A GPS Data Display

Peter D. Hiscocks
Royal Astronomical Society, Toronto Centre, Canada
Syscomp Electronic Design Limited
Email: phiscock@ee.ryerson.ca

June 20, 2010

Introduction

There are many applications where one wishes to acquire geographic position data from a GPS receiver. In this particular instance, I needed to acquire a series of street light level readings and tag each one with its location[1]. To get started, I decided to write a stand-alone program to display GPS data.

The program is written in the Tcl/Tk language, to acquire and display the data from a Pharos GPS-500 receiver. The Pharos receiver was bundled as part of the Microsoft *Streets and Trips* software package, which was being remaindered at a local computer discount house. The Pharos receiver produces data in standard NMEA[2] string format (comma-separated values). The receiver appears as a `USB-Serial` device, that is, it uses the USB hardware and power connection, but appears to the host computer as a serial port device. This enormously simplifies interaction with the receiver, and is common practice for GPS receivers. Consequently, it should be straightforward to use this Tcl program with other GPS receivers.

A description of GPS is in Wikipedia. A quick introduction to the Tcl/Tk language is in the paper `Hello Button,` *An Introduction to Tcl/Tk* [3].

The upper image in figure 1 shows the receiver, plugged into one end of a USB cable. The receiver can plug directly into a computer GPS port. However, in my experience with an Acer Aspire 3610 laptop, the reception was significantly improved (ie, number of satellites received increased) by operating the receiver on the end of its cable.

The lower image in figure1 shows the graphical user interface. The upper area contains the summary information that is most useful to an operator: longitude, latitude, altitude and UTC (Universal Time, Coordinated)[4]. The `Num Satellites` entry shows the number of satellites currently in use. The `Date` is the current date in UTC zone zero.

Figure 1: Receiver and Display

The lower area shows the satellites currently detected by the GPS receiver. There are usually more satellites listed than `Num Satellites` indicates, presumably because reception from some satellites is too weak to be reliable.

[1] The results of that project will be reported in a separate paper.
[2] NMEA: National Marine Electronics Association, see `http://en.wikipedia.org/wiki/NMEA_0183`.
[3] `http://www.syscompdesign.com/AppNotes/tcl-intro.pdf`
[4] Local time is offset from UTC by the local time zone. For example, Toronto, Canada is -5 hours with respect to UTC. During summer months, Daylight Savings is in effect, which makes Toronto -6 hours with respect to UTC.

- **PRN** Pseudo-Random Number, a number assigned to the satellite.
- **Elevation, Azimuth** The elevation and azimuth of the satellite as observed from the receiver location.
- **S/N Ratio** Signal to noise ratio. A larger signal indicates more reliable reception.

The satellites appear to be ordered by elevation angle, and signal-noise increases with increasing elevation. For weak signals, quite frequently the S/N is not indicated.

The lower area of the display is not essential.

It's adviseable as a first step to make sure the computer is recognizing the receiver as a serial-port device. Here's how you do that on Windows and Linux machines.

Windows: Checking the COM port Connection

Right click on `My Computer`
Navigate to `My Computer -> Manage -> Device Manager`
Double click on `Ports (COM and LPT)`
You should see something like `Microsoft USB-GPS Port (COM3)` If the port number is it not `COM3`, then right click on that entry and select `Properties`
Navigate to `Port Settings -> Advanced` and select `COM3`. Ignore any warning messages about other equipment using this COM port.

Linux: Checking the COM port Connection

On a Linux machine, the USB-Serial ports are assigned on a first-come first-served basis. To determine the assigned port, plug in the receiver to a USB port, open a terminal window and run the program `dmesg`. At the end of the message, you should see something like this:

```
usb 3-1: new full speed USB device using uhci_hcd and address 7
usb 3-1: new device found, idVendor=067b, idProduct=aaa0
usb 3-1: new device strings: Mfr=1, Product=2, SerialNumber=3
usb 3-1: Product: USB-Serial Controller D
usb 3-1: Manufacturer: Prolific Technology Inc.
usb 3-1: SerialNumber: 12345678
usb 3-1: configuration #1 chosen from 1 choice
pl2303 3-1:1.0: pl2303 converter detected
usb 3-1: pl2303 converter now attached to ttyUSB0
```

This indicates that the receiver is attached to `/dev/ttyUSB0`, which is the port hard-coded into the program. If the port number is *not* `ttyUSB0`, then you have two choices: edit the Linux COM port selection in the source code of the program to match the selected port, or remove existing USB devices until the receiver is the first one selected. You can then plug in the other USB devices, which hopefully are more flexible about using different USB-serial ports[5].

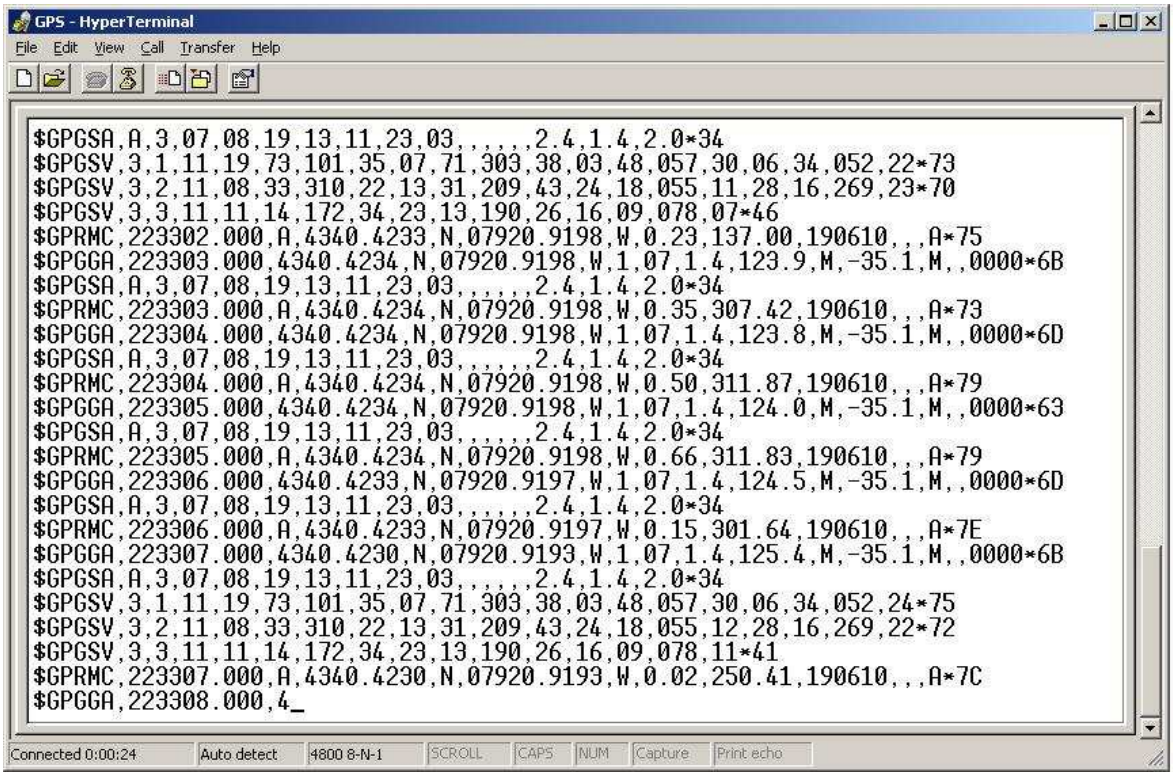

Figure 2: Terminal Display of NMEA Messages

Displaying NMEA Messages in a Terminal Window

Messages are streamed continuously from the GPS receiver. They may be displayed in a terminal window. First, plug the Pharos GPS-500 into a USB port.

On a Windows machine, figure 2 shows a typical display of messages as displayed in the `Hyperterminal` terminal emulator window, with the the terminal parameters set to 4800,8,N,1 (4800 baud, 8 data bits, no parity, and 1 stop bit). The GPS receiver and the terminal terminal are connected to `COM3`.

On a Linux machine, run `dmesg` to determine which port the receiver is assigned to. Start `gtkterm` or some other terminal emulator program. Configure the port selection to something like `/dev/ttyUSB0` (or whatever), and `4800,8,N,1` You should see messages streaming from the device.

Running the Program

Windows

On a Windows machine, plug in the GPS hardware and check device manager (under `Ports:COM and LPT`) and ensure that the device is assigned to COM3. If that's not the case, use the COM port properties to change the

[5]This is, we candidly admit, an incredibly crude approach to port selection. For port selection that provides more options, download one of the Syscomp instrument programs at `\http://www.syscompdesign.com/download.html`, browse the source code, and see what was done there.

assignment to COM3 or modify the source code to point at the assigned port.

If the Tcl language is not installed on your machine, download it from the ActiveState website: `http://www.activestate.com/activetcl` (use the `Free Download` button).

Once the ActiveState Tcl interpreter is downloaded, any file ending in `.tcl` should be associated with the Tcl interpreter. Then you can start the program by double-clicking on the source code icon for `gps-display.tcl`. Alternatively, start the Tcl/Tk interpreter by double-clicking on the `wish` icon. A terminal window will appear. In that window, type `source gps-display.tcl`. The program should start.

Linux

Verify that the Tcl interpreter is present by typing into a terminal window the command `which wish`. You should get an answer like `/usr/bin/wish`, indicating that Tcl is present.

You can also simply type into a terminal window the command `wish` which will startup the Tcl interactive window.

On a Linux machine, plug in the GPS hardware and use the command `dmesg` to verify that the devices is assigned to the serial port '/dev/ttyUSB0'. If that's not the case, either unplug other GPS devices or modify the source code to match the assignment. In a terminal window type `wish gps-display.tcl` and the program should start.

Mac

The code has not been tested on a Mac but it should run.

If the Tcl language is not installed on your machine, download it from the ActiveState website: `http://www.activestate.com/activetcl` (use the `Free Download` button).

Ensure the Tcl interpreter is downloaded and installed. Plug in the hardware. Determine the port that the hardware is assigned to, which might be something like `/dev/cua0`. Using the Linux and Windows port assignment code as a guide, modify the source code to add detection of the port on a Mac. You could also look at the source code for the Syscomp DVM-101 voltmeter[6] to see how that is done.

Details of the Software

Decoding NMEA Messages

The decoding of GPS messages is based on information at: *Index of Dale DePriest's Navigation and GPS Articles*[7].

Sentences are known by their preface, which might be `$GPGGA`. The first three characters are common to all sentence names: the last three describe the sentence. In the program, we decode three different sentences (and ignore others). The `GGA` and `RMC` sentences supply longitude and latitude, and other essential information. The `GSV` sentence (actually, group of sentences) supplies data on the satellites.

[6] See `www.syscompdesign/downloads.`

[7] `http://www.gpsinformation.org/dale/nmea.htm`

Here is an example, the GGA message, *Essential fix data which provide 3D location and accuracy data*. The message looks like this:

```
$GPGGA,022229.533,4340.4155,N,07920.9126,W,1,04,1.7,139.6,M,-35.1,M,,0000*61
```

It decodes as follows:

```
GGA           Global Positioning System Fix Data
022229.533    Fix taken at UTC time
4340.4155,N   Latitude 43 degrees etc N
07920.9126,W  Longitude 79 degrees etc W
1             Fix quality: 0 = invalid
                           1 = GPS fix (SPS)
                           2 = DGPS fix
                           3 = PPS fix
                           4 = Real Time Kinematic
                           5 = Float RTK
                           6 = estimated (dead reckoning) (2.3 feature)
                           7 = Manual input
                           8 = Simulation mode
04           Number of satellites being tracked
1.7          Horizontal dilution of position
139.6,M      Altitude, above mean sea level, metres
-3 5.1,M     Height of geoid (mean sea level) above WGS84
                 ellipsoid, metres
(empty field) Time in seconds since last DGPS update
0000         DGPS station ID number
*61          the checksum data, always begins with *
```

Notice that all variables are separated by commas, so this is a CSV (comma-separated-variable) string. The leading $ sign is a potential problem, since Tcl uses it to signify the 'value of a variable'. The checksum is after the asterisk, at the end of the string. Since this is not mission-critical software, it does not compute the checksum, which would verify the integrity of a sentence[8].

Why Use Tcl/Tk for this Project?

This program was written to provide the simplest possible display of GPS data. The advantage (I hope) is that it's relatively easy to understand. The disadvantage is that you, the reader, will have to add the bells and whistles.

The Tcl/Tk language is ideal for a project of this sort:

- Tcl/Tk is available as Open Source, that is, it is freely downloadable and the source code is available. It also runs under Windows, Linux and Mac operating systems, with very few – if any – differences[9].
- The code is interpreted[10], so there is no explicit edit-compile-debug cycle. It's shortened to an an edit-run cycle. Debug information surfaces during the run of the code.

[8] Ingo Cyliax, *Where in the World (Part 1): GPS Introduction*, shows Tcl code for computing the NMEA checksum in `http://www.circuitcellar.com/library/print/0899/Cyliax109/2.htm`

[9] The only difference in this program is the naming of the serial ports.

[10] More precisely, it appears to the user to be interpreted. In fact, there is a run-time compiler that optimizes code execution for speed.

- It's trivial to add print statements to the code to monitor the behaviour of varibles and determine the flow of control.
- It is very simple to construct a graphical user interface, and relatively few instructions can do the job.
- The Tcl language can be operated in command line mode (direct entry of commands) or by interpretation of source code in a text file. Command line mode allows one to very quickly test the operation of a command or group of commands.
- The Tcl language is a *scripting* language, and as such it has a wide range of string manipulation commands.
- An *event loop* is automatically created for handling of button events and incoming messages from the serial port. All of the machinery to implement this is hidden from the user.

Overall Structure of the Code

If you are used to conventional programming languages, the organization of a Tcl program is somewhat unusual. It consists of four sections:

- *Definition of variables* These are variables that are global in scope and of major importance in the program, such as the operating system of the hardware platform (Windows, Linux or Mac).
- *Definition of procedures* These procedures are called from other locations in the program.
- *Definition of the Graphical User Interface* This code establishes the hierarchy, shape and appearance of the user interface, with labels and variables specified into, for example, a rectangular grid. The `pack` command then invokes the GUI.
- *Start of the Program* This code initializes the system by opening a USB serial port to the hardware and sets up the event handler (the `fileevent` command) for messages arriving from the serial port. Every time a message arrives, the event handler passes its received string to the parsing procedure. The parser then updates various variables, which then automatically update on the display.

Enhancements

The program could be enhanced with the following features:

- Selection menu for serial port
- Display/Hide switch for the satellite display
- Capability of saving GPS data to a disk file. This could then be loaded into a mapping program to display a track.
- Display of local time and date, using a user-entered time offset value.

Reading the Source Code

The source code for this program is in the file `gps-display.tcl`. It can be viewed and modified in any editor, but is best viewed in an editor that is *tcl-aware*. Then the keywords of the Tcl language are highlighted.