

Embedded System Using GNU/Linux for Automating Low Earth Orbit Satellite Tracking

Omar Alvarez Cardenas, Miguel A. Garcia-Ruiz, Jesus Rafael Carrillo and Margarita Glenda Mayoral Baldivia

ABSTRACT

There are different types of artificial satellites orbiting the Earth; most of them are used for communications, astronomy, atmospheric studies, navigation, space exploration, and weather forecast, among other specific functions. All these satellites need to be located and tracked according to their orbit and altitude, but many commercial tracking systems are expensive and some of them are difficult to use and maintain. This chapter describes a low-cost and efficient embedded computer system running on GNU/Linux, intended for automating the orbit calculation and tracking of satellites, as well as performing the orientation of an antenna system to receive and transmit radio signals from Low Earth Orbit (LEO) satellites used for amateur radio communications. Our embedded system performs the same basic functions of any commercial satellite tracking system but with a low cost and running under the free software development philosophy, making the tracking process of Low Earth Orbit satellites accurate and easy to maintain.

Introduction

Amateur radio (also called ham radio) is a nonprofit, non-commercial and private recreation of exchanging messages via voice and data communication using designated segments of the radio frequency (RF) spectrum. Amateur radio enthusiasts generally use the RF spectrum to contact people from all over the world, carrying out wireless experimentation for self-training and for some types of emergency communications (ARRL 2010). Amateur

radio operation is regulated and coordinated globally by the International Telecommunication Union (ITU), a branch of the United Nations (UN) (ITU 2008), and licensed by national governments that regulate operational and technical characteristics of radio frequency transmissions and issue individual and group (club) stations with an identifying call sign (an unique identifier composed of letters and numbers).

In order to obtain a radio amateur license (and its respective call sign), future amateur radio operators are tested for their knowledge of key concepts in analog electronics, electricity, RF, and the host government's radio regulations. For example, the government body that regulates the amateur radio operations and licensing in the United States is the Federal Communications Commission (FCC), who defines amateur radio service as "a voluntary noncommercial communication service, used by qualified persons of any age who are interested in radio technique with a personal aim and without pecuniary interest" (FCC 2011). In some countries, future amateurs must also pass a Morse code test. Currently, an estimated two million people worldwide are licensed amateur radio operators. They usually consist of commercially-available transceivers, antennas and other parts of their amateur radio station, but operators can build them by themselves by building their own "homebrew" (homemade) equipment.

People involved in amateur radio communications use a variety of text, voice, and image communications modes and can communicate across a city, region, country, the entire and even to outer space with astronauts, since a number of them hold a radio amateur licenses and communicate to the Earth using their radio amateur equipment installed in spacecraft. There are also specially-built artificial satellites for radio amateur operators to practice communicating with.

Amateur radio operators can also use other resources to communicate regionally and even throughout an entire continent using outer space capacities. Some organisms have designed and built artificial satellites capable of this type of communications for decades. These satellites have been used for amateur radio communications using voice and data (mainly in the form of Morse code), using designated very-high and ultra-high frequencies (VHF and UHF), respectively (Davidoff 2009). Amateur satellites are generally used to communicate to two or more amateur radio operators by retransmitting an RF signal carrying voice or data. Amateur satellites may also transmit telemetry data that an amateur operator may pick up and analyze. In addition, some amateur satellites transmit a recorded greeting voice or data message using Morse code, which is transmitted continuously, working as electronic "beacons" for any amateur radio operator to pick up and decode. However, commercial satellite antennas and commercial satellite tracking systems for radio amateurs are expensive and difficult

to maintain. One way of overcoming this is to build homebrew satellite equipment for amateur radio operation.

This chapter describes the development of an embedded computer system running GNU/Linux which is responsible for automating the orbit calculation and orientation of an antenna used to transmit radio signals to radio amateur satellites and receive radio signals from them (the satellite tracking process). The advantages of our embedded system are: Ease of construction, low cost, and use of a homebrew antenna. Our satellite tracking system performs the same basic functions of any of the commercial systems for tracking satellites and orienting antennas, but under the free software philosophy. With the automation of the tracking process, the operation of low Earth orbit (LEO) satellites becomes easy and accurate.

Other types of artificial satellites that amateur operators may use (and be tracked using our above mentioned embedded computer system) are the weather satellites used for atmospheric analysis and forecast (Taggart 1996). Some of these weather satellites were built by the National Oceanic and Atmospheric Administration (NOAA) of the United States (NOAA 2011). They transmit real-time images of the surface of the Earth that can be down linked and decoded using special software. It is interesting to note that no amateur radio license is required to receive data from the weather satellites or for picking up beacons from amateur satellites.

A typical amateur radio station for satellite communications consists mainly of specialized antennas, pre-amplifiers, transceivers for the VHF, UHF and HF frequencies, a low loss feed line and satellite tracking software (Newport n/d). For best results, amateur radio operators use directional (Yagi) antennas. The use of a directional antenna is an important aspect and there is a great range of choice in commercial systems. Table 1 shows the most popular of these. For most amateur satellites, radio operators have to transmit and receive in different bands; one for the uplink (a radio signal transmission from the operator to the satellite) and a different one for the downlink (the reception of a radio signal from the satellite to the operator). An amateur radio operator can use a dual-band antenna or two separate antennas to carry out the uplink and the downlink. An operator also can use several antennas to operate on various uplink/downlink bands as needed (ITU 2008). One good option is to buy commercial satellite antennas from different brands. Table 1 describes some of them. However, most of them are expensive, especially the ones with high gain.

Table 1. Commercial Amateur Satellite Antennas (ARRL 2011).

<i>Model</i>	<i>Brand</i>	<i>Elements</i>	<i>Gain</i>
A-144S5	Diamond (VHF)	5	9.1dB
A-27010S	Cushcraft (VHF/UHF)	5/5	10/10 dB
2M/440L5	Elk (VHF/UHF)	5/5	8.9/9 dB
2MCP	M2 (VHF)	22	14.3 dB
436CP30	M2(UHF)	30	15.50 dB
146/437-10	Arrow Antenna (VHF/UHF)	3/7	0/0 dB

If the cost of the satellite antennas is a problem, there are some “homebrew” (homemade) low-cost antennas that can be used with great results (Britian 2009). They have been built, tested and used by many ham radio operators in different countries. The most popular homebrew antennas are:

- Turnstile
- Moxon
- Eggbeater
- TPA
- The quadrifilar helicoidal
- The EZ Linderblad
- XE1MEX Yagi/UDA

Having selected a homebrew antenna, a way to greatly improve the downlink signal is to install a receive preamplifier in the satellite antenna system. It consists of a high gain and low-noise amplifier with a frequency response for a specific band. A well-designed preamplifier can give and extra gain in the order of 15 to 25 dB with a minimum noise figure of 0.5 to 2 dB (ARRL 2007).

With respect to amateur radio transceivers for satellite operation, there are many commercial amateur radios that cover the VHF, UHF and HF bands and some of them offer satellite-specific features like the ICOM IC-2820, which receives and transmits on 2 m and 70 cm band independently. One of the transceivers with full HF coverage, plus VHF and UHF with full duplex satellite capability, is the Kenwood TS-2000. Another solution is using a hand-held (portable) VHF/UHF transceiver like the Yaesu FT-60R.

To complete an amateur radio satellite station, it is necessary to include a very low loss feed-line coaxial cable. All amateur satellites transmit using low power to extend their operation life in space, and the downlink signals must arrive readable at the amateur radio operator’s transceiver. Amateur radio transceivers are designed to work with 50-ohm coaxial cable. If the

transceiver finds a different impedance, it reduces its output to protect itself from damage that than can result from standing wave ratio (SWR) variations (ARRL 2007).

One important aspect of amateur radio satellite communications is the ability to track and predict a specific orbit related to the radio station position. To do this, it is possible to find satellite tracking software for the Windows, Macintosh and Linux operating systems (see Table 2). Most of them can automatically correct for the Doppler Effect and move the azimuth and elevation rotors of a directional antenna to get the best download radio signal. In general, most of the professional tracking software have the following functionalities:

- Professional features intended for serious satellite users
- Satellite tracking and prediction
- Antenna steering through popular interfaces
- Radio tuning with automatic Doppler correction
- Simulations and graphical interfaces
- Extensive documentation and vendor support (AMSAT 2011).

Table 2. Sampling of Satellite Tracking Software (AMSAT 2011; ARRL 2011).

<i>Name</i>	<i>Source</i>	<i>Operating System</i>	<i>Radio Control</i>	<i>Antenna Control</i>
Nova	www.amsat.org (store)	Windows	No	Yes
SCRAP	www.amsat.org (store)	Windows	No	Yes
SatPC32	www.amsat.org (store)	Windows	Yes	Yes
SatScape	www.satscape.co.uk	Windows	Yes	Yes
MacDoppler	www.dogparksoftware.com	Mac OS	Yes	Yes
Ham Radio Deluxe	hrd.ham-radio.ch	Windows	Yes	No
Predict	www.qsl.net/kd2bd/predict.html	Linux	No	Yes
WinOrbit	www.sat-net.com/winorbit	Windows	No	No

There are currently a number of Low Earth Orbit (LEO) satellites that can be used as digital/voice repeaters and receive weather images that amateur radio operators can use. In order to access them, an amateur radio operator needs to compute the orbit and the real-time position of an amateur radio satellite using one of many available computer programs for orbit calculation, adjust and point a special antenna towards the satellite, and use their radio communication equipment. Operators can manually set up their antennas, or use commercially-available systems to do it automatically.

However, these types of commercial systems are very expensive and difficult to maintain. To overcome the cost of commercial systems for satellite tracking, it is possible to build a homemade system running the Linux operating system (O.S.), which rivals expensive commercial equipment. The next pages describe the design and development of a homemade satellite tracking system that consists of an embedded computer system and state-of-the-art software, including a specially-adapted version of GNU/Linux O.S.

Our embedded system uses an adapted version of Slackware GNU/Linux, adapted to run from an USB Pen Drive. The core of the tracking system runs at the top of the O.S., which involves the pass scheduler, orbit calculation and the rotor control. Other system services such as SSH/Web also run on the Embedded OS, but with low priority.

The antenna orientation system uses a servo motor and a stepper motor to move the antenna. The servo motor is used to control the elevation of the antenna from zero to ninety degrees. The interface between the embedded system and the servo motor is the PicServo control board, which is an open-hardware design and uses a PIC Micro controller.

The stepper motor is used to move the antenna on the azimuth axis. The motor is connected to an adapted old dot matrix printer mechanism. This helps the stepper motor to gain the required strength to move the elevation mechanism and the antenna. The azimuth movement of the motor goes from zero to 360 degrees. This motor is controlled by a computer parallel port using a power interface that includes TIP120 transistors and opto-couplers (SHARP PC817) to prevent parallel port damage because the motor draws 1.5 Amperes.

The antenna system uses an antenna design proposed by two Amateur Radio Operators from Spain (with amateur radio call signs EB4DKA and EA4CYQ) which is popular among amateur satellite operators. This design is called The Canary Jail Umbrella (CJU) Antenna. This design has been discussed in a number of amateur radio magazines such as QST (www.arrl.org/qst), CQ Amateur Radio (www.cq-amateur-radio.com), AntenneX (www.antennex.com), CQ VHF (<http://www.cq-vhf.com>), and Onda Corta (Alvarez et al. 2010), among others. The CJU antenna has been praised worldwide because of its ease of construction and the good quality of signal reception. One of the key characteristics of its design is the weight of the antenna. Using light materials like Balsa wood or Bamboo in its construction can make it light enough to mount on a small servo motor.

Figure 1 shows the system and its components. To understand how it works, we need to first understand what a satellite is, what azimuth an elevation movements are, how the servo motor and stepper motor are controlled, how to modify a GNU/Linux distribution to run from an USB pen drive, and how satellite orbit calculations are done.

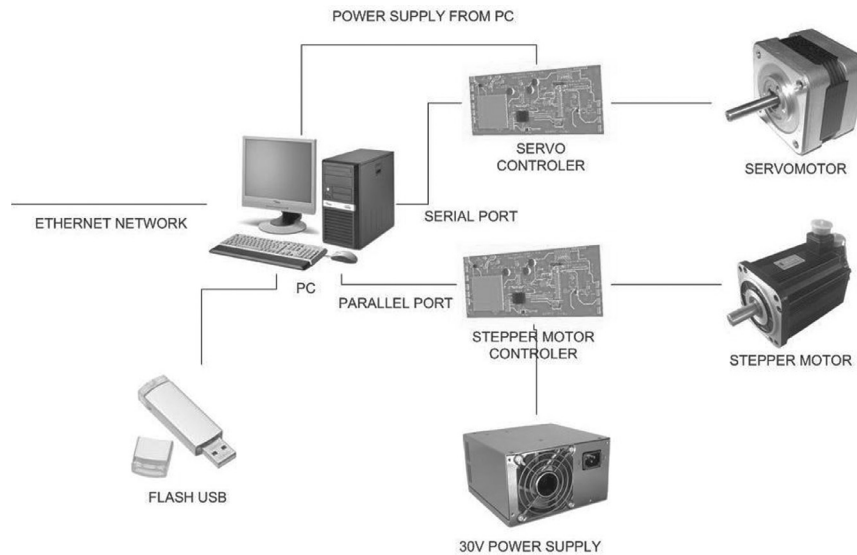


Figure 1. Elements of the embedded system.

To understand how amateur radio satellites work, it is necessary to describe some key concepts and definitions:

Azimuth: Azimuth refers to the rotation of the whole antenna around a vertical axis. It is a side to side angle measured clockwise with respect to the Earth's North Pole.

Equatorial orbit: The satellite spins around the Earth's equator in this kind of orbit.

Elevation Angle: This is the angle of the ground station from the horizon. If the angle is small, the distance the radio waves need to travel is longer.

GEO (Geosynchronous Orbit): In this type of orbit, satellites are at a distance of 35,848 Kilometers above the equator. At this altitude, the rotation period of satellites is 24 hours, which makes them appear in the same place from the planet surface.

Inclined Orbit: Any other kind of orbit is called inclined orbit because it goes from one side to another crossing the equator (Tomasi 1996).

LEO (Low Earth Orbit): These satellites orbit at 5,035 Kilometers above sea level or even lower, between 600 and 1,600 Km. (Roddy 2006). The amateur radio satellites are LEO Satellites. When a ground (amateur) station, A, wants to transmit a signal, it is first received by the satellite and then immediately amplified and retransmitted. After that, another amateur

station, B, receives the signal and starts communication. To perform the communication, two types of angles from the ground station must be known (Tomasi 1996).

MEO (Mid Earth Orbit): The satellites with this kind of orbit are between 10,075 and 20,150 Kilometers from the Earth. This type of satellite is not geostationary.

Polar orbit: Type of orbit in which a satellite passes above or nearly above both poles of the Earth.

Satellite: A satellite is an object that spins around the earth; specifically, a communications satellite is a radio frequency repeater that orbits the Earth. A satellite system consists of a transponder, a terrestrial station to control the satellite, and a group of users. The correct relations between them provide the facilities to the correct reception of the radio signals in a satellite system (Cakaj 2009; Tomasi 1996). Orbital satellites spin around the Earth using different types of orbits.

Trajectory: A trajectory defines a geostationary orbit, which is a geosynchronous orbit above the Earth's equator that has an orbit period equal to the Earth's rotational period and an orbital eccentricity of approximately zero. An object in a geostationary orbit appears motionless to ground observers.

The satellites stay in orbit due to the centrifugal force caused by the Earth's rotation. LEO satellites have closer orbits to the Earth and their speed is around 7,800 m/s. At this velocity, satellites orbit the earth in approximately 1.5 hours. Because of this, the satellite can only be seen on the horizon for about 15 minutes.

In order to know when and where a satellite is going to pass over a specific point on the Earth, there are several tools to calculate the pass. With the help of a computer and these tools, we can predict the coverage, direction, and position of a particular satellite. To calculate these parameters, we need the information contained in the Keplerians. Keplerian elements are the inputs to a standard mathematical model of spacecraft orbits and are formed by seven numbers. This set of numbers defines an ellipse, orient it about the earth, and place the satellite on the ellipse at a particular time. With the Keplerians, the correct time, and your local station, it is possible to compute the exact satellite position in the sky and where to point a satellite antenna (Leick 2004).

GNU/Linux for Embedded Systems

Having described the different types of satellites and their orbits, it is necessary to know about GNU/Linux embedded systems, which we used

for our satellite tracking module. GNU/Linux is being used as a platform for embedded systems in diverse applications. The main advantages of using GNU/Linux for embedded systems are the following (Sally 2009):

- Source code portability.
- GNU/Linux has support in a wide range of architectures.
- Long Term Support.
- Total access to the Kernel Source code and the ability of changing it.

In order to develop an efficient and reliable Linux version for our embedded system, we followed five general steps and ten best practices.

The following are the general steps defined by (Chou et al., 2007):

- Choose the right hardware platform.
- Get an efficient development environment. (BuildRoot is recommended).
- Get the necessary packages for our embedded system.
- Once packages are obtained, it is necessary to integrate all of the above.
- Test the System.

The following are some of the best practices to develop an efficient and small embedded system, according to (Chou et al., 2007):

- Remove unused executables.
- Only install the needed libraries.
- Compile the software platform kernel only with the options needed by the system.
- Remove unused libraries.
- Use a lightweight library like uClib.
- Compile the libraries manually.
- Remove manual pages (“man”) and documentation.
- Use a filesystem like CRAMFS.

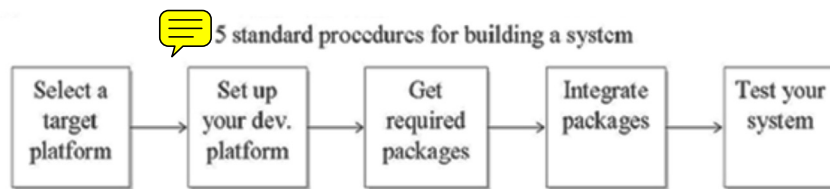


Figure 2. Steps for designing an embedded system (Chou et al., 2007).

After the kernel compilation is performed, one must decrease the “STATIC_MEMORY_ALLOCATION” parameter.

Do not compile unused kernel modules. Also, it is better to compile the kernel in a non-modular way.

Preparation of the Embedded System

The option of creating a GNU/Linux distribution from scratch was discarded because for prototyping software development purposes it is better to have the support and applications right away (Yaghmour et al., 2008). We developed the software system for our embedded system using Slackware Linux O.S. In addition, we used Qemu to install the Linux distribution on a USB flash drive. This is an x86 compatible processor emulator. With this software we can run the installation from a computer running an O.S. other than Linux (Surhone et al., 2010).

Installing GNU/Linux in a USB Flash Drive

For software development purposes, a 2 GB Flash drive it is recommended. We needed to delete the current flash drive partitions and creates a new one using all the space available. It also needs to have the bootable flag. To install Slackware on the flash drive we needed to follow the following steps:

USB Support

To create the partition, we first need to identify the device file of the USB Flash drive. This can be done following the following command from the Slackware Terminal:

```
root# dmesg
```

```
[118828.614766] sdb: assuming drive cache: write through
```

```
[118828.614769] sdb: sdb1
```

This command provides access to the kernel log, where it can be confirmed that Linux has detected the flash memory with the device name “sdb”.

Partitioning

After we have identified the name of the flash drive device, we started its partitioning with the following command:

```
root# cfdisk /dev/sdb
```

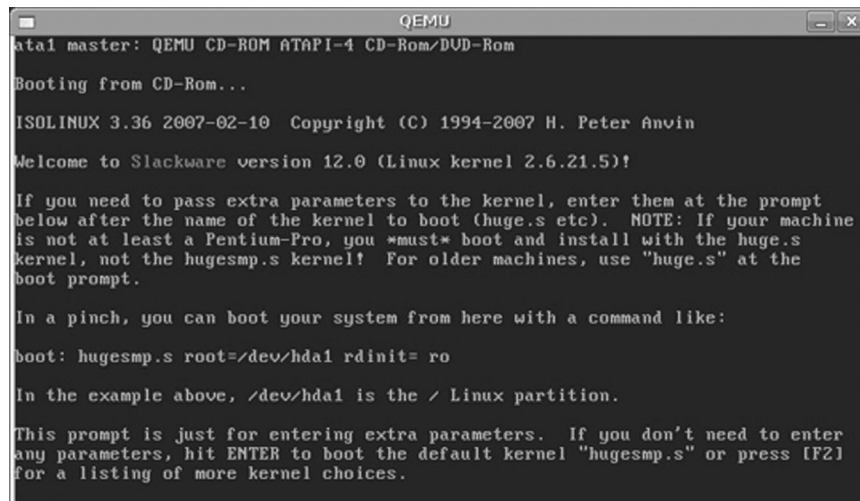
This command deletes all the partitions and creates a new one. Users must make sure to mark the partition as bootable.

Then the Qemu must initiate the installation (Fig. 3):

```
root# qemu -hda /dev/sdb1 -cdrom slackware-10.1-disc1.iso -boot d
```

The installer is run with the following command:

```
root# setup
```



```

QEMU
ata1 master: QEMU CD-ROM ATAPI-4 CD-Rom/DVD-Rom
Booting from CD-Rom...
ISOLINUX 3.36 2007-02-10 Copyright (C) 1994-2007 H. Peter Anvin
Welcome to Slackware version 12.0 (Linux kernel 2.6.21.5)!
If you need to pass extra parameters to the kernel, enter them at the prompt
below after the name of the kernel to boot (huge.s etc). NOTE: If your machine
is not at least a Pentium-Pro, you *must* boot and install with the huge.s
kernel, not the hugesmp.s kernel! For older machines, use "huge.s" at the
boot prompt.
In a pinch, you can boot your system from here with a command like:
boot: hugesmp.s root=/dev/hda1 rdinit= ro
In the example above, /dev/hda1 is the / Linux partition.
This prompt is just for entering extra parameters. If you don't need to enter
any parameters, hit ENTER to boot the default kernel "hugesmp.s" or press [F2]
for a listing of more kernel choices.

```

Figure 3. Slackware booting process.

Once placed in the installation menu, users should choose the “target” option and select /dev/hda1. Then the Ext2 should be selected as the file system for the partition (Fig. 4). Ext2 is an improved version of Fat32 File system. For a long time Ext2 was the default file system on most of the GNU/Linux distributions and it is suitable for this project since it is stable and does not have journaling (a file system that is used in the event of a system crash or power failure).

The installation process copies and install of all the packages into the flash drive. This process takes about 30 to 45 minutes. Importantly, Slackware uses LILO as the default bootloader. Finally, it was installed in the MBR of the flash drive. After that, the installation was completed.

Kernel Building

The latest version of the Linux kernel is recommended because it provides support for newer hardware and avoids old bugs. It is also better to build



Figure 4. File System selection menu.

```
root# make modules && make modules_install && make bzImage
```

After this, users must select the modules they wish to install on the kernel. Do not forget to mark them as static modules (*) and decrease the "STATIC_MEMORY_ALLOCATION" parameter. Users should also take into account the support for the ext2/ext3 file system.

The following must be executed to build the kernel:

```
root# make modules && make modules_install && make bzImage
```

The build process will then create a Linux kernel image. This image needs to be moved to the /boot directory using:

```
cp arch/i386/bzImage /boot/linux-kernel
```

After copying the kernel image, a series of changes needs to be done to the LILO configuration file in order to boot the new kernel. This time the changes need to be done in a live environment using Qemu again:

```
root # qemu -hda /dev/sdb
```

The first change needs to be done on the /etc/fstab file. The line that mounts the root / partition must be changed from:

```
/dev/hda1 /      ext2  defaults 1      1
```

to:

```
/dev/sda1 /      ext2  defaults 1      1
```

the kernel using the host computer rather than using Qemu since the build process is I/O, CPU, and memory intensive.

The steps to build the kernel are the following:

Have root access to the build computer.

Mount the flash drive.

Execute a chroot to the directory where the flash drive is mounted.

Download the kernel code from <http://kernel.org>

Decompress the source code.

Execute "make menuconfig"

Configure the kernel parameters.

Build and install the kernel.

The kernel building process will run natively using this procedure after executing chroot and running the following command:

```
root# mount -t ext2 /dev/sdb1 /SlackwareMemoria
```

After this, the working environment must be changed. The chroot command realizes this process:

```
root# chroot /SlackwareMemoria
```

The purpose of the previous steps was to create a Slackware terminal. The next step is to download the kernel source typing these commands:

```
root# cd /usr/src/
```

```
root# wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.25.4.tar.bz2
```

The kernel source comes stored in a TAR file, so an un-TAR needs to be run:

```
root# tar xjf linux-2.6.25.4.tar.bz2
```

Next, the menuconfic command needs to be executed

```
root# make menuconfig
```

The File Systems section activates the support for both the USB file system and the support for the ext2 file system. After this, the following command is used to compile and install it.

The next modification is to the LILO configuration file located at /etc/lilo.conf. This step is important because it defines the kernel image to load in the boot process. An important detail to keep in mind is that the Linux kernel is multi threaded; therefore, it is mandatory to stop the kernel execution a few seconds to let the motherboard initialize the USB Ports. This can be achieved by setting up the rootdelay directive defined in the append section in the LILO config file:

```
image = /boot//linux2.6  
root = /dev/sda1  
label = linux_SDA  
append = "rootdelay=8 ramdisk_size=16000"  
read-only
```

Next, the LILO must be installed:

```
root# /sbin/lilo
```

Design and Control the Elevation and Azimuth Mechanisms

Servo motors for Elevation System

Servo motors, also called servos, are direct current motors that have the capacity to spin to a desired position within their operation range. To do this, the servo needs a control signal, which is a pulse modulated signal. The duration of the pulse tells the servo how many degrees to spin. Thus, the operation range for common servos is 0 to 180 degrees.

Servo Motor Control Board

The control board has a PIC microcontroller, capacitors, and resistors. This board uses a PIC16F84 microcontroller from Microchip to optimally drive servo motors and digital outputs. The control board receives commands from the host embedded system via a standard RS232 serial interface. It was designed by Ashley Roll (an Australian engineer) who released its design to the public domain (Fig. 5).

To enable the controller and start moving the servo motor, it is necessary to send control commands to it. These commands have a 1-Byte length. The board has the following control commands:

a) Reset

The reset is needed when an emergency stop or any channel deactivation is required. This command is a zero ("0") succession and the board interprets them as a RESET and shut down all the outputs.



Figure 5. The PicServo controller (Roll 2000).

b) Channel activation/deactivation:

Activation

To move the servo, the channel that is connected to the motor must be enabled. The activation command is 48 Decimal + the channel number. For example, to enable Channel 4, we need to send the command $48 + 4 = 52$.

Deactivation

To deactivate a channel, the command is 64 + channel number. In other words, to deactivate Channel 4, the number 68 needs to be transmitted to the control board.

c) Servo movement 'position'/'offset'

In order to achieve the full 180 degrees of movement, we need to handle two types of movements; the position which goes from 0 to 90 degrees and the offset, which is an extra movement that also goes from 0 to 90.

Elevation Control Daemon

This daemon handles the elevation movement and the communication with the control board. It is possible to have 512 positions that cover 180 degrees. However, the system only requires 90 degrees. With a simple equation, we found a multiplication factor to move the motor. If users want to move the

servo 60 degrees, they need to send the code to move the motor and the number $60 * 2.8444444$ to the control board.

In order to use this daemon, the tracking software needs to write information to a UNIX Socket. The protocol is simple; the software only needs to send the degree to position the servo. For example, to move the servo 60 degrees, the tracking software only needs to open the socket and write 60.

2) Azimuth movement of the Stepper Motor

Stepper motors are used when precise movements are required. The main advantage of these kinds of motors is that they are pulse-driven and they move the same number of degrees each time. In addition, they remain locked on each position unless another pulse is applied.

There are two types of stepper motors: unipolar and bipolar. This project uses unipolar motors since they are easy to drive and they are widely available. This motor has five to six cables. Four cables are for the coils and the remaining ones are for ground. The basic setup in the control circuit for this kind of motors consists of an array of transistors which is found in the ULN2003 integrated circuit.

To drive these kinds of motors, it is necessary to activate their coils in a specific order. We use the wave drive sequence which provides a smooth movement and good speed of rotation. The disadvantage of this sequence is that only one coil is activated each time and this produces a loss of power. This can be solved with the use of a lightweight antenna attached to it.

Electromechanical Platform and Control Software

The mechanical platform was built using a stepper motor and a gear system taken from an old dot matrix printer (Figures 6 and 7). This gear demands the stepper motor to make 866 steps to complete a full spin. Using a simple mathematical procedure, a factor was calculated to obtain the number of steps required to move a certain number of degrees.

The parallel port of the PC used to communicate with our satellite tracking system did not deliver enough current to drive the stepper motor. Thus, a power interface, which uses transistors, was designed to isolate and protect the parallel port (Figure 8). The transistors we used are TIP120 NPN, which provide enough current to drive the stepper motor. Also, opto-couplers were used to isolate the parallel port from the high current of the transistors.

The parallel port control software responds to a series of codes that define the direction of rotation and the number of degrees to spin. It was



Figure 6. Stepper motor used for the Azimuth.



Figure 7. Gear system.

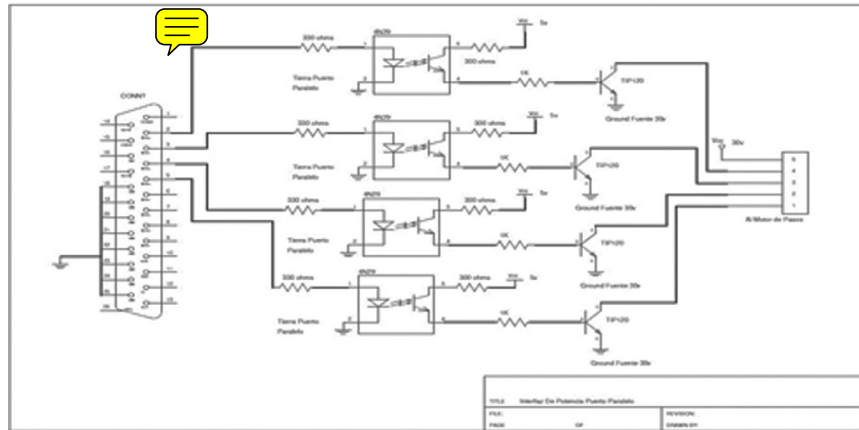


Figure 8. Parallel Port Electronic Interface.

necessary to define memory allocation for the parallel port, where the allocation address was the hexadecimal value 0x378h.

In order to write to the software to communicate through the parallel port, the software needs to ask the kernel for special access rights and be executed as root user. The request for the access is issued by the *ioperm* function. This function establishes the access bits for the memory address on which the parallel port resides.

The process of writing data to the parallel port is done by the *outb* function.

Prediction/Scheduling and Orbit Calculation

The most important software subsystems of our embedded system are: orbit prediction, orbit calculation and scheduling.

Orbit Prediction

This subsystem consists of a series of web pages that are used to predict the orbits and schedule them. This is the main user interface of the embedded system and the user (the operator) interacts directly with it.

Orbit Calculation

This is the most important subsystem of the project. It calculates the position of any satellite in real time. The information obtained from this subsystem is used to drive the azimuth and elevation subsystems.

Scheduling

This subsystem is in charge of verifying if there is a scheduled pass in real time and start the orbit calculation subsystem.

To achieve orbit prediction, scheduling, and orbit calculation, this project uses a piece of software called “Predict” which is delivered under the GNU/GPL License and can be freely used.

Antenna

To select or construct the antenna, we considered the following aspects:

- Easy to build.
- Light weight.
- Inexpensive.

The project uses the CJU (Canary Jail Umbrella) (Figure 9), which was designed by two amateur radio operators from Spain (the operators’ amateur radio call signs are EB4DKA and EA4CYQ). This type of antenna has all the above mentioned construction aspects and a radiation pattern that is not as directive as a Yagi antenna.

Results

Our first test was carried out on the prediction system selecting different LEO satellites to verify the correct orientation of the antenna. This was done

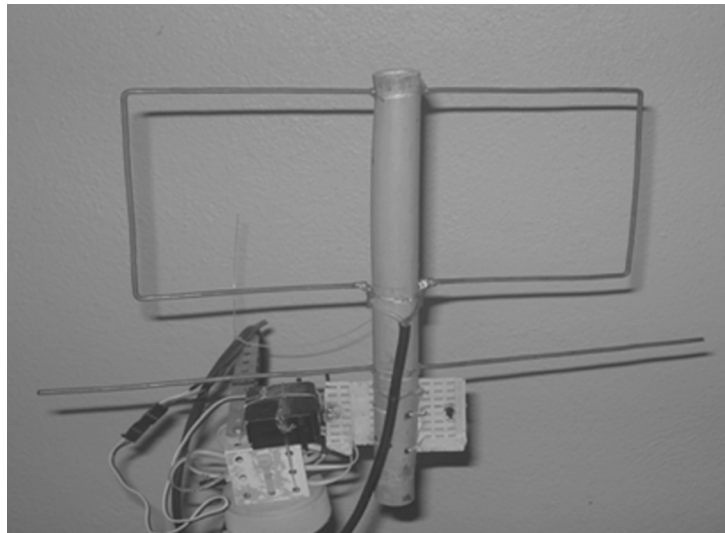


Figure 9. CJU antenna.

using the control algorithm. The angle was measured with an inclinometer, which matched with the values obtained from the Orbitron prediction software. The physical azimuth and elevation values that we obtained were exact, and suffered no modification in the algorithm software and gear system. The velocity and strength of the two rotors was sufficient and it was possible to support more weight from the antenna to increase the gain in the reception audio from LEO satellites.

The CJU antenna was capable of receiving incoming signals from a satellite during 70% of its orbit when the minimum elevation angle was 10 degrees. The only adjustment done in the CJU antenna was in its polarity. The first test included mounting the antenna with a vertical polarization that received some fading signals from the some LEO satellites. To solve this, we changed the antenna to horizontal polarization, but we obtained more fading. Upon orienting the antenna manually, the polarization can be changed manually between vertical and horizontal to correct o decrease fading. In this case, the best audio reception was when the antenna was mounted between the two polarizations (45 degree clockwise). That position best received an acceptable audio signal from the satellite, covering 70% of the satellite orbit.

The embedded system and the Slackware Linux worked acceptably with memory and CPU use was really low, with sufficient excess capacity to support other processes or applications in the future. The 2 GB Flash drive was enough for the applications and software that we developed; the response time for all the processes let the embedded system be quickly pointed at and follow the LEO satellites, comparable to commercial solutions.

Conclusion

Our embedded system makes the tracking of LEO satellites easier to perform than manually pointing an antenna, thus significantly reducing orientation errors and enhancing the quality of audio received from an amateur satellite.

The modular approach used in our system allows anyone to change procedures to control the azimuth or elevation rotors without affecting the rest of the system. The use of an embedded system reduced the energy consumption. We obtained the same results as a commercial solution while using only a minimal hardware setup.

Most of the Linux control daemons (pieces of software) where written in PERL language and only the parallel port control software was written in C. This can be seen as an advantage because the software development

time can be dramatically, compared to other programming languages (Love 2007).

GNU/Linux can be compiled as a reduced version, which includes only the modules and programs that we only needed for our particular satellite tracking application. This allowed more control of the embedded architecture and reduced the memory, storage, and CPU usage. In addition, GNU/Linux is a quick and inexpensive way to develop and control embedded systems and it has many technical advantages.

A number of videos of our satellite tracking system prototype can be downloaded from this link: <http://xe1azj.sat-xe.org/video.html>

REFERENCES

- Alvarez, O., M.G. Mayoral, M.A. Garcia-Ruiz and H.E. Preciado. 2010. Antenas IOio y Moxon para satélites LEO. *Onda Corta*, No. **424**:7–10.
- AMSAT. 2011. Professional Tracking Software for Purchase. Available from <http://www.amsat.org/amsat-new/tools/software.php>.
- ARRL. 2011. ARRL Handbook. Eighty-Eighth/One Edition. The American Radio Relay League, (ARRL). Newington, CT.
- ARRL. 2010. The ARRL Handbook for Radio Communications: The Comprehensive RF Engineering Reference, 88th Edition. American Radio Relay League (ARRL), Newington, CT.
- ARRL. 2007. The ARRL Antenna Book: The Ultimate Reference for Amateur Radio Antennas, Transmission Lines and Propagation. 21st edition. American Radio Relay League, Newington, CT.
- Britian, K. 2009. Cheap Antenna Designs for the AMSAT LEOs. Kindle Edition. Wilhite Write. San Francisco, CA.
- Cakaj, S. 2009. Practical horizon plane and communication duration for low earth orbiting (LEO) satellite ground stations. In Proceedings of the 8th WSEAS international conference on Telecommunications and informatics (TELE-INFO'09), 62–67.
- Chou, C.H., T.H. Yang, S.C. Tsao and Y.D. Lin. 2007. Standard Operating Procedures for Embedded Linux Systems *Linux. Journal*, Volume 2007, Issue. **160**:10.
- Davidoff, M.R. 2009. The ARRL Satellite Handbook. American Radio Relay League (ARRL), Newington, CT.
- FCC. 2011. Federal Communications Commission. Amateur radio service. Available from: http://wireless.fcc.gov/services/index.htm?job=service_home&id=amateur
- ITU. 2008. Amateur and Amateur-satellite Services. Radiocommunication Bureau, International Telecommunication Union, Geneva, Switzerland.
- Leick, A. 2004. GPS Satellite Surveying, Third Edition. John Wiley and Sons, Hobokewn, NJ.

- Love, R. 2007. *Linux System Programming: Talking Directly to the Kernel and C Library*. O'Reilly Media, Sebastopol, CA.
- Newport, J. (n/d). *Amateur Satellites as a Vehicle for Satellite Communication Education*. Available from <http://www.oosa.unvienna.org/pdf/sap/centres/satcomE.pdf>.
- NOAA. 2011. National Oceanic and Atmospheric Administration website. Available from: <http://www.noaa.gov/satellites.html>
- Roddy, D. 2006. *Satellite Communications, Fourth Edition*. McGraw-Hill Professional, Berkely, CA.
- Roll, A. 2008. PICServo controller. Available from <http://www.digitalnemesis.com/info/projects/picservo/>.
- Sally, G. 2009. *Pro Linux Embedded Systems*. 1st edition. Apress, Berkely, CA.
- Taggart, R. 1996. *Weather Satellite Handbook*, 5th edition. American Radio Relay League (ARRL), Newington, CT.
- Tomasi, W. 1996. *Electronic Communications Systems*. Pearson Education, Upper Saddle River, NJ.
- Surhone, L., M. Timpledon and S. Marseken. 2010. *QEMU*. Betascript Publishing, Mauritius.
- Yaghmour, K., J. Masters and G. Ben-Yossef. 2008. *Building embedded Linux systems*. 2nd Edition. O'Reilly & Associates, Sebastopol, CA.