

# Amateur Satellite Tracking Communication System

**Junsung Cho** *chojs@cs.utah.edu*

**Vinh Pham** *vpham@cs.utah.edu*

**Suresh Subasinghe** *subasing@cs.utah.edu*

December 17, 2004

## Table of Content

A Brief History of Amateur Satellites.....	p.4
Project Overview.....	p.4
System Hardware and Software Overview.....	p.4
Transmission and Reception Antenna Array.....	p.5
Antenna Specifications.....	p.5
Feed Matching System.....	p.7
SWR & Power Measurement.....	p.7
Radio Specification & Modification.....	p.7
Cabling And Wiring.....	p.7
Preamplifier.....	p.8
Horizontal And Vertical Motors.....	p.8
Relay Circuits for Directional Control .....	p.9
Microcontroller Circuit.....	p.11
Auto Tracking Client.....	p.13
Auto Tracking Server.....	p.14
TNC/Modem Terminal Client.....	p.15
Acknowledgements.....	p.15

## Appendices

### Appendix A

References.....	p.20
-----------------	------

### Appendix B

What is Doppler effect?	p.21
What is the PL-67 Tone?	p.21
Why do satellites use different bands?	p.21
What are the Keplerian Elements?	p.21

### Appendix C

AMSAT-OSCAR 51 Operation Parameters	p.22
Fuji OSCAR 29 Operation Parameters	p.22
AMRAD OSCAR 27 Operation Parameters	p.22

### Appendix D

Other Modification.....	p.23
LNY PreAmp Parts.....	p.23

### Appendix F

Assembly Code for Serial Communication and Relay Control.....	p.25
---	------

Distance Bearing Calculation Code for Auto Tracking Client.....	p.29
Satellite TNC   Terminal Client .NET Code.....	p.31
Space Agent Client .NET Code.....	p.35
Space Agent Server .NET Code.....	p.37

## **A Brief History of Amateur Satellites**

The first amateur satellite, OSCAR I was launched December 12, 1961 piggyback with Discover 36, a United States Air Force satellite. Launch of OSCAR I led to the creation of The Amateur Satellite Corporation (AMSAT) in 1969. However the first remotely controlled amateur satellite was OSCAR 5 which was developed by electrical engineering students at the University of Melbourne in Australia. Amateur satellites continue to be designed and built in countries around the world. There are several new amateur satellites that will soon join the ranks above. Latest being the Amateur Radio onboard the International Space Station (ARISS) in 2004.

## **Project Overview**

Our objective in this project was to design an Antenna/Transmitter module to communicate with Low Earth Orbiting (LEO) amateur Satellites in the sky. We based our design specifically for the OSCAR series satellites AO-51, FO-29, AO-27, and FO-20.

## **System Hardware and Software Design Overview**

The sections to follow will discuss the hardware and software designs and implementations that were used during the course of this project. The overall system components can be summarized below.

### Hardware

- Antenna array for transmitting and receiving radio signal.
- Horizontal and vertical rotators.
- Relay circuits for directional control of horizontal and vertical rotators.
- Transformers to step down the AC power supply to 24V DC.
- Microcontroller to interface with the relay circuit and the computer software.
- Coaxial cables, power cables, CAT5 cables, serial cables etc.
- Joystick for Manual Control override of the antenna array.
- Radio for signal transmission and reception.
- Modem/TNC for decoding/encoding data packets.
- Adjustable DC power supplies to power up the Radio, Modem and Microcontroller.
- Amplifiers for the relay circuit.
- Amplifier for signal amplification

### Software

- Client side software for broadcasting co-ordinates to the server and interfacing with satellite coordinate software.
- Server side software for controlling the antenna array.
- Terminal software to connect to the TNC/Modem and monitor data flow.
- Assembly software for programming the microcontroller.
- Satellite co-ordinate tracking software.

## Transmission and Reception Antenna Array

This satellite communication system uses two antennas, each operating at VHF<sup>1</sup> and UHF<sup>2</sup> range of the spectrum. The VHF antenna is the primary transmitting antenna to up link with various satellites. The UHF antenna is used as the downlink antenna.

- VHF( 2 m ): 145 MHz ~ 146 MHz ( 6 elements )
- UHF(70 cm): 435 MHz ~ 437 MHz ( 11 elements )

Both antennas are directional (YAGI)<sup>3</sup>, constructed from stainless steel using gamma matching method



**Figure 1:** Antenna array mounted on a tripod with azimuth and elevation control rotators

## Antenna Specifications

**Table 1** below contains the design specifications for the VHF YAGI antenna, and **Table 2** contains the design specifications for the UHF YAGI antenna. These parameters were calculated based on the formulae provided in the “*The ARRL Antenna Book*”.

---

<sup>1</sup> Very high frequency (VHF) is the radio frequency range from 30 MHz (wavelength 10 m) to 300 MHz (wavelength 1 m).

<sup>2</sup> Ultra high frequency (UHF) radio frequencies are those between 300 MHz and 3.0 GHz

<sup>3</sup> A Yagi-Uda Antenna, commonly known simply as a Yagi antenna, is an antenna consisting of an array of a dipole and additional parasitic elements. The dipole in the array is driven, and another element, slightly longer, operates as a reflector. Other shorter parasitic elements can be added in front of the dipole as directors. This arrangement gives the antenna directionality that a single dipole lacks. Yagis are directional along the axis perpendicular to the dipole in the plane of the elements, from the reflector through the driven element and out the director(s)

YAGI ANTENNA DESIGN DETAILS ( VHF )	
DESIGN FREQUENCY	145.92 MHz
BAND	2M Band
GAIN (With -15 db side lobes)	8.88 DBD
NUMBER OF ELEMENTS	6
DIAMETER OF ELEMENTS	4.00 mm
DIAMETER OF BOOM (O.D.)	22.00 mm
BOOM LENGTH	1873.69 mm
HORIZONTAL BEAM WIDTH	52 degrees
VERTICAL BEAM WIDTH	70 degrees
DIMENSIONAL TOLERANCE (+/-)	6.16 mm

Element	Length (mm)
REFLECTOR	1016.98
FOLDED DIPOLE	971.78
DIRECTOR # 1	935.88
DIRECTOR # 2	926.21
DIRECTOR # 3	914.08
DIRECTOR # 4	904.34

element spacing (center to center)	mm
REFLECTOR ~ FOLDED DIPOLE	380.08
FOLDED DIPOLE ~ DIR 1	162.30
DIR 1 ~ DIR 2	369.81
DIR 2 ~ DIR 3	443.77
DIR 3 ~ DIR 4	517.73

Table 1: VHF YAGI antenna design details<sup>4</sup>

YAGI ANTENNA DESIGN DETAILS (UHF)	
DESIGN FREQUENCY	435.30 MHz
BAND	70cm Band
GAIN (With -15 db side lobes)	12.27 DBD
NUMBER OF ELEMENTS	11
DIAMETER OF ELEMENTS	4.00 mm
DIAMETER OF BOOM (O.D.)	22.00 mm
BOOM LENGTH	1714.18 mm
HORIZONTAL BEAM WIDTH	30 degrees
VERTICAL BEAM WIDTH	34 degrees
DIMENSIONAL TOLERANCE (+/-)	2.07 mm

element	length (mm)
REFLECTOR	349.86
FOLDED DIPOLE	325.76
DIRECTOR # 1	314.16
DIRECTOR # 2	310.18
DIRECTOR # 3	305.21
DIRECTOR # 4	301.24
DIRECTOR # 5	299.25
DIRECTOR # 6	297.26
DIRECTOR # 7	295.26
DIRECTOR # 8	295.26
DIRECTOR # 9	290.30

element spacing (center to center)	mm
REFLECTOR ~ FOLDED DIPOLE	127.41
FOLDED DIPOLE ~ DIR 1	54.41
DIR 1 ~ DIR 2	123.97
DIR 2 ~ DIR 3	148.76
DIR 3 ~ DIR 4	173.55
DIR 4 ~ DIR 5	193.52
DIR 5 ~ DIR 6	207.99
DIR 6 ~ DIR 7	218.32
DIR 7 ~ DIR 8	227.96
DIR 8 ~ DIR 9	238.29

Table 2: UHF YAGI antenna design details<sup>4</sup>

<sup>4</sup> Elements are NOT INSULATED from the boom

## Feed Matching System

Each antenna was tuned using the gamma matching method. The gamma match capacitor can only cancel reactance, it can not modify the "real part" (resistance) presented to the feed-line. It is the simplest form of matching, and has the lowest operating Q and loss of any system. Adjustment of resistance requires adjusting the diameter, spacing, or length of the gamma section.



Figure 2 gamma match capacitor

## SWR & Power Measurement

We used a SWR9 VHF/UHF Dual Band SWR/Power meter from VANCO for tuning the antennae.



Figure 3 Dual Band Power meter

## Radio Specification & Modification

A TM-741 Kenwood multi band Radio transceiver was used for connecting to the antenna system. This radio operated in the Frequency Ranges 144MHz ~ 147MHz and 434MHz ~ 450MHz

We were able to send/receive voice communications using this unit. However because it does not support a 9600 bps TNC/Modem connection for data transfers, we had to modify the radio to receive an unfiltered signal from discriminator. This was done by measuring the incoming signal on an oscilloscope and soldering a wire at a node before the signal was filtered (see *Appendix D section 1*)



Figure 4 TM-741 Kenwood radio

## Cabling and wiring

We used two, 200 feet long 50 ohm coaxial cables to carry radio signals to and from the antenna to the radio. To power up the two rotators and carry directional control signals to them a CAT-5 Ethernet cable was used. Serial communication cables were used for connecting the transceiver, motor driver control circuit and the tracking server.

## Preamplifier

During the initial testing of the antenna reception, we observed that the incoming signal to the transceiver was much weaker than anticipated. This could have been due to the atmospheric interferences or simply because of a weak transmission signal. The length of the cables being used was also a major factor in signal loss.

As a solution to this problem we implemented an inline preamplifier based on the LNY receiver preamplifier circuit design<sup>5</sup>.

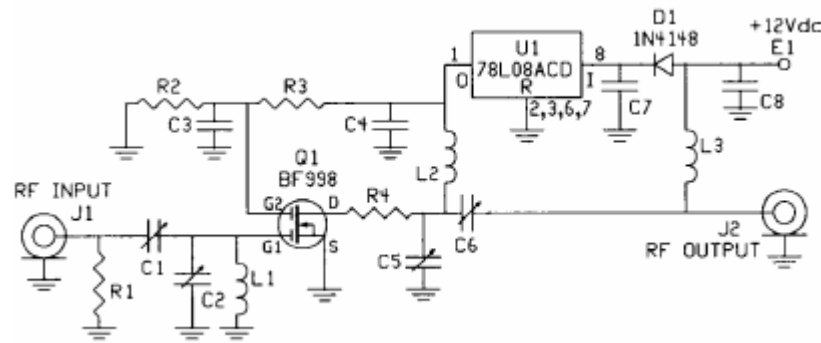


Figure 5 : LNY receiver preamplifier circuit

This amplifier circuit was placed at the antenna before connecting to the coaxial cables, and it improved the receiver signal quality significantly. Following this modification we were able to listen to a wide range of satellites for voice and data including the amateur radio onboard the International Space Station.

Prior to these modifications we had the added advantage of being able to use both antennae as transmitters. However after placing the amplifier circuit in place the UHF antenna could no longer be used as a transmitter anymore.

<sup>5</sup> <http://www.hamtronics.com/pdf/inst%20manuals/LNY%20Preamp.pdf>



## Horizontal and Vertical Motors

Ideally we would have liked to construct the antenna array controller using stepper motors. However, because we were designing our system for outdoor use and because of budget constraints, we decided to use normal rotator motors.

A total of two rotators were used, one for elevation control of the antenna array and another is for azimuth control of the antenna array. Each rotator could be controlled for direction and they required 24V to operate. Directional control signals had to be in the order of 5~10V.

We were able to borrow one rotator from the Electrical Engineering department and purchase the other rotator. Each rotator also came with a transformer/controller box that could be plugged in to a wall power supply. These controllers use AC synchronous motors that run at a predictable speed, and the control box simply runs the motor for the amount of time needed to turn. We also borrowed a tripod from the Electrical Engineering department to mount the two rotators.

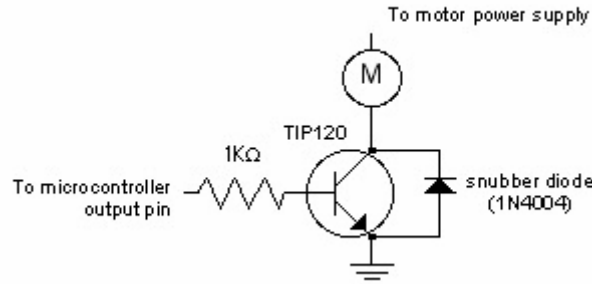


**Figure 6:** Antenna Rotator with transformer and controller.

## Relay Circuits for Directional Control

The rotators consume about 3~4 Amperes on average depending on the load that is being mounted on them. Each rotator could be powered with 24V DC; however the directional control of the rotators also requires a significant amount of current even under a low voltage.

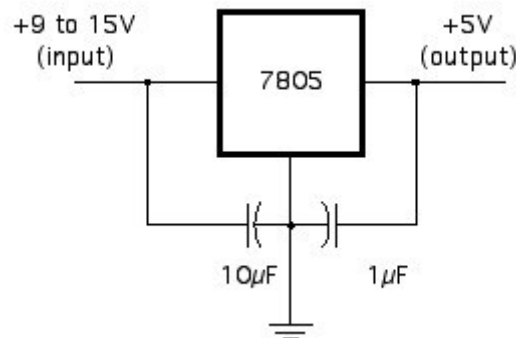
Therefore in order to conduct the high current and turn on the motors using microcontroller output signals which were very weak, we designed an array of transistor switches connected to Voltage regulators and relays for each directional control signal.



**Figure 7 :** Each microcontroller output pin is connected to a transistor switch circuit

Each directional control signal requires 5 Volts to become active. The voltage output of the transistor switch is connected to a 7805A voltage regulator to supply a constant 5V output to the relay switches.

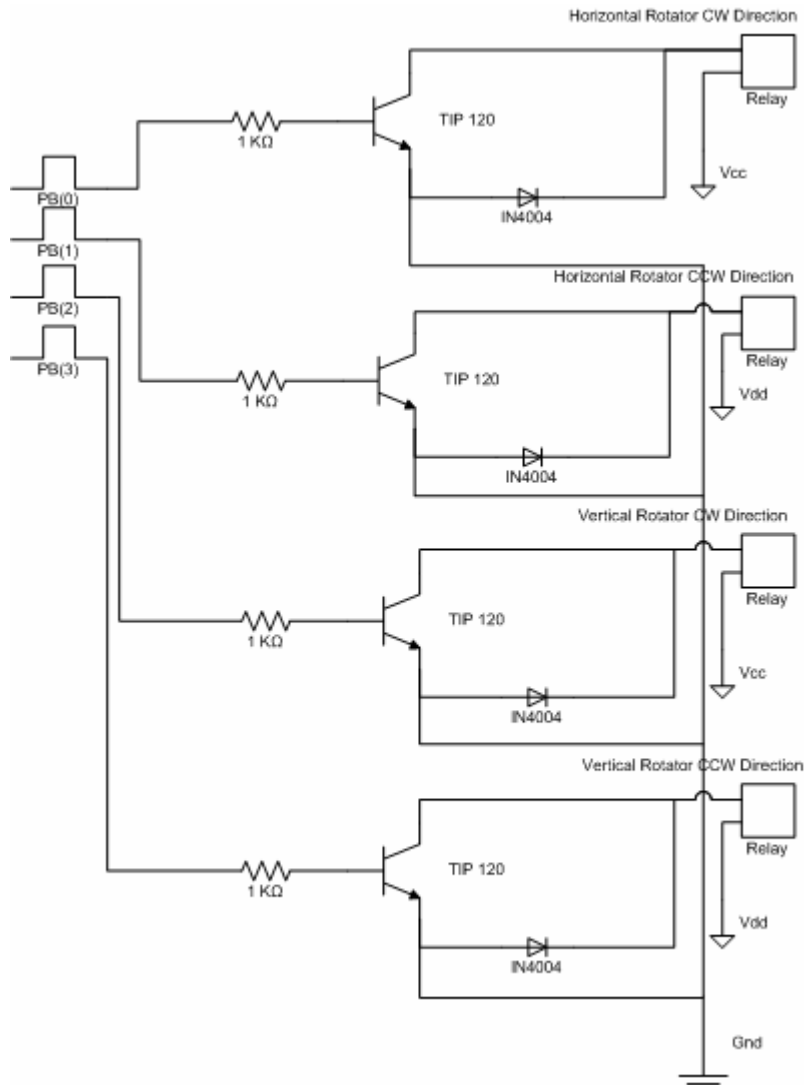
The relay switch is connected to one end of the voltage regulator output and the other end is connected to the directional control pins of the rotator. The two capacitors on either side of the regulator smoothen out the voltage, as the relay switch and the motor will cause spikes and dips when it turns on and off.



**Figure 8:** Voltage regulator circuit to smoothen out the spikes caused by the switches & motors.

In order to power up the two rotators we used the original controller that came with them, and modified the circuits for our needs. We bypassed the built-in timer and the dial controller so that we could use the built in transformer. The only use of these controller boxes was its transformers. Directional control pins were connected to the microcontroller outputs.

**Figure 9** shows the overview of the motor driver control circuit. On the left microcontroller output signals control the transistor switches. In turn they turn the relays on and off causing the rotators to move in any given direction individually or simultaneously.



**Figure 9** : Overall view of the motor driver control circuit

### Microcontroller Circuit

We used a Motorola 68HC11E1 series microcontroller to control the Motor driver control circuit. The MCU was mounted on an evbplus2 development board with 30KB RAM, 8KB EEPROM and onboard debugging capabilities.

The primary task of the microcontroller is to communicate with the host program via its RS232 interface and control the motor driver circuit with its output signals accordingly. The host program issues a set of commands that the MCU interprets and moves the motors in a certain direction.

These commands (please refer to **Table1**) allow the basic four way motion of the rotators by 1 degree, and they can be used in any combination to achieve diagonal motion or

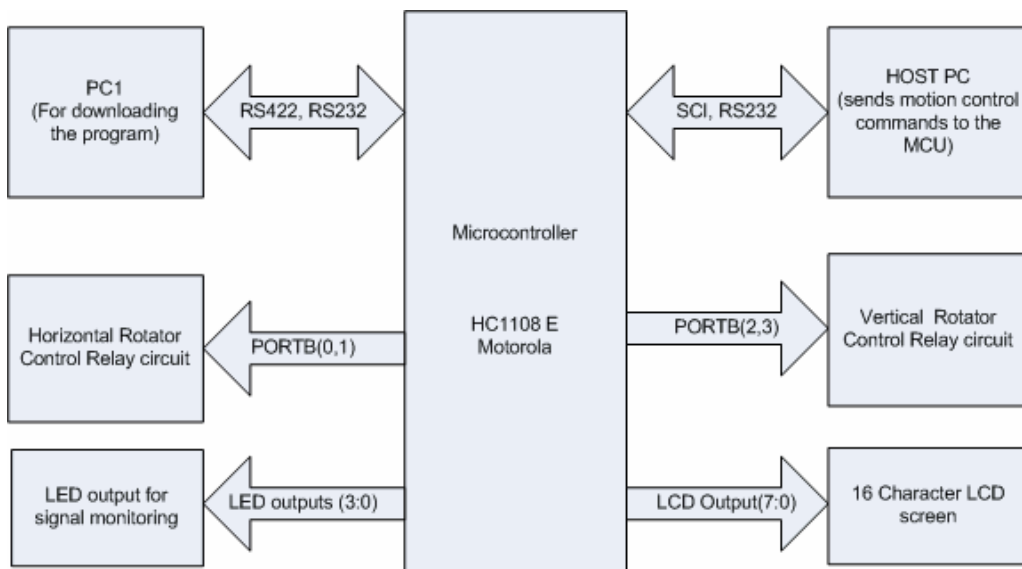
another preconfigured pattern. The MCU also responds to each command by sending an acknowledge signal back to the host program after every execution.

command	Action
<b>A</b> (Acknowledge)	MCU sends a READY message back to the host. This is used for checking the MCU status.
<b>U</b> (up)	MCU signals the elevation control rotator to move up by 1 degree. When it's done moving, an acknowledge signal is sent back to the host.
<b>D</b> (down)	MCU signals the elevation control rotator to move down by 1 degree. When it's done moving, an acknowledge signal is sent back to the host.
<b>R</b> (right)	MCU signals the azimuth control rotator to move clockwise by 1 degree. When it's done moving, an acknowledge signal is sent back to the host.
<b>L</b> (Left)	MCU signals the azimuth control rotator to move down counter clockwise by 1 degree. When it's done moving, an acknowledge signal is sent back to the host.

**Table 3** : Control commands interpreted by the MCU

The MCU is programmed and calibrated to control the timing of the directional control signals to move it exactly by 1 degree in each direction. It also makes use of the onboard LCD display and LED display to notify the status of each signal and controller status. We used these feature for monitoring the control system and debugging firmware.

Please refer to *Appendix F section 1* for the assembly program used in the MCU



**Figure 10** : Block level representation of the MCU and its interfaces to other peripherals

## Auto Tracking Client

During the research phase of our project we found two viable methods of obtaining live satellite coordinate data. The primary one being the web based Java applet *J-Track* that ran on NASA's website. The second option was *NOVA for windows* from <http://www.nlsa.com>. Both were free for use; however we decided to choose *NOVA* because of its ability run on the local host. This gave us more control over the software and allowed us to integrate it in to our client module without much hassle.

*NOVA* provides a user friendly GUI interface to display available satellites and their footprints. It also allows script tracking (ability to track a given list of satellites in order). For us to use *NOVA* as part of the client side software, we designed a software module that was able to interface with *NOVA* and extract the data it was displaying on the screen. This was done using a technique called DDE (Dynamic Data Exchange)<sup>6</sup>. Once the information required is extracted, it would then connect to the Tracking software that is running on the server side and transmit the data using TCP/IP over the internet.

The data that is being transmitted to the server from the client side is formatted as a continuous string value, where each field is delimited by a space character.

**SatName AZ:Azimuth EL:Elevation RR:RangeRate** where

**SatName** = the current auto tracking satellite name as derived from the AMSAT database, maximum 12 characters;

**Azimuth** (AZ) = current satellite azimuth, 0.1 degree precision, no sign;

**Elevation** (EL) = current satellite elevation, 0.1 degree precision, signed;

**RangeRate** (RR) = satellite rate of change of distance from the observer expressed as a signed floating point number in units of  $\frac{1}{\text{Speed of Light}}$

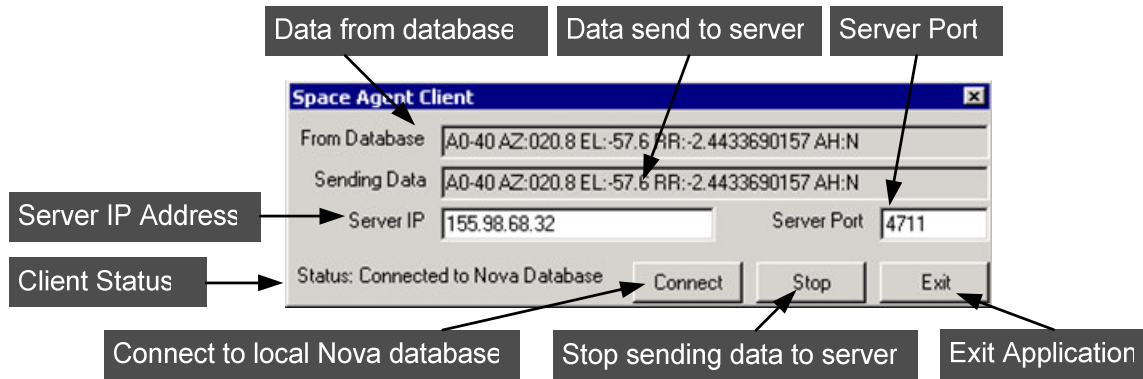
Doppler Frequency (Hz) = -Freq (Hz) \* RR \* (1.0/299792.458)

**Table 4** : Explanation of the data transmitted by the Auto Tracking Client to the Auto Tracking Server and Doppler Frequency calculation based on the Range Rate value.

The client software module has a GUI designed using Microsoft .NET SDK framework. **Figure 11** shows a screenshot of the Auto Tracking client connected to *NOVA* running on the client side using DDE and to server over the Internet using TCP port 4711.

<sup>6</sup> DDE enables two running applications to share the same data. For example, DDE makes it possible to insert a spreadsheet chart into a document created with a word processor. Whenever the spreadsheet data changes, the chart in the document changes accordingly.

Please see *Appendix F section 4* to view code for the Auto Tracking Client module.



**Figure 11** : Screenshot of the Auto tracking Client software

### Auto Tracking Server

Once the Auto Tracking client selects a satellite and begins transmitting coordinates, the Auto Tracking Server running on the server side will decode the incoming data and send a series of commands to the MCU which in turn positions the antenna grid accordingly, and follows the path as the satellite as it moves. The server was designed using Microsoft .NET SDK framework. *Figure 12* shows the tracking server in action.

Apart from its ability to automatically control the antenna array, the tracking server also allows the user to manually take control of the motion using the buttons on the GUI or a joystick. This is a useful feature that comes in handy when the software is initially started, because it assumes that the antenna grid is positioned at zero, zero.<sup>7</sup>

The software also allows the user to customize the boundary limits of the antenna movement, which can be useful when there are rotator constraints to be concerned about. If for some reason the server stops receiving any updates from the client, it will automatically position the antenna grid to its default position and realign to zero, zero.

Please refer to *Appendix F section 5* to view code relevant to the Auto Tracking Server software module.

<sup>7</sup> Default positioning of the antenna grid is assumed to be 0 degrees elevation and 0 degrees azimuth.

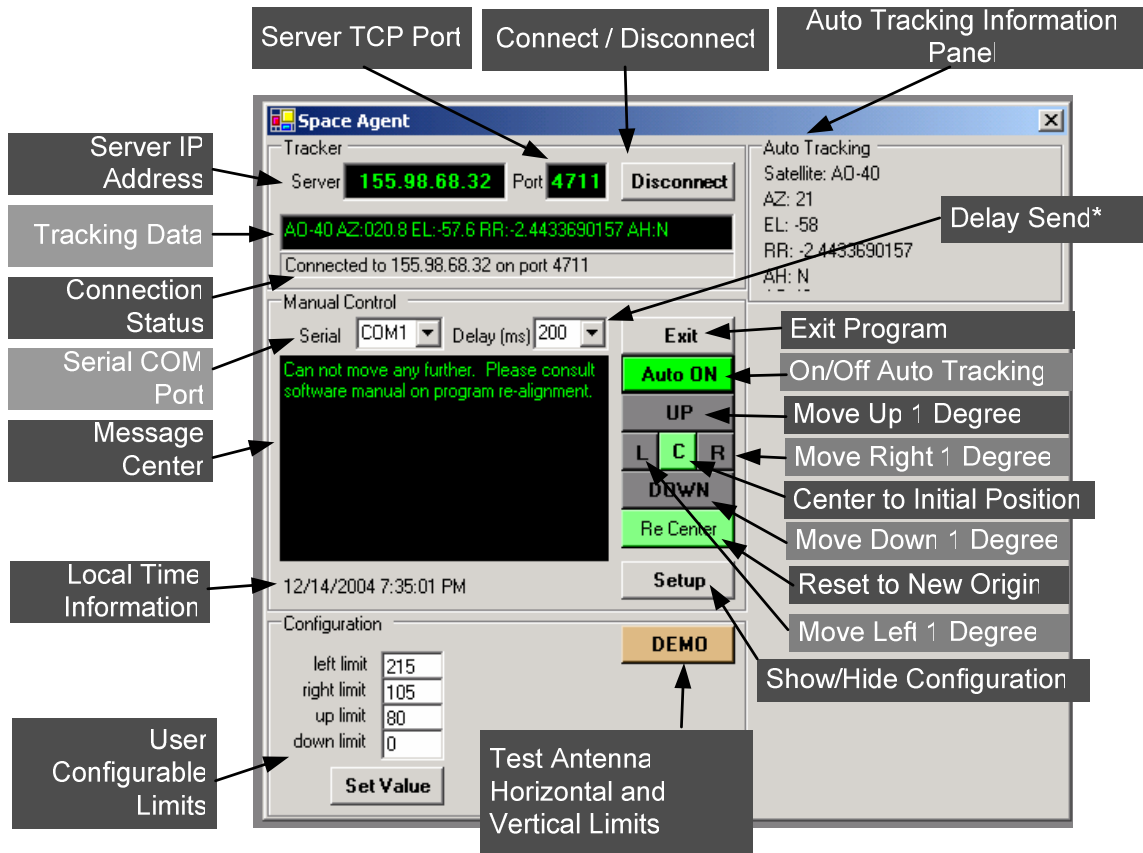


Figure 12 : Screenshot of the Server side tracking software

### TNC/Modem and Terminal Client software

The TNC/Modem facilitates the decoding of a packet radio signal. We used a PacComm SPIRIT-2 TNC/modem. In order to view the data that is being decoded, we designed a terminal client software program that was able to connect to the modem via its serial interface and download the data. The terminal client connected to the modem using the settings (9600, N, 8, 1). **Table 5** contains a list commands that are recognized by the modem that can be entered via the terminal program in order to successfully send and receive data packets. **Figure 13** shows the terminal program in action.

command	argument	Action
<b>MY</b>	Callsign	Set callsign unique ID.
<b>BEACON</b>	Every After n	Send alive signal every 'n' seconds.
<b>CONNECT</b>	Callsign string	Command requires callsign. We acquired a callsign previously used by other University student.
<b>List</b>		Lists the 10 latest messages.
<b>Read</b>	N	Reads message number n.
<b>Send</b>	data	Send a message.
<b>Bye</b>		Disconnects from PMS.

Table 5 : Modem Commands

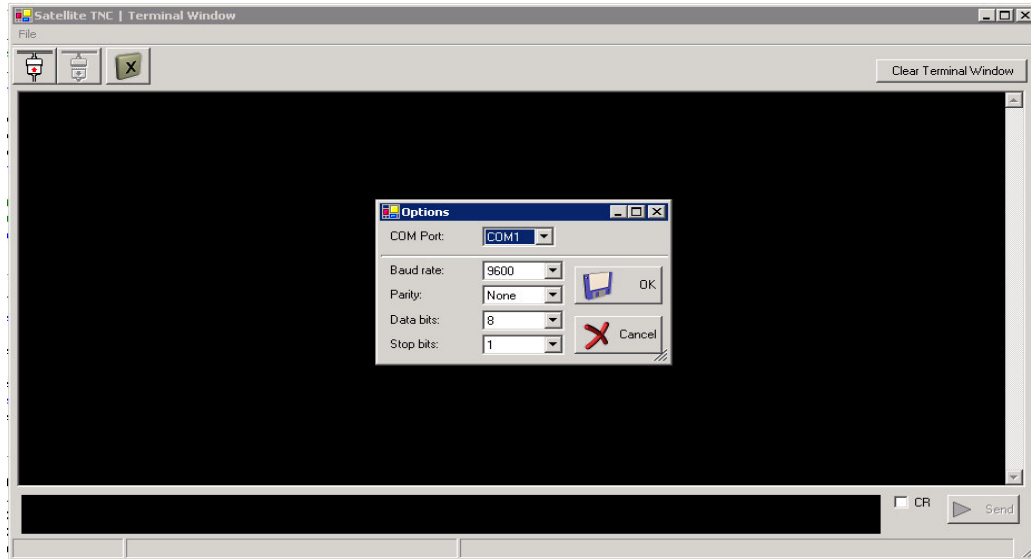


Figure 13 : Screenshot of the TNC | Terminal program

*Satellite TNC Terminal Window* program was designed using the Microsoft .Net SDK framework. This program uses a COM port to communicate with the modem. It could be run on the server side computer. However we were able to run it on the client side computer using virtual serial port mapping supported by Windows Remote Desktop.

Please refer to *Appendix F section 3* to view code relevant to the Auto Tracking Server software module.

## Using the system to communicate

### Setting up the Auto Tracking Server

1. Initialize the Antenna Grid to so it is positioned at zero, zero.<sup>8</sup>
2. Turn on the Motor Driver Control circuit and power up the microcontroller. When the microcontroller is ready, it will display a message on the LCD screen saying "Controller Ready, reading signal"
3. Start the Auto tracking Server program. When the program is first launched it will start querying all the serial ports and try to determine the COM port the microcontroller is connected to. If an error message is displayed make sure that the serial cable to the microcontroller is connected and that the microcontroller is powered up.

---

<sup>8</sup> Default positioning of the antenna grid is assumed to be 0 degrees elevation and 0 degrees azimuth.



### Setting up the Client Software

4. Start the NOVA client program.
5. Start the Auto tracking Client program and click on the *Connect* button. If there is an error message, make sure that the port number is the same port that the server is listening to.
6. The Tracking server will continue to track the selected satellite until it is out of range, to select a different satellite simply click on the orbit of another satellite from NOVA or wait until the same satellite comes back in to range.

### Setting up the Radio and Modem

7. Set the proper frequency for a beacon signal which is just a beep.
8. Open filter for listening all sound. Set the transceiver on the lowest power.
9. Set the PL - 67 KHz Tone Signal for opening a channel for the satellite.
10. If a beacon signal can be heard, adjust the beacon frequency to an operational frequency which is considered by Doppler Effect. Transmit a short signal and if you can hear it back from the receiver side after a short delay, the radio is correctly setup and is ready for communicating with the satellite.
11. For voice communications, listen to the radio and use the handset to transmit.
12. For data communication, turn on the Packet Modem and launch the *TNC terminal Program*. By typing commands in **Table 5** you can send and receive data from digital satellites.

### Satellite Communication Etiquette

- Listen for the satellite beacon or other operators before transmitting.
- Use as little power as needed to complete the QSO.
- When pileups occur, give your information (callSign, gridSquare) quickly.
- Be courteous to other operators.

## **Acknowledgements**

We cannot fully express our gratitude to everyone who helped us coordinate our efforts to put this project together. For their generous assistance in lending equipment, providing access to the roof and facilitating other needs we are thankful to Corey from the DSL lab, Ernie Flynn, Marvin Match (Laboratory Supervisor / PC Computer Specialist), and Bob ( carpenter / Door Shop – supervisor ).

Our gratitude goes to Professor Al Davis, for his superb guidance and keeping us on track to complete this challenging project.

## **Conclusion**

When we first started this project, satellite data communication was a fairly new subject for us, and all we had was a radio transceiver (which is how we came up with the idea). As a team we had little experience in radio communication, packet decoding and networking. During the first semester, we spent time researching the AMSAT website and other ham radio related websites trying to understand the process of communicating with satellites. It was a great learning experience for most of us, because in general we thought that satellite communication is not something that the public was able to do without special equipment, licenses and possibly paying a fee.

Once we came up with a project plan and submitted the proposal to design and build a satellite communication system with auto tracking capabilities, we spent the summer collecting parts, designing circuit layouts, and assembling various equipments.

Even though we started off with four team members, by the end of the summer one member dropped out. After a swift reassignment of tasks, we were able to make headway in our overall project. Antenna design was completed by the end of the summer and we had partially built our motor control system by that time. After the preliminary phase of troubleshooting and debugging we had an antenna and the controller system in place for the software to be written. We spent the last two months of our project writing the Auto Tracking Client, server and TNC Terminal software programs and testing them with the complete system.

During these testing phases we ran in to some minor and critical problems. On the antenna system, we noticed significant signal loss shortly after setting it up. We had to build an amplifier to be placed inline with the coaxial cable, and this solved most of the signal loss problems.

The physical limitations of the rotators were overcome by reconfiguring the software to work around these limitations. For example the rotators could not continuously rotate around its axis; instead they have to rotate back and forth the opposite directions. This was more evident when we were tracking a satellite with azimuth starting from 90 degree or higher to about 270 degrees.

The mountains surrounding our location also caused blackout areas for radio reception. Especially when covering the South East hemisphere.

Another problem we had was repositioning the antenna array when the system is being initialized because we did not use an active feedback system. We overcame this problem by allowing manual override of the control system using a joystick controller, which proved useful when debugging the control software to protect the motors.

Both the above problems can be overcome if we were to use a stepper motors in place of the rotator motors.

During mid semester we found out that some of the satellites were not operational, however we were still able to communicate with AO-51, AO-27, FO-29. Once the system was operational this was not a big concern, however during testing periods we had to wait up to 5 hours before another available satellite came into range to test our system.

Even though it was a minor problem, having access to the roof was quite critical during the testing phase of our antenna system. There were occasions when the motors got stuck because of software bugs, or had to readjust the antennae, and we had to wait up to five days before we could make a simple fix. This was valuable testing time for us. Once again had we used stepper motors this probably would not have been a big concern.

While dismantling the antenna system we found some problems that we did not notice before. We found water inside the coaxial cable connectors that can contribute to noise. Later we found that grounding the entire system including the antenna, controller and PC can help improve the signal quality.

We also got some recommendations that suggested we might be able to get better performance from our inline amplifier, if we connected it as close as possible to the antenna itself.

After standing on the roof for about a month going through warm and cold weather cycles, and constant movement, we noticed that the pipes connected to the antennae had loosened. This caused vibrations when the antennae array was rapidly moving while trying to lock in to a satellite orbit. We overcame the vibration by allowing some time to settling between moves. Another fix for this problem would be to use some other material instead of plastic pipe to hold the antenna and securing it.

In conclusion we were quite happy with the overall system performance despite the minor drawbacks. Each team member lived up to their expectations and delivered when it came to completing their assigned tasks. We successfully communicated with about ten different satellites including the International Space Station.

We hope that our project will raise awareness and boost interest in amateur satellite communication amongst others and that our work will be a basis for designing their satellite communication systems.

## Appendix A

### References

1. The ARRL Antenna Book (19th Ed./Book & CD-ROM) by R. Dean Straw
2. The Radio Amateur's Satellite Handbook published by The American Radio Relay League.
3. <http://en.wikipedia.org>
4. <http://www.nlsa.com/index.html>
5. <http://science.nasa.gov/RealTime/JTrack/>
6. <http://www.arrl.org/>
7. <http://www.sat-net.com/winorbit/index.html#winorbit>
8. <http://www.tele-satellite.com/>
9. <http://www.amsat.org/amsat/sats/n7hpr/satsum.html>
10. <http://www.tbs-satellite.com/tse/online/>
11. <http://www.amsat.org/amsat/ftpsoft.html>
12. <http://www.sao.ac.za/~wpk/index.html>
13. <http://home.nycap.rr.com/capcom/fsattrak.html>
14. <http://home.hiwaay.net/~wintrak/>
15. [http://www.riverland.net.au/~hutchja/tracking\\_antenna.htm](http://www.riverland.net.au/~hutchja/tracking_antenna.htm)
16. <http://web.usna.navy.mil/~bruninga/astars.html>
17. <http://www.qsl.net/kd2bd/predict.html>
18. <http://www.amsat.org/amsat/instanttrack/sattrack.html>
19. <http://www.jpl.nasa.gov/releases/96/leoterm.html>
20. <http://www.kantronics.com>
21. <http://www.shpc.pe.kr/sbaycom.html>
22. <http://users3.ev1.net/~medcalf/>

## Appendix B

### Satellite FAQ

#### What is Doppler Effect and how does it impact satellite communication?

The *Doppler Effect* in satellite communications is the change in frequency of an electromagnetic signal that results from the relative speed of the satellite and the Earth terminal. When the orbital parameters of a satellite are known, Doppler shift can be used to determine the position of the Earth terminal. When an Earth terminal's position is known, Doppler shift can be used to estimate the orbital parameters of a satellite. When the satellite (or the Earth station) is moving rapidly, the Doppler Effect is an important consideration in satellite communications.

For example, for a working satellite frequency of 436.795 MHz, it will begin around 436.805 at Acquisition of Signal (AOS). At the center of path, it will reach 436.795MHz and eventually reaching 436.785MHz at Loss of Signal (LOS). The transceiver we used does not have the ability to automatically adjust with the Doppler frequency shift. *Table 4* shows how the Doppler Frequency is calculated using the Range Rate value.

#### What is the PL-67 Tone?

It is a 67 KHz signal used for protecting the receiver circuit onboard satellites from unexpected ground station transmissions. For instance, when trying to transmit a signal to the AO-51 satellite, a tone signal should be loaded with the main signal. Most FM radios have this functionality.

#### Why do satellites use different bands?

Satellites do not have the physical space to separate receiving and transmitting antennae a great distance, therefore they use different bands.

#### What are Keplerian Elements?

*Sample Keplerian Elements for AO-7 satellite*

```
1 07530U 74089B 04140.70617484 -.00000029 00000-0 10000-3 0 2774 2  
07530 101.6834 187.8825 0012044 277.9198 82.0507 12.53568957350341
```

- Keplerian elements are the inputs to a standard mathematical model of spacecraft orbits. With the "keps", the correct time, and the ground station location, we can compute when the satellite will be in view and where to point the antennae. This is the main technique used by tracking programs to predict where a satellite is at a given time.
- Need to be updated periodically
- Most tracking programs do this over the internet
- Two formats –
  - NORAD Two Line Elements (TLE – most common)
  - AMSAT Verbose Format

## Appendix C

### 1) Operational Satellites and frequencies as of 12/17/04

#### AMSAT-OSCAR 51 (Echo or AO-51)

Analog Uplink:	145.920 MHz FM (PL - 67Hz) 1268.700 MHz FM (PL - 67Hz)
Analog Downlink:	435.300 MHz FM 2401.200 MHz FM
PSK-31 Uplink	28.140 MHz USB
Digital Uplink:	145.860 MHz 9600 bps, AX.25 1268.700 MHz 9600 bps AX.25
Digital Downlink:	435.150 MHz 9600 bps, AX.25 2401.200 MHz 38,400 bps, AX.25
Broadcast Callsign:	PACB-11
BBS Callsign:	PACB-12
Launched	June 29, 2004

#### Status: **Operational**

AMSAT-OSCAR 51 or Echo as it is more commonly known is a FM satellite carrying 4 VHF receivers, 2 UHF transmitters, a multimode receiver and a 2400MHz transmitter. It can handle voice and FSK data up to 76.8Kbps. Echo was launched into a low, sun-synchronous polar orbit approximately 850 km high. You must transmit a 67Hz PL tone in order to access the Echo voice repeater.

*Please note the change in operational phone downlink frequency to 435.300 MHz.*

#### Fuji OSCAR 29 (FO-29)

Analog Uplink	146.000 to 145.900 MHz CW/LSB
Analog downlink	435.800 to 435.900 MHz CW/USB
Beacon	435.795 MHz (normally CW telemetry)
Digital Uplink	145.850, 145.870, 145.910 MHz FM
Digital Downlink	435.910 MHz 1200 baud BPSK or 9600 baud FSK
Digitalker	435.910 MHz FM
Launched	17 August 1996

#### Status: **Operational**

Despite a failure in June 2003 probably due to solar flares, FO-29 continues strong operation and has become very popular in the wake of the failure of UO-14. The analog mode is operational.

### **AMRAD OSCAR 27 (AO-27)**

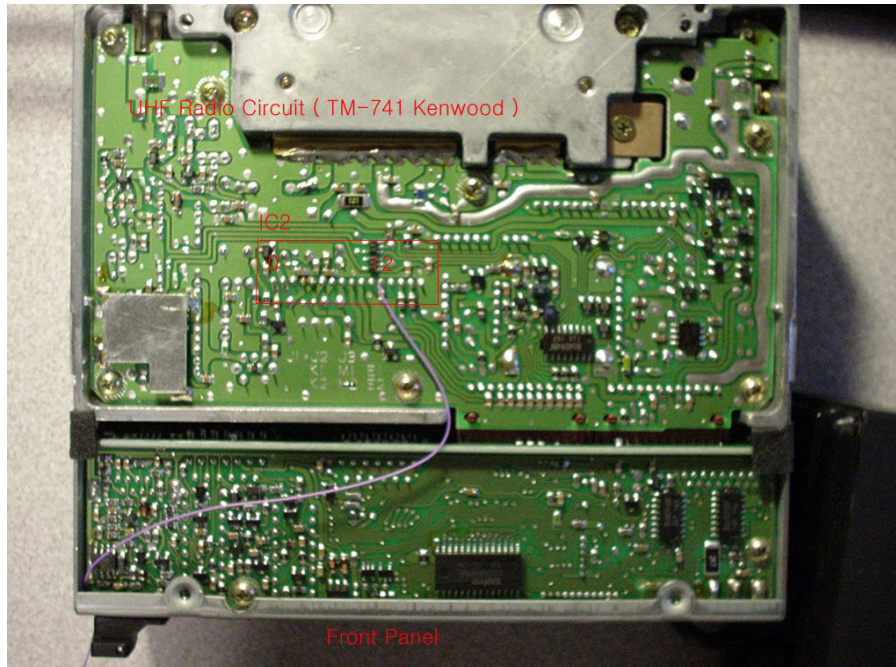
Uplink        145.850 MHz FM  
Downlink     436.795 MHz FM  
Launched     26 September 1993

#### **Status: Operational**

The AO-27 Team has reported that a new schedule has been uploaded. If the batteries hold then it should continue to run. If the batteries get too low then the software will turn the schedule off and the control operators will have to compute another one. The current schedule indicates that AO-27 is turned on for analogue operations during a 6 minute pass on the ascending pass. As in the past, the analogue mode is preceded and followed by 1 minute of digital TLM. A TLM stream is also transmitted for one minute on the descending pass.

## Appendix D

1) A wire is soldered at the point before the signal is being filtered (Other end is connected to the modem input)



2) Parts used in the LNY preamp.

Ref #	Value (marking)
C3, C4	.001 $\mu$ F chip cap
C7, C8	.001 $\mu$ F chip cap
D1	1N4148 diode
J1, J2	RCA jack (rf type)
Q1	Philips BF-998 MES FET
R1	620 $\Omega$ chip resistor
R2, R3	86K $\Omega$ chip resistor
R4	47 $\Omega$ chip resistor
U1	78L08ACD voltage regulator
C1, C2, C5, C6	4.5pF variable
L1, L2	2T #20 bus 6-19 screw form
L3	0.33 $\mu$ H choke



## Appendix F

### 1) Assembly Code for serial communication and relay control

\*\*\*\*\*

\* RS 232 at 1200,8,N,1,Xon/Off  
\* Tested with COM1 HyperTerminal Oct 6,2004  
\*\*\*\*\*

\*Constants cont

\*\*\*\*\*

BAUD: EQU \$102B  
SCCR1: EQU \$102C  
SCCR2: EQU \$102D  
SCSR: EQU \$102E  
SCDR: EQU \$102F

PB4: EQU 16 ;BIT 4 OF PORT B  
PB3: EQU 8 ;BIT 3 OF PORT B  
PB2: EQU 4 ;BIT 2 OF PORT B  
PB1: EQU 2 ;BIT 1 OF PORT B  
PB0: EQU 1 ;BIT 0 OF PORT B

CARRIAGE\_RET EQU \$0D  
LINE\_FEED EQU \$0A

STACK: equ \$FF

\*\*\*\*\*

\* Subroutines

\*\*\*\*\*

\*-----

delay\_1s:

```

    pshx
    ldx 50000 ; 50000 x 16 = 200,000 cycles = 1s
del1s: dex ; 3 cycles
    nop ; 2 cycle
    nop ; 2 cycle
    nop ; 2 cycle
    nop ; 2 cycle
    nop ; 2 cycle
    bne del1s ; 3 cycles
    pulx
    rts

```

\*-----

delay\_10ms:

```

    pshx
    ldx 2500 ; 2500 x 8 = 20,000 cycles = 10ms
del10ms: dex ; 3 cycles
    nop ; 2 cycle
    bne del10ms ; 3 cycles
    pulx

```

```

        rts
*-----
delay_1ms:
    pshx
    ldx 250          ; 250 x 8 = 2,000 cycles = 1ms
del1ms: dex          ; 3 cycles
        nop          ; 2 cycle
        bne del1ms   ; 3 cycles
        pulx
        rts
*-----
init    ldaa #$33 ;1200 baud
        staa BAUD
        ldaa #$00 ;mode
        staa SCCR1
        ldaa #$0C ;tie=rie=0,
        staa SCCR2 ;te=re=1
        rts
*-----
InChar  ldaa SCSR ;status
        bita #$20 ;rdrf?
        beq InChar
        ldaa SCDR ;SCI data
        rts
*-----
OutChar ldab SCSR ;status
        bitb #$80 ;tdre?
        beq OutChar
        staa SCDR ;output
        rts
*-----
; Output a string to the SCI
; Inputs: Reg X points to string
; String ends with 0
; Outputs: none
OutString ldaa 0,X
        beq OSdone ;0 at end
        bsr OutChar
        inx
        bra OutString
OSdone rts
*-----
moveHorRight:
    pshx
    LDX  #$1000 ;XREGISTER POINTS TO THE REGISTER BLOCK
    BSET  4,X PB0 ;TURN ON THE PB0 LED = 1
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    ;JSR  delay_10ms
    ;JSR  delay_1ms
    BCLR  4,X PB0 ;TURN OFF PB0 LED = 0
    pulx
    rts
*-----
```

```
moveHorLeft:
    pshx
    LDX  #$1000 ;XREGISTER POINTS TO THE REGISTER BLOCK
    BSET 4,X PB1 ;TURN ON THE PB1 LED = 1
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    ;JSR  delay_1ms
    ;JSR  delay_10ms
    BCLR 4,X PB1 ;TURN OFF PB1 LED = 0
    pulx
    rts
*-----

moveUp:
    pshx
    LDX  #$1000 ;XREGISTER POINTS TO THE REGISTER BLOCK
    BSET 4,X PB2 ;TURN ON THE PB2 LED = 1
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    BCLR 4,X PB2 ;TURN OFF PB2 LED = 0
    pulx
    rts
*-----

moveDown:
    pshx
    LDX  #$1000 ;XREGISTER POINTS TO THE REGISTER BLOCK
    BSET 4,X PB3 ;TURN ON THE PB3 LED = 1
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    JSR  delay_10ms
    BCLR 4,X PB4 ;TURN OFF PB3 LED = 0
    pulx
    rts
*-----

    org  $F000

start:  jsr    delay_10ms
        jsr    delay_10ms                ; delay 20ms during power on

        jsr    init
        ldx    #HELLO                    ;CONTROLLER IS READY
        jsr    OutString

loop:   jsr    InChar ;Next input
        jsr    OutChar ;Echo
        cmpa  #A
```

```

        beq acknowledge
        cmpa #R
        beq horRight
        cmpa #L
        beq horLeft
        cmpa #U
        beq verUp
        cmpa #D
        beq verDown

        jmp loop

acknowledge:
        pshx
        ldx #READY           ; READY for PORTD/RS232, x points to READY
        jsr OutString
        pulx
        jmp loop

horRight:
        pshx
        jsr moveHorRight
        jsr OutChar         ;echo after COMPLETE
        pulx
        jmp loop

horLeft:
        pshx
        jsr moveHorLeft
        jsr OutChar         ;echo after COMPLETE
        pulx
        jmp loop

verUp:
        pshx
        jsr moveUp
        jsr OutChar         ;echo after COMPLETE
        pulx
        jmp loop

verDown:
        pshx
        jsr moveDown
        jsr OutChar         ;echo after COMPLETE
        pulx
        jmp loop

READY: FCC "CONNECTED TO CONTROLLER"
MSG1: FCC "CONTROLLER READY"
MSG2: FCC "READING SIGNAL.."
HELLO: FCC "CONTROLLER INITIALIZED"
MSG3: FCC "READING SIGNAL"

        End

```

**2) Distance Bearing Calculation Code to be used with the Auto Tracking Client**

```
// Subject: DISTBEAR // Date: Mar.05,2004
// Modifier: Junsang Cho (chojs@cs.utah.edu)
// Note: This code is converted from the codes that were made in Basic
language.
//
// DISTBEAR provides distance and bearing information for two stations,
P1
// and P2, on the surface of the Earth. With small modifications it can
provide
// azimuth and elevation antenna aiming information for a ground
station at P1
// and a subsatellite point at P2.
#include <iostream>
#include <string>
#include <math.h>
using namespace std;
float fnarccos( float z );
int sgn( float a ); //Check the sign of a given value

int main(){

    string sLocation1 = "Baltimore";
    string sLocation2 = "Moscow";
    float fLatitude1 = 39.3; //'deg. E**
    float fLongitude1 = -76.6;
    float fLatitude2 = 56.1; //'deg. E**
    float fLongitude2 = 37.5;
    float PI = 3.141593;
    float fRadians2Deg = 180.0/PI;
    float fDeg2Radians = PI/180.0;
    float fKm2mile = 0.6214;
    float fMeanEarthRadius = 6371.0; //killometer
    float fLat1 = fLatitude1 * fDeg2Radians;
    float fLng1 = fLongitude1 * fDeg2Radians;
    float fLat2 = fLatitude2 * fDeg2Radians;
    float fLng2 = fLongitude2 * fDeg2Radians;
    float dsp,dlp;

    cout << "Starting... Program DISTBEAR ..." << " from " <<
sLocation1 << " to " << sLocation2 << endl;

    if ( abs( fLatitude1 ) > 89.99 ) { // P1 is North or South Pole
        cout << "If P1 is North or South pole then azimuth is not
defined" << endl;
        cout << "Point antenna along P2 longitude (short path) or"
<< endl;
        cout << "P2 longitude + 180 degree (long path)." << endl;
        if( sgn( fLat1 ) == sgn( fLat2 ) )
            dsp = fMeanEarthRadius*( PI/2 - abs( fLat2 ) );

        if( sgn( fLat1 ) != sgn( fLat2 ) )
            dsp = fMeanEarthRadius*( PI/2 + abs( fLat2 ) );

        cout << "Short path: " << dsp << "km" << endl;
    }
```

```
        cout << "Long path: " << int( 2*PI*fMeanEarthRadius - dsp )
<< "km" << endl;
        exit(0);
    }

    float fCosBeta = sin(fLat1) * sin(fLat2) * cos(fLat1) *
cos(fLat2) * cos(fLng2-fLng1);

    if( fCosBeta > 0.99999 ) {
        cout << "points coincide." << endl;
        exit(0);
    }

    if ( fCosBeta < -0.999999 ) { //Antipodes
        cout << "Antipodes, azimuth not defined, distance = " <<
fMeanEarthRadius * PI << " km" << endl;
        exit(0);
    }

    float fBeta = fnarccos( fCosBeta );
    float fDistanceShortPath = fBeta * fMeanEarthRadius;
//Distance
    short path float fDistanceLongPath = 2*PI*fMeanEarthRadius
- fDistanceShortPath; //Distance
    long path float fCosAZ = (sin(fLat2)-
sin(fLat1)*cos(fBeta))/(cos(fLat1)*sin(fBeta));
    float AZ = 0;
    if(fCosAZ > 0.999999) {
        AZ = 0;
    } else if ( fCosAZ < -0.999999 ) {
        AZ = 180; 3
    } else {
        AZ = fnarccos( fCosAZ ) * fRadians2Deg;
    }

    float AZSP = 0;
    float AZLP = 0;

    if ( sin( fLng2-fLng1 ) >= 0 ) {
        AZSP = AZ; AZLP = 180 + AZ;
    }

    if( sin( fLng2-fLng1 ) < 0 ) {
        AZSP = 360 - AZ; AZLP = 180 - AZ;
    }

    AZLP = int(AZLP*10 + 0.5)/10; //round off
    AZSP = int(AZSP*10 + 0.5)/10; //round off
    dsp = int(fDistanceShortPath + 0.5);
    dlp = int(fDistanceLongPath + 0.5);
    int dspmi = int(fDistanceShortPath * fKm2mile + 0.5 ); //
km -> miles
    int dlpmi = int(fDistanceLongPath* fKm2mile + 0.5);
    cout << "Short Path: " << dsp << "km (" << dspmi << "mi.) "
<< AZSP << " Deg. E of N" << endl;
    cout << "Long Path: " << dlp << "km (" << dlpmi << "mi.) "
<< AZLP << " Deg. E of N" << endl;
```

```
        return 0;
    }

float fnarccos( float z ){
    float PI = 3.141593;
    return (PI/2 - atan( z/sqrt(1-z*z)));
}

//Sign checker
int sgn( float a ){
    if( a > 0 ) return 1;
    if( a == 0 ) return 0;
    if( a < 0 ) return -1;
} //END
```

---

### 3) Satellite TNC | Terminal Client Code (Figure 13)

`Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MenuItem2.Click`

```
    Dim options As New Options()

    options.ShowDialog()
    If options.Go Then
        sbr.Panels(0).Text = options.ComboBox1.Text
        sbr.Panels(1).Text = options.ComboBox2.Text & " - " &
options.ComboBox3.Text & " - " &
        & options.ComboBox4.Text & " - " & options.ComboBox5.Text
        _port = CInt(Mid(options.ComboBox1.Text, 4))
        _baud = CType(CInt(options.ComboBox2.SelectedItem),
RS232.BaudRates)
        _data = CType(CInt(options.ComboBox3.SelectedItem),
RS232.DataSizes)
        Select Case options.ComboBox4.SelectedIndex
            Case 0
                _parity = RS232.Parities.Even
            Case 1
                _parity = RS232.Parities.Odd
            Case 2
                _parity = RS232.Parities.None
            Case Else
                _parity = RS232.Parities.Mark
        End Select
        Select Case options.ComboBox5.SelectedIndex
            Case 0
                _stopBit = RS232.StopBits.Bit1
            Case Else
                _stopBit = RS232.StopBits.Bit2
        End Select

        _rs232.CommPort = _port
        _rs232.BaudRate = _baud
        _rs232.DataSize = _data
        _rs232.Parity = _parity
        _rs232.StopBit = _stopBit
```

```
' or
' _rs232.Settings = "9600, n, 8, 1"
_rs232.InBufferSize = 4096
_rs232.InputLength = 0 ' read everything in the buffer.
_rs232.ReceiveThreshold = 1
_rs232.PortOpen = True

MenuItem2.Enabled = False
MenuItem3.Enabled = True
ToolBar1.Buttons(0).Enabled = False
ToolBar1.Buttons(1).Enabled = True
TextBox1.Enabled = True
TextBox2.Enabled = True
Button1.Enabled = True

End If

options.Dispose()

End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    _rs232 = New RS232()

    Dim i As Integer

    For i = 0 To 3
        General.Comm(i) = IsPortAvailable(i + 1)
    Next

End Sub

' This function attempts to open the passed Comm Port. If it is
' available, it returns True, else it returns False. To determine
' availability a Try-Catch block is used.
Private Function IsPortAvailable(ByVal port As Integer) As Boolean
    Try
        _rs232.CommPort = port
        _rs232.PortOpen = True
        _rs232.PortOpen = False
        Return True
    Catch
        ' If it gets here, then the attempt to open the Comm Port
        ' was unsuccessful.
        Return False
    End Try
End Function

Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object, ByVal e
As System.Windows.Forms.ToolBarButtonClickEventArgs) Handles
ToolBar1.ButtonClick
    Select Case ToolBar1.Buttons.IndexOf(e.Button)
        Case 0
            MenuItem2_Click(MenuItem2, New System.EventArgs())
        Case 1
```



```
        MenuItem3_Click(MenuItem3, New System.EventArgs())
    Case Else
        MenuItem5_Click(MenuItem5, New System.EventArgs())
    End Select
End Sub

Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem3.Click
    _rs232.PortOpen = False
    MenuItem2.Enabled = True
    MenuItem3.Enabled = False
    ToolBar1.Buttons(0).Enabled = True
    ToolBar1.Buttons(1).Enabled = False
    TextBox1.Enabled = False
    TextBox2.Enabled = False
    Button1.Enabled = False
End Sub

Private Sub Form1_Closing(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
    If MenuItem3.Enabled Then MenuItem3_Click(MenuItem3, New
System.EventArgs())
End Sub

Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem5.Click
    Me.Close()
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

    Try
        ' Enable the timer.
        ' Write an user specified Command to the Port.
        If CheckBox1.Checked Then
            _rs232.Output(Encoding.ASCII.GetBytes(Me.TextBox1.Text &
Chr(13)))
        Else
            _rs232.Output(Encoding.ASCII.GetBytes(Me.TextBox1.Text))
        End If
        ' unremark to output what you just sent.
        ' WriteMessage(TextBox1.Text, True)

    Catch ex As Exception
        ' Warn the user.
        MessageBox.Show("Unable to write to comm port")
    Finally
        TextBox1.Text = ""
        TextBox1.Focus()
    End Try

End Sub

' This subroutine writes a message to the txtStatus TextBox.
Private Sub WriteMessage(ByVal message As String)
    Me.TextBox2.Text &= message & vbCrLf
```

```
        TextBox2.SelectionStart = TextBox2.Text.Length
    End Sub

    ' This subroutine writes a message to the txtStatus TextBox and allows
    ' the Carriage Return to be suppressed.
    Private Sub WriteMessage(ByVal message As String, ByVal linefeed As
    Boolean)
        Me.TextBox2.Text &= message
        If linefeed Then
            Me.TextBox2.Text &= vbCrLf
        End If
        TextBox2.SelectionStart = TextBox2.Text.Length
    End Sub

    Private Sub oCP_OnComm(ByVal sender As Object, ByVal e As
    DBComm.CommEventArgs) Handles _rs232.OnComm

        Me.Text = e.Event.ToString & " - " & Now

        If _rs232.InBufferCount > 0 Then
            Do
                WriteMessage(_rs232.Input, False)
                If _rs232.InBufferCount = 0 Then
                    Exit Do
                End If
            Loop
        End If

    End Sub

    Private Sub TextBox1_TextChanged(ByVal sender As System.Object, ByVal e
    As System.EventArgs) Handles TextBox1.TextChanged

    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button2.Click
        TextBox2.Clear()
    End Sub
End Class
```

#### 4) Space Agent Client Code (Figure 11)

```
' enable send tcp data to server
Private Sub disTCP_Click()
    Timer1.Enabled = False
    enTCP.Visible = True
    disTCP.Visible = False
End Sub

' disable tcp data to server
Private Sub enTCP_Click()
    Timer1.Enabled = True
    enTCP.Visible = False
    disTCP.Visible = True
End Sub

'begin form load.  Initialized data here.
Private Sub Form_Load()
    StopNDDE.Visible = False
    disTCP.Visible = False
    Me.Caption = "Nova DDE Stopped"
    Timer1.Enabled = False

    tcpSERVER.Protocol = sckTCPProtocol
    tcpSERVER.LocalPort = 4711
    tcpSERVER.Listen()

    Text1.Text = "Not running..."
    Text3.Text = "Not running..."
    Text2.Text = tcpSERVER.LocalIP
    Text4.Text = tcpSERVER.LocalPort
End Sub

'exit client
Private Sub ExitDDE_Click()
    Unload(Me)
End Sub

'send data to server function
Private Sub sendnova()
    tcpSERVER.SendData(Text1.Text)
    Text3.Text = Text1.Text
End Sub

'connect to NOVA database
Private Sub StartNDDE_Click()
    Dim Ret As Integer
    On Error GoTo DDE_Nova_Error

    Me.Caption = "Nova DDE Started"
    StartNDDE.Visible = False
    StopNDDE.Visible = True

    Text1.LinkMode = 0
    Text1.LinkTopic = "NFW32|NFW_DATA"
    Text1.LinkItem = "NFW_SERVER"
```

```
Text1.LinkMode = 1
Text1.Text = "TUNE ON"
Text1.LinkPoke()
Exit Sub

DDE_Nova_Error:
    Ret = MsgBox("Error Opening the Nova DDE interface, Error: " &
Err.Description, vbCritical, "Nova DDE Error")
    Err.Clear()
End Sub

'disconnect from NOVA database
Private Sub StopNDDE_Click()
    Dim Ret As Integer
    On Error GoTo DDE_Nova_Error
    Me.Caption = "Nova DDE Stopped"
    StartNDDE.Visible = True
    StopNDDE.Visible = False
    Text1.LinkMode = 0
    Text1.LinkTopic = ""
    Text1.LinkItem = ""
    Text1.LinkMode = 0
    Text1.Text = "TUNE OFF"
    Timer1.Enabled = False
Exit Sub

DDE_Nova_Error:
    Ret = MsgBox("I can't be stopped!! Muhahaha! Error:" &
Err.Description, vbCritical)
    Err.Clear()
End Sub

Private Sub Timer1_Timer()
    Call sendnova()
End Sub
```

---

**5) Space Agent Server Code (Figure 12)**

```
Public Class Form1
    Inherits System.Windows.Forms.Form
    Dim WithEvents wsTCP As New OSWINSCK.Winsock
    Private comPORT As New RS232
    Dim COM1_2 As Integer
    Dim COMspeed As Integer

    Dim ISTRacking As Boolean
    Dim curAZ As String
    Dim curEL As String
    Dim curSAT As String

    Dim startAZ As Integer
    Dim startEL As Integer

    Dim counTer1 As Integer
    Dim counTer2 As Integer
    Dim counTer3 As Integer
    Dim counTer4 As Integer

    Dim cordLeft As Integer
    Dim cordRight As Integer
    Dim cordUp As Integer
    Dim cordDown As Integer

    Dim leftLIMIT As Integer
    Dim rightLIMIT As Integer
    Dim upLIMIT As Integer
    Dim downLIMIT As Integer

    Dim isCENTER As Boolean

    Dim defaultWIDTH As Integer
    Dim defaultHEIGHT As Integer

    Dim xdir As Integer
    Dim ydir As Integer

    Dim isdemo As Boolean

    #####Windows forms codes omitted#####

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        chkCON.Enabled = True
        chkCON.Interval = 100
        TextBox3.Text = ""

        defaultWIDTH = 310
        defaultHEIGHT = 320
        Me.Width = defaultWIDTH
        Me.Height = defaultHEIGHT
    End Sub
End Class
```

```
    ITracking = False
    COM1_2 = 1
    COMspeed = 100

    cordLeft = 0
    cordRight = 0
    cordUp = 0
    cordDown = 0
    xdir = 0
    ydir = 0

    TextBox1.Text = "155.98.70.190"

    TextBox5.Text = 215
    TextBox6.Text = 105
    TextBox7.Text = 80
    TextBox8.Text = 0

    leftLIMIT = TextBox5.Text
    rightLIMIT = TextBox6.Text
    upLIMIT = TextBox7.Text
    downLIMIT = TextBox8.Text

    isCENTER = True

    counTer1 = 0
    counTer2 = 0
    counTer3 = 0
    counTer4 = 0

    startAZ = 0
    startEL = 0

    isdemo = False

    'Label8.Text = "(U)" & cordUp & " (R)" & cordRight & " (D)" &
cordDown & " (L)" & cordLeft
    'Label8.Refresh()

    If IsPortAvailable(SEL_COMPORT.SelectedIndex + 1) Then
        COM1_2 = SEL_COMPORT.SelectedIndex + 1
        TextBox3.Text = "Program loaded successfully." & vbCrLf &
"Program settings: " & vbCrLf & "Serial Port: COM" & COM1_2 & vbCrLf &
—
        "COM Delay: " & COMspeed & "ms" & vbCrLf &
"Auto Track: " & ITracking & vbCrLf & "Track Server: " & _
        TextBox1.Text & vbCrLf & "Track Server TCP
Port: " & TextBox2.Text
        enDIRbutton()
    Else
        TextBox3.Text = "Program loaded successfully." & vbCrLf &
"Program settings: " & vbCrLf & "Serial Port: COM" & COM1_2 & " not
available" & vbCrLf & _
        "COM Delay: " & COMspeed & "ms" & vbCrLf &
"Auto Track: " & ITracking & vbCrLf & "Track Server: " & _
```

```
        TextBox1.Text & vbCrLf & "Track Server TCP
Port: " & TextBox2.Text

        disDIRbutton()
    End If

End Sub

'connect to server
Private Sub tcpCON_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tcpCON.Click
    Dim servIP As String
    Dim servPORT As String
    servIP = TextBox1.Text
    servPORT = TextBox2.Text
    wsTCP.Connect(servIP, servPORT) 'state = 7
    tcpDIS.Visible = True
    tcpCON.Visible = False
    TextBox3.Text = "Connected using: " & vbCrLf & "Server: " &
servIP & vbCrLf & "Port: " & servPORT
    TextBox3.Refresh()
End Sub

'disconnect from server
Private Sub tcpDIS_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles tcpDIS.Click
    wsTCP.CloseWinsock() 'state = 8
    tcpDIS.Visible = False
    tcpCON.Visible = True
    TextBox3.Text = "Track data has stopped with reason:
Disconnected from server."
    TextBox3.Refresh()
End Sub

'get incoming data
Private Sub wsTCP_OnDataArrival(ByVal bytesTotal As Integer)
Handles wsTCP.OnDataArrival
    Dim sBuffer As String
    wsTCP.GetData(sBuffer)
    getDATA.Text = sBuffer

    parser(sBuffer)

End Sub

'pars all incoming data from client
Private Sub parser(ByVal sBuffer As String)

    Dim dlist() As String
    Dim dlist2() As String

    Dim sa As String
    Dim az As Double
    Dim el As Double
```

```
Dim rr As Double
Dim ah As String

dlist = Split(sBuffer, " ")

dlist2 = Split(dlist(0), ":")

sa = dlist2(0)
dlist2 = Split(dlist(1), ":")
az = dlist2(1)
dlist2 = Split(dlist(2), ":")
el = dlist2(1)
dlist2 = Split(dlist(3), ":")
rr = dlist2(1)
dlist2 = Split(dlist(4), ":")
ah = dlist2(1).ToString

curSAT = sa
displayDATA(sa, az, el, rr, ah)
AutoTracker(az, el)

End Sub

Private Sub displayDATA(ByRef sa As String, ByRef az As Double,
ByRef el As Double, _
    ByRef rr As Double, ByRef ah As String)
Label3.Text = "Satellite: " & sa
Label4.Text = "AZ: " & CType(az, Integer)
Label5.Text = "EL: " & CType(el, Integer)
Label6.Text = "RR: " & rr
Label7.Text = "AH: " & ah

End Sub

Private Sub chkCON_Elapsed(ByVal sender As System.Object, ByVal e
As System.Timers.ElapsedEventArgs) Handles chkCON.Elapsed
    Me.Text = wsTCP.State
Label11.Text = Today & " " & TimeOfDay
    If (wsTCP.State = 7) Then
        Label8.Text = "Connected to " & TextBox1.Text & " on port "
& TextBox2.Text
    ElseIf (wsTCP.State = 6) Then
        Label8.Text = "Connecting to... " & TextBox1.Text & " on
port " & TextBox2.Text
    ElseIf (wsTCP.State = 9) Then
        tcpDIS.Visible = False
        tcpCON.Visible = True
        Label8.Text = "Failed to connect with " & TextBox1.Text & "
on port " & TextBox2.Text
    Else
        Label8.Text = "Not Connected."
    End If
End Sub
```



```
Private Function IsPortAvailable(ByVal ComPortn As Integer) As Boolean
    Try
        comPORT.Open(ComPortn, 115200, 8,
RS232.DataParity.Parity_None, _
        RS232.DataStopBit.StopBit_1, 4096)
        ' If it makes it to here, then the Comm Port is available.
        comPORT.Close()
        Return True
    Catch
        ' If it gets here, then the attempt to open the Comm Port
        ' was unsuccessful.
        Return False
    End Try
End Function

'##### Manual Directional Movements #####
Private Sub dirUP_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles dirUP.Click
    xyMove("u")
    System.Threading.Thread.Sleep(COMspeed)

    If ((xdir = 0) And (ydir = 0)) Then
        isCENTER = True
    Else
        isCENTER = False
    End If
End Sub

Private Sub dirDOWN_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles dirDOWN.Click
    xyMove("d")
    System.Threading.Thread.Sleep(COMspeed)
    If ((xdir = 0) And (ydir = 0)) Then
        isCENTER = True
    Else
        isCENTER = False
    End If
End Sub

Private Sub dirLEFT_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles dirLEFT.Click
    xyMove("l")
    System.Threading.Thread.Sleep(COMspeed)
    If ((xdir = 0) And (ydir = 0)) Then
        isCENTER = True
    Else
        isCENTER = False
    End If
End Sub

Private Sub dirRIGHT_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles dirRIGHT.Click
    xyMove("r")
    System.Threading.Thread.Sleep(COMspeed)
    If ((xdir = 0) And (ydir = 0)) Then
        isCENTER = True
    End If
End Sub
```

```
        Else
            isCENTER = False
        End If
    End Sub
'##### Manual Directional Movements #####

'Reset coordinates to zero. This will allow user to re center the
antenna to any given origin.
Private Sub ResetCord_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ResetCord.Click
    cordLeft = 0
    cordRight = 0
    cordUp = 0
    cordDown = 0
    startAZ = 0
    startEL = 0
    xdir = 0
    ydir = 0

    TextBox3.Text = "Resetting cordinates : " & vbCrLf & "X Axis: "
& xdir & vbCrLf & "Y Axis: " & ydir & vbCrLf & "Please use manual
control to to re-align antenna array to point NORTH."
    TextBox3.Refresh()
End Sub

'recenter the antenna array
Private Sub dirCENTER_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles dirCENTER.Click
    CENTER_ME()
End Sub

Private Sub CENTER_ME()
    disDIRbutton()
    Me.Text = "Please wait..."
    While (xdir > 0)
        TextBox3.Text = "Current cordinate : " & vbCrLf & "X Axis:
" & xdir & vbCrLf & "Y Axis: " & ydir
        TextBox3.Refresh()
        System.Threading.Thread.Sleep(COMspeed)
        xyMove("l")

    End While
    While (xdir < 0)
        TextBox3.Text = "Current cordinate : " & vbCrLf & "X Axis:
" & xdir & vbCrLf & "Y Axis: " & ydir
        TextBox3.Refresh()
        System.Threading.Thread.Sleep(COMspeed)
        xyMove("r")
    End While

    While (ydir > 0)
        System.Threading.Thread.Sleep(COMspeed)
        xyMove("d")

    End While
```

```
While (ydir < 0)
    System.Threading.Thread.Sleep (COMspeed)
    xyMove ("u")

End While

If ((xdir = 0) And (ydir = 0)) Then
    isCENTER = True
    TextBox3.Text = "Center at x= " & xdir & " y= " & ydir
Else
    isCENTER = False
End If

enDIRbutton()
Me.Text = "Space Agent"
End Sub

'move to specify amount
Private Sub xyMOVETO (ByVal amount As Integer, ByRef direct As
String)
    Dim i As Integer
    i = 0
    Select Case direct
        Case "l"
            If (amount >= leftLIMIT) Then
                'CENTER_ME ()
                TextBox3.Text = "Can not move that far. Please
consult software manual on program re-alignment."
            Else
                For i = 0 To amount
                    System.Threading.Thread.Sleep (COMspeed)
                    xyMove (direct)

                Next
            End If

        Case "r"
            If (amount >= 360 - rightLIMIT) Then
                'CENTER_ME ()
                TextBox3.Text = "Can not move that far. Please
consult software manual on program re-alignment."
            Else
                For i = 0 To amount
                    System.Threading.Thread.Sleep (COMspeed)
                    xyMove (direct)

                Next
            End If

        Case "u"
            If (amount >= upLIMIT) Then
                'CENTER_ME ()
                TextBox3.Text = "Can not move that far. Please
consult software manual on program re-alignment."
            Else
                For i = 0 To amount
                    System.Threading.Thread.Sleep (COMspeed)
```

```
        xyMove(direct)
    Next
End If

Case "d"
    If (amount >= downLIMIT) Then
        'CENTER_ME()
        TextBox3.Text = "Can not move that far. Please
consult software manual on program re-alignment."
    Else
        For i = 0 To amount
            System.Threading.Thread.Sleep(COMspeed)
            xyMove(direct)
        Next
    End If
    System.Threading.Thread.Sleep(COMspeed)
End Select
End Sub

'move according to "direct" param
Private Sub xyMove(ByVal direct As String)
    Me.Refresh()
    TextBox3.Refresh()
    Select Case direct
        Case "l" 'move Left
            If (xdir >= leftLIMIT) Then
                'CENTER_ME()
                TextBox3.Text = "Can not move any further. Please
consult software manual on program re-alignment."
                Return
            Else
                TextBox3.Text() = "xyMove() Moving LEFT: " & vbCrLf
& "X Axis: " & xdir & vbCrLf & "Y Axis: " & ydir
                TextBox3.Refresh()
                comPORT.Open(COM1_2, 1200, 8,
RS232.DataParity.Parity_None, RS232.DataStopBit.StopBit_1, 512)
                comPORT.Write(2)
                comPORT.ClearInputBuffer()
                comPORT.Close()
                xdir -= 1
            End If

        Case "r" 'move right
            If (xdir >= 360 - rightLIMIT) Then
                'CENTER_ME()
                TextBox3.Text = "Can not move any further. Please
consult software manual on program re-alignment."
                Return
            Else
                TextBox3.Text() = "xyMove() Moving RIGHT: " &
vbCrLf & "X Axis: " & xdir & vbCrLf & "Y Axis: " & ydir
                TextBox3.Refresh()
                comPORT.Open(COM1_2, 1200, 8,
RS232.DataParity.Parity_None, RS232.DataStopBit.StopBit_1, 512)
                comPORT.Write(1)
                comPORT.ClearInputBuffer()
                comPORT.Close()
            End If
        End Select
    End Sub
```

```
        xdir += 1
    End If

    Case "u" 'move up
        If (ydir >= upLIMIT) Then
            'CENTER_ME()
            TextBox3.Text = "Can not move any further. Please
consult software manual on program re-alignment."
            Return
        Else

            TextBox3.Text() = "xyMove() Moving UP: " & vbCrLf &
"X Axis: " & xdir & vbCrLf & "Y Axis: " & ydir
            TextBox3.Refresh()

            comPORT.Open(COM1_2, 1200, 8,
RS232.DataParity.Parity_None, RS232.DataStopBit.StopBit_1, 512)
            comPORT.Write(3)
            comPORT.ClearInputBuffer()
            comPORT.Close()
            ydir += 1

        End If

    Case "d" 'move down
        If (ydir <= downLIMIT) Then
            'CENTER_ME()
            TextBox3.Text = "Can not move any further. Please
consult software manual on program re-alignment."
            Return
        Else

            TextBox3.Text() = "xyMove() Moving DOWN: " & vbCrLf
& "X Axis: " & xdir & vbCrLf & "Y Axis: " & ydir
            TextBox3.Refresh()

            comPORT.Open(COM1_2, 1200, 8,
RS232.DataParity.Parity_None, RS232.DataStopBit.StopBit_1, 512)
            comPORT.Write(4)
            comPORT.ClearInputBuffer()
            comPORT.Close()
            ydir -= 1

        End If
    End Select

End Sub

'auto tracker function. This is where auto tracking actually
happen
Private Sub AutoTracker(ByRef az As String, ByRef el As String)
    Dim temp_az As Integer
    Dim temp_el As Integer
    Dim azimuth As Integer

    temp_az = CType(az, Integer)
```

```
temp_el = CType(el, Integer)

If ITracking Then 'get to the initial tracking point

    '           System.Threading.Thread.Sleep(COMspeed)
    azimuth = temp_az 'calculate azimuth
    Dim elevation As Integer = Math.Floor(el)

    TextBox3.Refresh()
    If (temp_az > 180) Then
        azimuth = temp_az - 360
    End If

    'Azimuth tracking
    If (temp_az < leftLIMIT) And (temp_az > rightLIMIT) Then
'in the blocked zone
        TextBox3.Text = "Error: " & curSAT & " outside antenna
movement range."
        Return
    Else 'valid antenna zone
        'MsgBox("Autotracking")
        If (azimuth > 0) Then 'in the right half
            If (xdir < azimuth) Then
                xyMove("r")
            ElseIf (xdir > azimuth) Then
                xyMove("l")
            End If
        End If

        If (azimuth < 0) Then 'in the right half
            If (xdir < azimuth) Then
                xyMove("r")
            ElseIf (xdir > azimuth) Then
                xyMove("l")
            End If
        End If
    End If

    'Elevation Tracking
    System.Threading.Thread.Sleep(COMspeed)
    If (temp_el > upLIMIT) And (temp_el < downLIMIT) Then 'in
the blocked zone
        TextBox3.Text = "Warning: " & curSAT & " outside
antenna movement range (Below Horizon). "
        Return
    Else 'valid antenna zone
        'MsgBox("Autotracking Vertical")
        If (ydir < temp_el) Then
            'Dim i As Integer
            'i = temp_el
            'While (ydir < i)
            'TextBox3.Text = "ydir < elevation: " & ydir &
vbCrLf & "elevation: " & elevation & vbCrLf & "llimit: " & downLIMIT &
vbCrLf & " uplimit: " & upLIMIT
            ' xyMove("u")
            ' End While
        End If
    End If
End If
```

```
        xyMove("u")
    ElseIf (ydir > temp_el) Then
        'Dim i As Integer
        'i = temp_el
        'While (ydir < i)
        'TextBox3.Text = "ydir < elevation: " & ydir &
vbCrLf & "elevation: " & elevation & vbCrLf & "llimit: " & downLIMIT &
vbCrLf & " uplimit: " & upLIMIT
        'xyMove("u")
        'End While
        xyMove("d")
    End If

    End If
End If

End Sub

'Auto track button
Private Sub aTRACK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles aTRACK.Click
    If Me.Width <= defaultWIDTH Then
        'CENTER_ME() 'no need to center anymore because program
will automatically check for it
        Me.Width = 512
        ITracking = True
        aTRACK.Text = "Auto ON"
        aTRACK.BackColor = Color.Lime
        trackEvent.Enabled = True
        trackEvent.Interval = COMspeed
        TextBox3.Text = "Auto Tracking Enabled."

    Else
        'TextBox3.Text = "Auto Tracking Disabled." & vbCrLf &
"Centering antenna array. Please wait...."
        CENTER_ME()
        Me.Width = defaultWIDTH
        ITracking = False
        aTRACK.Text = "Auto Track"
        aTRACK.BackColor = Color.FromName("control")
        trackEvent.Enabled = False

    End If
End Sub

'set COM port selection
Private Sub SEL_COMPORTE_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
SEL_COMPORTE_SelectedIndexChanged
    TextBox3.Text = "Checking serial COM port availability. Please
wait...."
    TextBox3.Refresh()
    If IsPortAvailable(SEL_COMPORTE.SelectedIndex + 1) Then
        COM1_2 = SEL_COMPORTE.SelectedIndex + 1
        TextBox3.Text = "Serial port COM" & COM1_2 & " is available
and ready."
        enDIRbutton()
    End If
End Sub
```

```
Else
    TextBox3.Text = "Serial port COM" &
SEL_COMPORT.SelectedIndex + 1 & " is not available. Please check your
connection and try again."
    disDIRbutton()
End If

End Sub

'disable all manual direction button
'need this when antenna array is moving
Private Sub disDIRbutton()
    dirUP.Enabled = False
    dirDOWN.Enabled = False
    dirLEFT.Enabled = False
    dirRIGHT.Enabled = False
    dirCENTER.Enabled = False
    ResetCord.Enabled = False
    aTRACK.Enabled = False
    Button2.Enabled = False
    exitBUTTON.Enabled = False
End Sub

'enable all directional button
Private Sub enDIRbutton()
    dirUP.Enabled = True
    dirDOWN.Enabled = True
    dirLEFT.Enabled = True
    dirRIGHT.Enabled = True
    dirCENTER.Enabled = True
    ResetCord.Enabled = True
    aTRACK.Enabled = True
    Button2.Enabled = True
    exitBUTTON.Enabled = True
End Sub

'check for send speed to serial port
Private Sub sendSPEED_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
sendSPEED.SelectedIndexChanged
    COMspeed = sendSPEED.SelectedItem
    TextBox3.Text = "Using COM delay: " & COMspeed & "ms"
    TextBox3.Refresh()
End Sub

'exit program
Private Sub exitBUTTON_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles exitBUTTON.Click
    End
End Sub

'show configuration panel
Private Sub Button2_Click_1(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button2.Click

    If (Me.Height = 448) Then
        Me.Height = defaultHEIGHT
```



```
Else
    Me.Height = 448
End If
'TextBox3.Text = upLIMIT & " - " & downLIMIT & " - " &
rightLIMIT & " - " & leftLIMIT
End Sub

'set configuration parameters
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
    leftLIMIT = TextBox5.Text
    rightLIMIT = TextBox6.Text
    upLIMIT = TextBox7.Text
    downLIMIT = TextBox8.Text
    Me.Height = defaultHEIGHT
End Sub

'enable autotracker
Private Sub trackEvent_Tick(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles trackEvent.Tick
    AutoTracker(curAZ, curEL)
End Sub

'enable demo mode
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    If isdemo Then
        isdemo = False
    Else
        isdemo = True
    End If

    If isdemo Then

        TextBox4.Visible = True

        COMspeed = 80
        TextBox4.Text = "Moving UP 30 degrees.."
        xyMOVETO(20, "u")

        '        COMspeed = 10
        TextBox4.Text = "Moving LEFT 100 degrees.."
        xyMOVETO(100, "l")

        TextBox4.Text = "Moving RIGHT 40 degrees.."
        xyMOVETO(40, "r")

        '        COMspeed = 200
        TextBox4.Text = "Moving DOWN 5 degrees.."
        xyMOVETO(5, "d")

        '        COMspeed = 10
        TextBox4.Text = "Moving DIAG 15 degrees.."
        movediagnol(15, "l")

        TextBox4.Text = "Moving RIGHT 40 degrees.."
        xyMOVETO(40, "r")
```

```
        TextBox4.Text = "Moving DIAG 10 degrees.."
        movediagnol(10, "r")

        '      COMspeed = 200
        TextBox4.Text = "Moving DOWN 5 degrees.."
        xyMOVETO(5, "d")

        '      COMspeed = 10
        TextBox4.Text = "Moving DIA 15 degrees.."
        movediagnol(10, "l")

        TextBox4.Text = "DONE"
        COMspeed = 200
        TextBox4.Visible = False

    End If
End Sub

'move diagonal
Private Sub movediagnol(ByVal mystep As Integer, ByVal direct As
String)
    Dim i As Integer
    i = 0
    If (direct = "l") Then
        While (i < mystep)
            xyMOVETO(1, "l")
            xyMOVETO(1, "u")
            i += 1
        End While
    End If

    'send move diagonal direction
    If (direct = "r") Then
        While (i < mystep)
            xyMOVETO(1, "r")
            xyMOVETO(1, "u")
            i += 1
        End While
    End If
End Sub
End Class
```