considered only one factor, the signal strength, to compute a weighed average. A more complex algorithm, which considers other factors, could improve the performance. The algorithm begins by first scanning for nearby access points. For each item found during the scan, the signal strength is weighed between zero and five. Next the database is queried, using the MAC address, to retrieve the latitude and longitude. The resulting latitude and longitude are multiplied by the signal strength factor and added to a running total. The number of entries to average is then incremented by the signal strength factor number. After every item had been processed the running total is divided into the number of entries and an estimated GPS location is obtained.

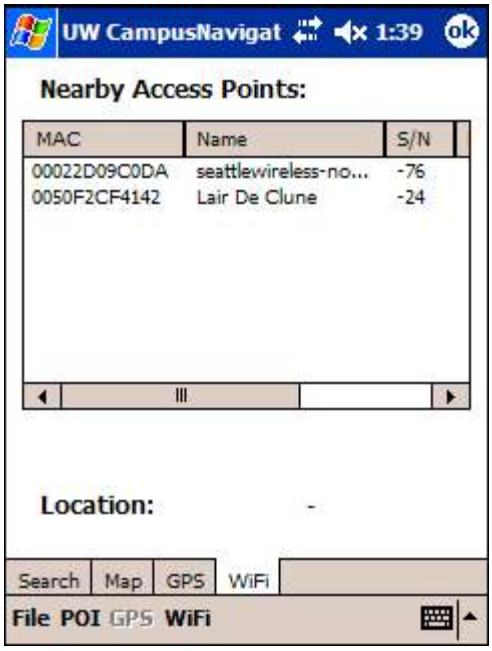

Figure 4: A screenshot of the UW Campus Navigator with WiFi Navigation implemented.

After the algorithm was implemented and all the remaining components fully debugged the result was a complete WiFi Navigation system. The next step was integrating the system into the UW Campus Navigator. Added to the UW Campus Navigator was a WifiGps class, which contained a subset of the functionality available with GPS, but was based on the WiFi Navigation system. By linking the WifiGps class to the map I

had a useable WiFi Navigation system fully implemented.

WiFiGPS Compared to GPS

Only after looking at alternatives such as the U.S. government's global position system (GPS) can the benefits of WiFi Navigation be seen. Since GPS has been standardized and receivers are widely available to the public, GPS is popular among consumers for positioning and navigation. Instead of using a fixed position as a reference, GPS works by using twenty-four satellites moving in low-orbit, which broadcast positioning information to a GPS receiver. Once the receiver has a clean signal from at least three satellites it uses a complex algorithm to accurately position the user to within a couple meters.

The first version of the UW Campus Navigator relied upon GPS because of its reliability and ease of use. Though prices for GPS receivers are decreasing they are still costly. Cost was an important factor to consider when developing WiFi Navigation, due to the lower cost of wireless radios over GPS receivers. With the widespread adoption of wireless networking technology soon access points will be on every corner and every mobile device will have an 802.11 radio (or so we hope).

After a little testing it didn't take long to realize that GPS doesn't work indoors. The signal transmitted from the GPS satellites are weak and are easily distorted without a clear view of the sky. Access points do not suffer from this problem hence wireless networking works through walls. This was another motivation for WiFi Navigation. Although WiFi Navigation is not as accurate as GPS, for a number or reasons, it is beneficial to have some form of navigation indoors. The signal transmitted from the GPS satellites also can take time to synchronize before acquisition can occur. When turnrf on, a GPS receiver can take between 30 seconds to 30 minutes to acquire an accurate GPS position. As soon as a wireless radio is turned on, scanning can begin. The quick initialize time of WiFi Navigation avoids long wait periods before navigation can begin.

Error Analysis

One major downfall of WiFi Navigation is the error factors that account for a decrease in the accuracy of positioning. After performing field testing the most significant contribution to this error is the method used to assign access points in the collection algorithm. By simply assigning a location based on the highest observed signal strength, access points can only be positioned at locations where the scanner was once located. When the application attempts to later reposition the user with the positioning algorithm, an incorrectly assigned position will be used in the calculations. If the application detects that the signal strength observed is similar to the signal strength used when assigning the access point a location, the device will assume the user is at the assigned location. The user could actually be on the opposite side of the access point locating the user at an incorrect position.

The remainder of the positioning errors are accounted for by using only one factor in position algorithm. The weighed average does not account for the variance in signal attenuation through walls or object made of various materials. Since the wireless hardware provides only the signal strength other factors have to be derived during the collection process.

Improvements

With additional research there are likely many ways to improve WiFi Navigation. I will describe one such way suggested to me by Professor Mari Ostendorf. These suggestions help overcome the drawbacks of positioning errors, by using more complex but more reliable algorithms for both the collection process and the positioning process.

To improve the collection process the algorithm needs to actually guess the true location of the access point, not simply assign the access point a location. To make a guess, the signal strength and GPS location at three different positions around an access point must be recorded. After collecting this information the signal strength can be used to approximate the distance from the access point to the device. This will be the radius of the first guess circle. After doing the same for the other two positions there will be three overlapping guess circles. The area of the overlapping region between the three guess circles is the probability that our guessed location is correct. The center of the overlapping region is the guess location. If more than three positions are available, the guess location will be closer to the true location there by increasing the probability of the guess. After each guess the MAC address, latitude, and longitude are saved along with the probability (reliability) of the guessed location.

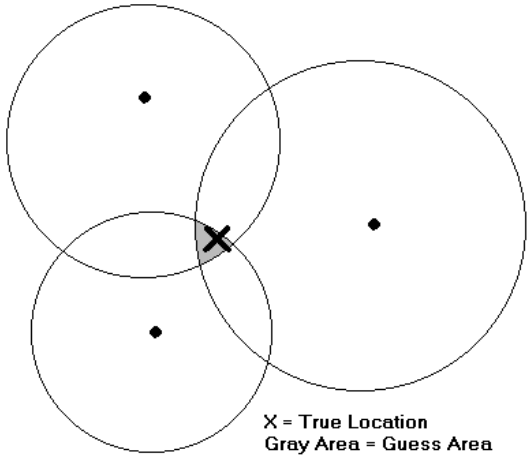

Figure 5: Depiction of guessing an access point's true location using three sample locations

Since the location probability was saved during the collection process, this new information can be incorporated into the positioning algorithm by now having two factors. The first factor remains the same while the second factor is now a rating based upon the reliability of the access points location. Estimating the user's position is now simply a weighted average of two factors. These two algorithms when used in conjunction will most certainty increase the accuracy of WiFi Navigation.

Conclusion

During this research project I have had the opportunity to learn about the Windows CE API, in particular NDISUIO and wireless networking. I have also had the opportunity to explore methods for triangulation and positioning. There is still work to be done and things to learn. WiFi Navigation is a cutting-edge research topic with room for exploration and improvement. The implementation that has been described in this paper is very basic and without a doubt further research can improve WiFi Navigation. I hope

that one-day WiFi Navigation will become common among mobile devices.

I would like to thank Mari Ostendorf for sponsoring the research and the Electrical Engineering department at the University of Washington for purchasing the hardware.

References:

[1] Gurley, J. William. “Why 802.11 is underhyped.” http://news.com.com/2010-7351-5153319.html (4 Feb 4, 2004)

[2] Intel Research Seattle. “Place Lab.” http://www.placelab.org/

[3] University of Washington. “UW Campus Navigator”. http://uwcampusnav.sourceforge.net/

[4] Dataworm. “Pocket Warrior.” http://pocketwarrior.sourceforge.net