U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
TECHNICAL REPORT

Midshipmen Blue Force Tracking

Evans, Paul K. Stahl, David J.

USNA-CS-TR-2005-08

December 13, 2005

USNA Computer Science Dept. ◦ 572M Holloway Rd Stop 9F ◦ Annapolis, MD 21403

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
13 DEC 2005		**00-12-2005 to 00-12-2005**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Midshipmen Blue Force Tracking	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **U.S. Naval Academy,Computer Science Department,572M Holloway Rd Stop 9F,Annapolis,MD,21403**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**20**	

Computer Science Department
SI495 Research Project Report
Fall 2005

Midshipman Blue-Force Tracking

by

Midshipman Paul K. Evans, 061896

United States Naval Academy
Annapolis, MD

Certification of Faculty Mentor's Approval

Assistant Professor David J. Stahl, Jr.
Department of Computer Science

Department Chair Endorsement

Professor Kay Schulze
Chair, Department of Computer Science

ABSTRACT

This project explores the feasibility of networking Windows CE based handheld devices using inexpensive off-the-shelf hardware and software systems to provide Midshipmen with a tactical training system simulating the FBCB2 system - Force XXI Battle Command, Brigade and Below ("Blue Force Tracking"). In conjunction with the YP Tactical Data Simulator, "Midshipman Blue Force Tracking" is intended to be used as a pedagogical tool for educating midshipmen in the concepts of Network Centric Warfare and operations.

TABLE OF CONTENTS

TABLE OF FIGURES

1. BACKGROUND

The US military continues its transformation from the large, Cold War model into the smaller, more mobile, highly *network centric* Information Age force. A concept first articulated in the late 90's, this paradigm shift in war fighting has been vindicated by the military successes in Afghanistan and Iraq. Network Centric Warfare (NCW) is more than just the hardware, however. The 2001 Quadrennial Defense Review notes that "fundamental changes in ... organizational culture and behavior are usually required to bring ... [transformation] about" [1]. As Adm. Cebrowski observes, "The implementation of NCW is first of all about human behavior as opposed to information technology" [2]. At USNA, the officers assigned to the Department of Professional Development - those who are perhaps most responsible for developing a culture receptive to force transformation and educating midshipmen in its principles - seem reluctant to do so. The reluctance here is consistent with the results of a recent survey concerning officer attitudes towards military transformation, which concluded: "Junior officers do not see transformation as something important to them" [3].

We claim that NCW should be embraced in theory and practice across the professional development curriculum, and not isolated to the strictly theoretical treatment it receives in the NS410 Network Centric Warfare course. To that end, this project focuses on technology: developing a tool to facilitate hands-on education in the tenants of NCW. One of those tenants is *shared situational awareness*: developing a common tactical picture that allows increased speed of command and coordination of effort across widely distributed forces. We propose a "Midshipman Blue Force Tracking" (BFT-M) system for midshipmen, similar to the Army FBCB2/BFT system, that can be used as a pedagogical tool for educating midshipmen in the concepts of Network Centric Warfare and operations. This foundation would greatly enhance each new officer's ability to use similar tools, as well as provide a platform for midshipmen to develop the tactical training taught in the classroom in a simulated war environment. Such training only occurs in a handful of optional summer training courses to include Leatherneck and Mini-BUDS which only reaches a small number of midshipmen.

2. APPROACH

Force XXI Battle Command, Brigade and Below/Blue Force Tracking (FBCB2/BFT) is the major digital command and control system for the Army at brigade level and below. The primary functions of FBCB2 are to send and receive position reports derived from the Global Positioning System (GPS) and to exchange command and control messages via secure digital satellite radio transmissions. FBCB2 has been widely praised, proving its worth in the rapid, coordinated advance to Baghdad by Coalition forces during Operation Iraqi Freedom (March 2003).

A previous midshipman's research projects developed a similar system designed for education and training use aboard USNA Yard Patrol Craft: the YP Tactical Data Simulator (YP-TDS) [4]. YP-TDS uses non-secure HF packet radio as the communications mechanism, allowing up to six YPs to operate jointly in a coordinated manner. YP-TDS represents a first attempt at incorporating hands-on exposure to NCW in the curriculum.

Figure 1: Blue Force Tracking on a PDA

Using lessons learned from the two previous YP-TDS midshipman research projects, a Midshipmen Blue Force Tracking system similar in functionality to FBCB2, which can be integrated with YP-TDS, will be developed. Fully envisioned, BFT-M would provide ground unit leaders in a training environment a full digital command interface, able to see locations and status of all friendly and known enemy forces. A simple vector/distance reading to all contacts in conjunction with a map would provide all the necessary information without allowing the trainees to become over dependant on this system for navigation purposes or over burden the PDA processor with extra graphics. Each contact should have specific data that is accessible by tapping on the contacts mark on the screen. A simple chat/mail function should also be built in so commanders can communicate with each other to discuss current strategy and tactics.

Figure 2: PDA-VHF handheld pair

The specific goal of this project is to implement the basic software and hardware needed to allow PDA to exchange position information read from GPS, via VHF pocket radio using PSK-31 encoding. PSK-31 is a low data-rate digital-analog-digital amateur radio communications protocol that

appears ideally suited for the non-secure transmissions we envision. By proving this approach is feasible, subsequent research projects can then can focus on integrating BFT-M and YP-TDS into a multi-platform land and sea system.

2.1. PSK31 - Phase Shift Keying (31 baud)

PSK31 was created by amateur radio user Peter Martinez to provide a replacement to RTTY in keyboard to keyboard communications [5]. It uses a variable length alphabet and a narrow bandwidth to efficiently transmit each character. We chose PSK31 due to its simplicity and abundance of tools for PC development. One of these tools is the PSKCore Dynamic Link Library written by Moe Wheatley, which formed the foundation of our development and testing.

The varicode alphabet used by PSK31 is designed so that the number of bits required to represent each character is indirectly proportional to how often each character is used. For example, the letter 'e' is encoded as '11' where as 'z' is '111010101'. It is also important to notice that all uppercase characters have a longer varicode length than the lowercase alphabet so messages should contain as few uppercase characters as possible. The varicode allows common characters to be sent over the air more quickly and use less CPU power to decode. To achieve the varicode alphabet, each character must be separated by two consecutive zeros ('00'). This means that no character can have any consecutive zeros in it encoding and each must start and end with a one. The varicode alphabet also means that the signal is self-synchronizing, dividing up characters as it reaches each '00'.

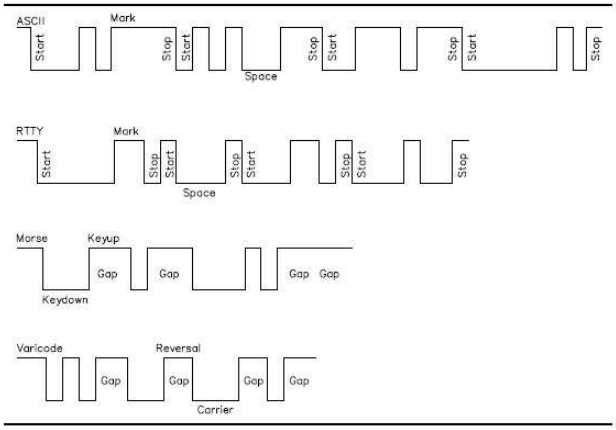

Figure 3: ASCII varicoding

PSK31 is based on phase shifting a signal. This makes decoding a signal a little more difficult. In order to decode an input signal, a Fourier Transform must be done to break up the signal into a useable function of sines and cosines. A Discrete Fourier

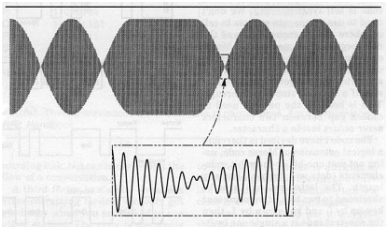
Figure 4: Signal phase shift

Transform usually can be computationally expensive $O(n^2)$, however the fast Fourier transform algorithm reduces this to O(nlg(n)) which makes it feasible for accomplishing the digital sound processing. Once we have the Fourier Series we can then analyze the signal for phase shift to decode the characters.

2.2. PSKCore

The PSKCore Library [6] implements PSK31 encoding/decoding as a Windows Dynamic Link Library, written in the C programming language. It allows full duplex communication simultaneously through multiple channels. The library offers numerous features suitable for flexible, general purpose HAM radio type communication, beyond those required for the simple, low data rate, non-secure, non-reliable data transfer we envision needed for BFT-M.

2.2.1. Signal Generation

PSK31 Signal Generation can occur in either Binary or Quad PSK. To keep the problem as simple as possible, we used the Binary PSK mode which is only capable of a 180 degree phase shift. The block diagram below maps the signal generation process as implemented by PSKCore.

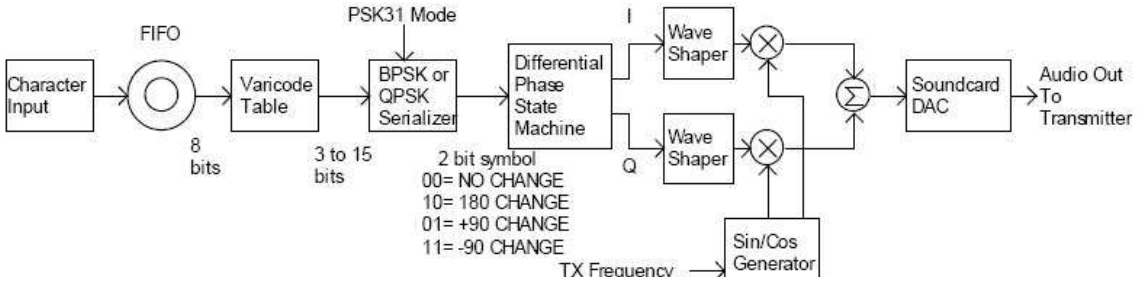

Figure 5: Signal generation block diagram

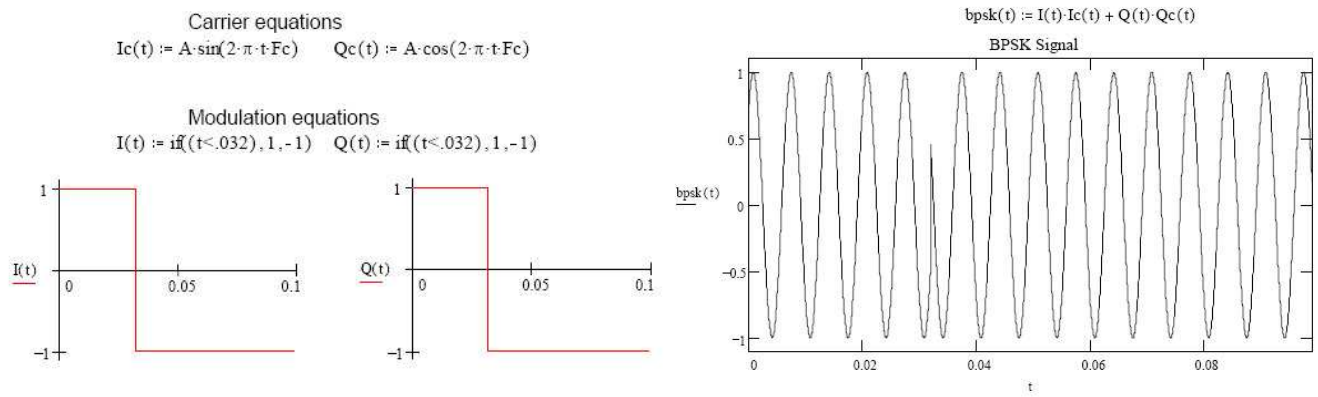

Figure 6: Modulatin step function

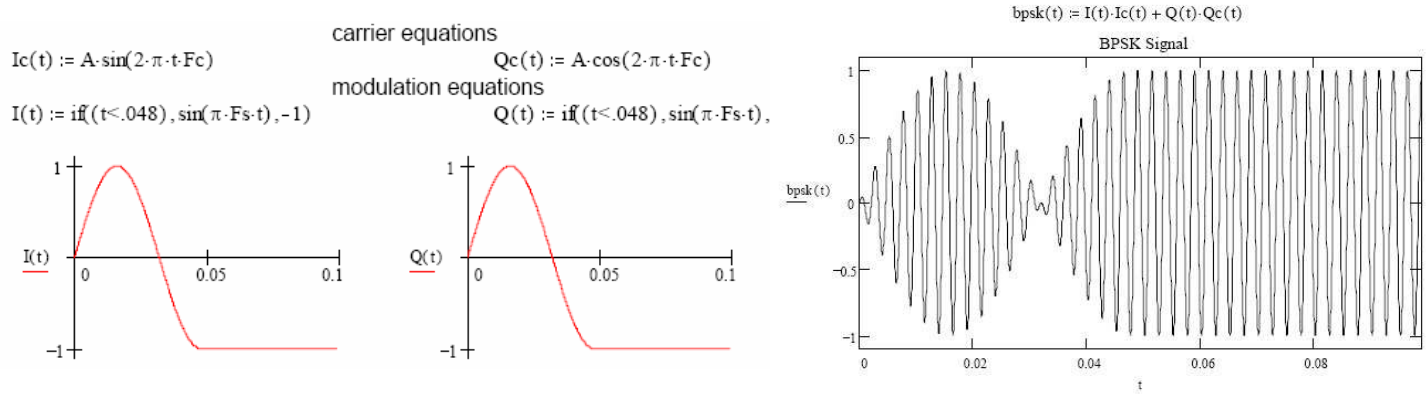

Figure 7: Cosine modulation and resulting signal

A character input from the user is sent to a queue where it waits for transmission. Once ready, the character is then translated into varicode and sent to the serializer. The serializer takes the varicode word and translates it to either '00' or '10' where '00' indicates no change in phase and '10' indicates a 180 degree phase change. Once the input has been serialized, it is sent to a differential phase state machine. The state machine then provides two outputs, I (in phase) and Q (quadrature phase) which are in simplest terms step functions to modulate the phase of the two carrier sinusoidal waves generated (Ic and Qc). The addition of the waveforms multiplied by their respective modulation is what gives the output BPSK signal a readable phase change. By using the modulation functions above we get a sharp change in phase. This sharp change in the phase of the wave causes the signal to become very wide. To counter this, a cosine transition is used in place of the pure step function. This allows a smooth transition and also prevents the use of post filtering.

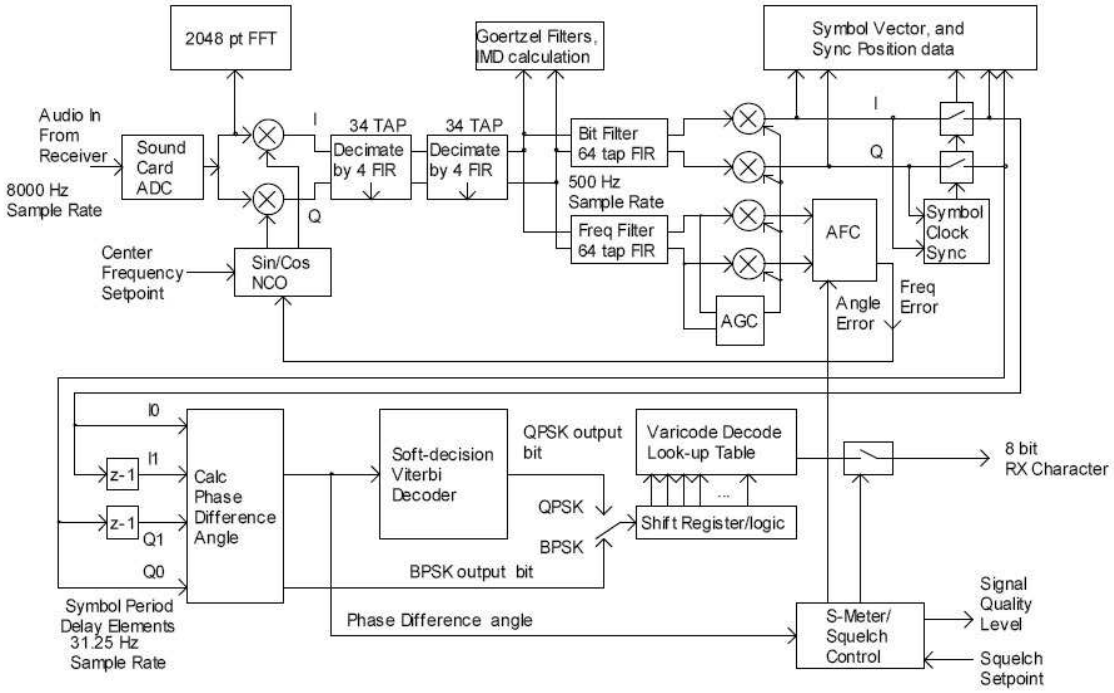

Figure 8: Signal detection block diagram

2.2.2. Signal Detection

Signal detection for PSK31 is much more involved than signal generation. To begin, the audio is read in from the sound card at an 8000Hz sampling rate and is then converted to floating point representation for the remainder of the processing. Next, the input is fed into a complex mixer where the real audio is converted into a baseband centered on a user set frequency. This allows us to then extract I and Q. Since the whole data set is not needed, both I and Q are then decimated by 16. The decimation is done through two finite impulse response filters (FIR filters) instead of one to help with CPU efficiency.

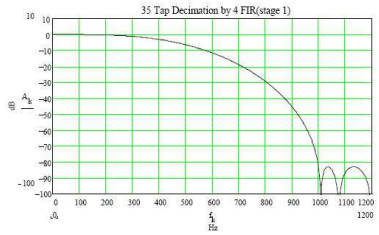

Figure 9: FIR decimation

The next series of steps deal with gain and frequency control as well as error control and correction. These tools are used to help clear up the signal when there is distortion. After the error checking is complete the signal is synchronized at a symbol clock rate. Once synchronized each symbol is then calculated for a phase change angle and then loaded into a shift register. This takes place until two zeros are read indicating the end of a character. The shift register is then sent to the varicode look-up table to be translated into the 8 bit character.

3 IMPLEMENTATION

This section details the steps of implementing the software used to test the capability of using PSK31 as a mobile networking tool. Our goal was to develop a simple application with the means to input text to be transmitted, display decoded received text, and provide diagnostic testing information. An MFC (Microsoft Foundation Class) dialog window application written for Windows Mobile 2003 was the approach we used [7].

3.1. Embedded C++

For creating the software to run on the Windows CE devices we used Microsoft eMbedded Visual C++, an embedded C++ integrated development environment. Embedded C++ was created in the mid-90s to allow software designers to write programs for their embedded systems. Essentially it is a subset of the C++ language designed to be easily run on smaller devices.

eMbedded C++ is available on Microsoft's site [8]. To install it, we had to first install the Microsoft Pocket PC 2003 SDK. This SDK allowed us to compile and run a simulated Pocket PC on your computer. It is also important to load Microsoft ActiveSync before installing eMbedded C++.

The SP4 update was the last download we needed before we could eMbedded C++. None of the earlier updates are necessary as SP4 is a cumulative update and handles all previous updates for you. Once these four programs were installed, we were ready to begin developing our testing software.

3.2. Porting PSKCore dll

PSKCore was originally written as a dynamically linked library for developing applications to be run on desktop-type Windows PCs. We eventually chose to modify the source code as needed and incorporate it directly in our own code for deployment on a PDA.

We originally had trouble linking the dynamic library to our program. This is what prompted us to simply include the source code for the library inline with our program. On our first attempts to compile we found that the PSKCore source code caused many errors. The first of which was a string conversion error. Since eMbedded C++ uses UNICODE it was not able to take char arrays as input to functions. This was remedied using the _T() macro built in eMbedded Visual C++.

Another significant problem was within PSKCore's wave.cpp file which was responsible for the .wav file input and output functions. Pocket PC does not include the MMIO API so it is unable to make the proper calls to create or read from a .wav file. Since we were only using the program for live conversation and did not require this functionality, we simply commented out the implementation of these problem functions to allow the program to function as we would like.

3.3. Testing Setup

To test our software we configured two laptops with microphones and two Windows CE Pocket PCs. Each laptop was installed with the setup as specified in 3.1 and 3.2 as well as a copy of WinPSK.

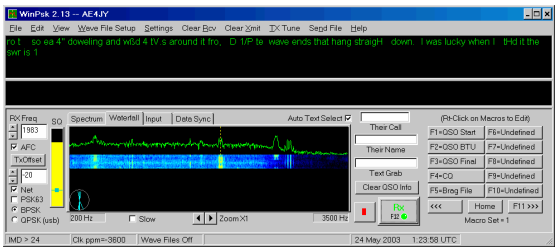

Figure 10: Win PSK

WinPSK is a Windows based program also developed by Moe Wheatley which utilizes the PSKCore dll [9]. This program was used to generate known input and output for testing our software. To ensure we knew the quality of our input we used the PC to generate the audio used to test the receive function as implemented for the PDA. To test transmitting from the

PDA we used the microphone as input to the PC and again running WinPSK captured our PDA output to see how it would read on known software.

3.4. Testing

Once the program properly compiled, testing started with attempting to transmit and receive a single character between the PC and PDA. To monitor the current state of the program, PSKCore provided Windows Messages which provided a status code upon each status change. By printing this message (MSG_STATUSCHANGE) out onto the screen of the program we were able to continuously monitor the current state to evaluate how well the program was running. To capture these messages we had to set up a message handler and place that function within the message map. We could then use this handling function to perform any task based on a given message.

To initialize PSKCore the following commands were necessary:

```
fnInitPSKLib();
fnStartSoundCard(m_hWnd, 0, 1);
fnEnableRXChannel(1, 1);
fnSetRXFrequency(3000, 25, 1);
fnSetRXPSKMode(0, 1);
```

When the function fnStartSoundCard is called this immediately put the program in receive mode with the number of channels on the frequency set in the functions above. When in receive mode, the program is constantly taking input from the sound card and checking for valid characters. Once valid characters are found they are sent to a queue and waited to be read. Another Windows Message, MSG_PSKCHARRDY, provided us with the notification that characters were ready to be read from the queue. The function OnCharReady is the message handler we set up to intercept this message and to display the received characters on the screen.

Transmitting through PSKCore only requires adding the characters you want to transmit to a queue and then running a transmit function. Since PSKCore is able to be ran in several different modes, you mush specify how you would like the data to be sent. To help generate a clean transmission we tested all the different transmit modes. We also worked at sending a single character at a time or simple string all at once to see how the program would react to each.

There were also many issues with stopping the program. Just exiting often left the PDA confused and it was unable to return to the previous program. This meant we had to find a way to easily shutdown the program safely after use. To do so we linked a function to the "ok" button that is responsible for closing the application. This function made the following calls:

```
fnStopSoundCard();
fnTermPSKLib();
CDialog::OnClose();
```

Once the program was properly shut down, the PDA could return to normal use.

4. RESULTS AND LESSONS LEARNED

We divided up the testing into its two natural parts: transmission and receiving. We tested each independently of the other starting with the transmit function. The results are below:

4.1. Transmit

The initial transmission through our program gave marked the successful use of the PSKCore functionality. Unfortunately, it did not see the end of this particular part of the project. When we ran our program with one set of input to be transmitted and compared the produced audio to that of WinPSK we noticed a large discrepancy. The BFT-M testing audio was choppy and was not able to produce a whole string with out creating breaks in the audio stream. This caused apparent problems with being able to receive the broken data.

Our first attempt to remedy this problem was to simply send a string a single character at a time with a 3 second delay between each character. This delay would allow the PDA to complete the calculations for the previous character before attempting to send the next. This method also resulted in producing choppy output as it got to the later characters in the string it was still trying to send.

We then attempted modifying each of the initial two ideas by switching between the different transmit modes and varying the delay between each character, but were still unsuccessful at consistently generating usable audio. Analyzing the data that we would be using with this project (typically only GPS positions), we decided that another solution would be to prerecord the needed characters and simply have BFT-M play each as a .wav

file. To accomplish this we used the .wav output function on WinPSK and recorded a few test digits. After modifying our interface and transferring the necessary files we were able to test our new theory. Our results were mixed occasionally receiving a clean signal then other times receiving complete gibberish. Even still this proved to be the most successful transmission scheme we tested.

4.2. Receive

Our initial test of the receive function was to allow the program to run and once characters were decoded to pop-up a text box displaying the received characters. This particular idea proved to be extremely slow and often caused problems in shutting down the program due to its attempt to open several pop-up text boxed for each instance it decoded a character.

For the next series of tests on the receive function we again used the PC and WinPSK to generate our input, but changed to a higher frequency, usually 3000Hz, to avoid noise noticed in WinPSK's Spectrum analysis. Our testing program's user interface also changed to consist of a static text box which would be edited to show the current received characters and a checkbox which indicated new characters were being decoded. After running some initial input into the PDA we found that if the message was sent just after the program started running, then the intended message had a greater chance of being received correctly. However, it often took a long period of time—upward of a minute—before the message was decoded and displayed on the screen.

To investigate the problem further we edited the checkbox label to display the current status of the program using the MSG_STATUSCHANGE as described in section 3.4. After running the program several times we found that in every case the program eventually went into state 2, "CPU too slow or busy", and remained for a long period of time. It seems the processor was unable to keep up with the data being supplied by the soundcard forcing the program to restart the soundcard and dumping the current data in an attempt to salvage the program and keep it running on new input. The main problem is that it never seems to come out of this state and revert back into receive mode.

4.3. Final Results

Due to the lack of floating point hardware and the processor speed of current hand held devices we were unable to adopt a software solution to connect handheld devices through a long range wireless network. PSK31, as implemented in this project, simply proved to be too taxing on the handhelds hardware to be a viable solution. Even if characters could be sent and received successfully, the handheld would be to overwhelmed to process any of the other data required to have the functionality needed to run the rest of the components of BFT-M. This however does not mean that the BFT-M is not possible, but simply that it may not be quite as simple as anticipated to rely on software for encoding/decoding data for transmission through a VFH radio.

Figure 11: Test interface showing signal receipt with noise.

4.4. Lessons Learned

When starting on this project, I thought that much of the time would be on developing the interface for users to interact with the YP-TDS environment. I felt that with the tools available, primarily PSKCore, the networking issue would not be the primary focus of my project. Though this did not cover the problems I was expecting to solve, it did allow me to expand my horizons into an unexpected but exciting field that otherwise I would not have explored. Networking through a HF/VHF radio was not something I had heard of before and it was interesting to see the capability such a network would have.

My experience with C++ and Java, primarily through SI221 and SI321, is what gave me most of the tools to work on this project. Much of what I did not know was dealing with the PSK31 encoding and decoding as well as understanding the Windows system calls and variables that allow messages to be passed within the operating system. This basic understanding of how to program in a Window environment will greatly help my ability to produce useful software in the future. Much of Computer Science is knowing the tools and when to use each one and I believe that this project expanded my tool set greatly allowing me to become a more diverse programmer.

5. FUTURE WORK

These results do not show the end of a solution using PSK31. The next step of this project had we more time was to go through and eliminate some of the extra functionality of PSKCore to help save some computational steps. Some of these tools include "Frequency Error Detection/Correction", "Frequency Error Filter", as well as the spectrum analysis tools provided to visually graph the signal. These tools are integrated into the PSKCore code making it difficult to separate them, however by doing so we might be able to help the processor spend more time on the actual encoding/decoding and not as much on these other tools which do not have immediate use to us. The key to solving this problem is processor speed and efficiency.

Other approaches to achieving this network are available to explore and each offer a great deal of potential. The first option is a hardware solution. Currently there are amateur radio operators working on portable hardware to be able to send data directly from a keyboard [10]. This option is possibly the most appealing since it would lay the burden of encoding and transmitting to an external device so the PDA would be able to actively run the simulation software. One such example is "Portable PSK" by George Heron who has a working portable PSK unit. Though it is large and requires AC power to run, it does show that such a solution is possible.

Figure12: PSK in hardware

Another option would be to set up a 802.11 wireless network for the handheld devices. The immediate drawback to this approach is the range that an internal Wi-Fi card can operate. With the shortened range, base stations would have to be established to pickup and forward characters to and from each handheld device. The second drawback is power consumption. Using external hardware to handle data communication would allow a large battery to be used allowing longer connection times. Though bulky, this is not too different from what current field radio operators deal with in the Fleet/Marine Corps. By relying on the small battery to power a Wi-Fi card to maintain connection, we would greatly shorten the time spent in the field training by being required to recharge the handheld units once the battery runs out.

Once we are able to successfully network handheld devices over a long range network we would be able to then develop the software to interface the handheld network with YP-TDS to develop a true Network Centric Warfare training environment. The network interface will have to be done on the YP-TDS side with the Master Server. The Master Server can then take this information and distribute it to the rest of the YP-TDS network. Since PSKCore does work without problems on a PC as shown with WinPSK and allows multiple receive channels, all handheld devices could communicate to the Master Server through a different channel to prevent collisions on the network.

A major starting point would be to simply allow the entire YP-TDS network to view the location of the BFT-M network with the handheld devices only sending their current position over the network and not receiving any data. Once this can be established with some reliability, we can then move to allowing BFT-M to send and receive simple strings over the network as simple messages. Though this does not provide a full digital battle field command, it does allow the YP-TDS users to pass instructions to the BFT-M network. Once we are able to truly send and receive in full duplex mode, we can move on to a more interactive user interface which would allow users on the BFT-M network to view locations of all known contacts, much like the current YP-TDS system, but with simpler graphics.

6. REFERENCES

[1] Department of Defense, Quadrennial Defense Review Report, Washington, DC, September 2001.

[2] Cebrowski, A.K., "The Implementation of Network Centric Warfare", Department of Defense Office of Force Transformation, January 2005.

[3] Mahnken, T. and FitzSimonds, J., "The Limits of Transformation. Officer Attitudes toward the Revolution in Military Affairs", Newport Papers #17, Naval War College Press, 2003.

[4] *YP-TDS A Tactical Data Simulator for USNA Yard Patrol Craft*, Jeffrey P. Wilcox. 2003.

[5] www.psk31.com/G3PLXarticle.pdf (PSK31: A new radio-teletype mode with a traditional philosophy)

[6] www.qsl.net/ae4jy/pskcoredll.htm (PSKCore DLL Project)

[7] Prosise, Jeff, *Programming Windows with MFC*, 2nd ed., Microsoft Press, Redmond, WA., 1999.

[8] www.miscrosoft.com (eMbedded Visual C++ 4.0)

[9] www.qsl.net/ae4jy/winpsk.htm (WinPSK Program)

[10] www.njqpr.org/portablepsk (Portable PSK)