eSalvage

Hobbyist adventures in electronics, mostly related to salvaging and tinkering with disparate parts found here and there (but also some "new" creations, mostly related to Arduino). There will likely be a shared structure for the entries: a description of the salvaged part, a couple of pictures, the schematics and a circuit simulation with QUCS. Usually I will run the simulation in order to (better) understand the behavior or theory of some parts of the circuit. Level: close to a novice...

domingo, 23 de abril de 2017

GPS and Arduino

0.- Readme

Nothing in this page is "new", of course.

It is a (reasonably complete, I think) compilation of information related to GPS and Arduino handling of GPS receivers.

If you are "simply" looking for direct printing out of GPS stuff on your screen, this video may help you. If you are a bit more curious, the following lines may be of interest. If, at the other end, you are an advanced programmer or have worked in details related to GPS for a while, probably nothing here will be new for you.

The starting point was that (as usual!) I simply had some practical trouble with several aspects of the GPS-Arduino thing, and it took me a bit of time to find the answers (which, once found, proved to be quite obvious!). Later I expanded it to provide a bit of depth into the reasons of why things are like they are.

The page is organized in five blocks:

- how GPS work,

- the GPS module GY-GPS6MV2 (also NEO6MV2),

- what is NMEA (meaning NMEA standard 0183),

- Arduino sketches for GPS receivers,

- Handling the GPS data.

1.- GPS

GNSS

There are several satellite positioning systems, and overall they are known as GNSS, for Global Navigation Satellite System.

GPS is American (initially named Navstar GPS), GLONASS is managed by Russia, Galileo is being developed from Europe, and China has BeiDou, also in construction.

Basis

The GPS (Global Positioning System) currently includes some 30 satellites orbiting the Earth at some 20,000 km of altitude.

The satellites are travelling at fixed orbits (not geostationary). They keep track of their position and radiate it, together with the time when the signal is sent.

At any time and any position on Earth, at least 4 GPS satellites are visible

The GPS receiver can calculate the distance to the emitting satellites in view. From the distance to at least four satellites the position of the receiver can be calculated.

Now with some more detail.

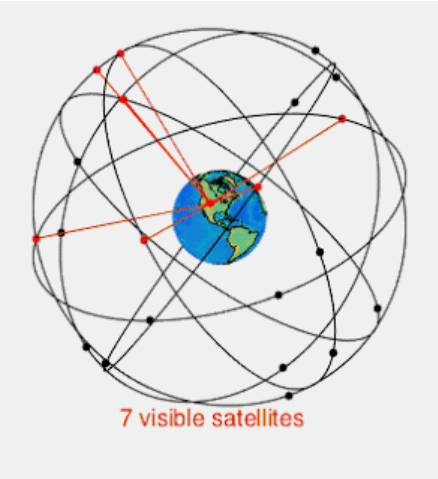
Position of the Satellites

The precise number of satellites varies depending on the source, from 24 to 32. The minimum is 24. Additional satellites allow for increased precision, while some may be temporarily down.

The satellites travel at high orbits, which apparently are quite predictable.

They are placed in six orbital planes, with at least 4 satellites on each. These orbits are equally inclined 55º from the Equator and are separated by 60º.

The satellites complete an Earth rotation twice per day, and due to the planet's rotation they repeat the same ground track once every day.



The theoretical position of all GPS satellites at any moment is stored in the "almanac" of a satellite. The almanac is, in general, valid for some 180 d as there are unforeseen actions on the satellite that may affect their position (errors in the initial placing location, gravitational forces from moon, solar pressure, loose atmospheric friction, etc).

There are verifications of the satellites from radar ground stations. The checks include position, altitude, speed and operation, and are performed twice a day.

The deviations from the foreseen position are collected in the "ephemeris" of the satellite, which are uploaded from the ground stations into each satellite once or twice a day. (However, a reference indicates that the ephemeris is valid for about 4 hours).

With the almanac and the ephemeris, the GPS receiver can determine the precise location of the emitting satellite.

The information on the almanac also allows the receiver to search for the satellites expected to be located within its field of view.

More information on almanacs and ephemeris: here, here and here.

Communication Emitter-Receiver

The satellites emit messages in the radio-micro-wave spectrum.

Two basic frequencies are used by GPS satellites: L1 at 1575.42 MHz and L2 at 1227.60 MHz, although up to five bands are studied or used. The frequencies are related to the on-board atomic frequency standard (10.23 MHz), itself related to the message coded length (see below).

The messages sent by the satellites contain three types of information:

- identification of the satellite and date+time of signal (plus other information on the health of the satellite),

- ephemeris data for the emitting satellite,

- almanac data for all the satellites in the system.

The structure of the message sent by a satellite is as follows:

- 25 frames of 1,500 bits per frame,

- each frame is divided into 5 sub-frames, each of 300 bits.

- each sub-frame is made up of 10 words, each of 30 bits.

The content is:

- the first sub-frame of each frame encodes the week, the time within the week, and data about the satellite's health,

- second and third sub-frames are used for the satellite's ephemeris,

- fourth and fifth sub-frames contain the almanac of up to 32 satellites, spread over the 4th and 5th sub-frames of the 25 frames of the message.

The navigation messages are broadcast at 50 bits/s, so the full message takes 750 s, or 12.5 min. Sub-frame #1 is therefore transmitted each 30 s.

All satellites transmit in the same frequencies, using encoding systems that allow the receivers to identify the emitting satellite.

There are two codes for the L1 and L2 carriers:

- C/A (Coarse Acquisition), for L1. It is the most common GPS signal, but also the less accurate.

- P(Y) (Precise Code), available with both L1 and L2 frequencies. Encrypted and reserved for military use.

The original low-bit rate (50 bit/s) message is encoded in a PRN (pseudo random noise) code.

The C/A PRN code has clock rate of 1.023 MHz (chip rate) and a period of exactly 1 ms (1023 chips).

The P(Y) PRN code has a clock rate of 10.23 MHz (chip rate) and a period of exactly one week (so it is a very long binary code).

The 50 bit/s data stream thus coded is then phase modulated over L1 or L2.

At the GPS receiver, the satellite message needs to be demodulated and separated from the signals sent by the other satellites with the same wave frequency.

More information:

- GPS Interface Control Document here.

- Wikipedia's GPS Signals, here.

Time and Distance

The calculation of the distance satellite-receiver is done measuring the time required for the signal to travel.

The travel time of the radio signal is approx. 65-70 ms. A positioning accuracy of 15 m requires to accurately measure 50 nanoseconds.

This accuracy is provided by atomic clocks, four of which are mounted on each GPS satellite. They are also monitored and controlled from the ground stations.

GPS receivers do not include atomic clocks, but just (high end) quartz crystals which by their lower accuracy have an offset to the satellite clocks (synchronized among them).

The time of transmission (TOT) of an element of the signal (the "epoch") is included in the message from the satellite. The time of arrival (TOA) of the epoch can be determined by the receiver's clock.

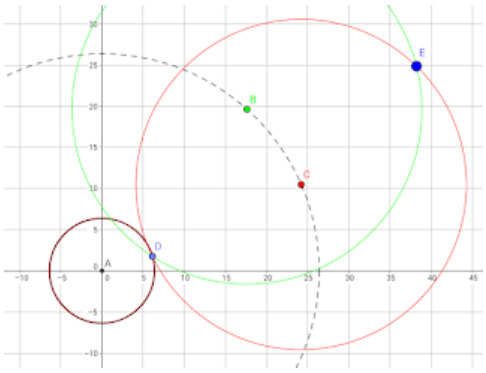
The time of flight (TOF) of the signal can be calculated, with the uncertainty of the offset between the satellites and the receiver's clocks.

The offset, constant for a measure, is an additional unknown to be solved.

Trilateration

It is sometimes stated that three satellite signals are needed to locate a receiver on the surface of the earth, and the fourth signal is required to calculate its actual elevation.

This does not seem to make much sense to me: the second geometrical solution is in the outer space, so it should be easy to grasp (point E in the following 2D sketch, which is to-scale).

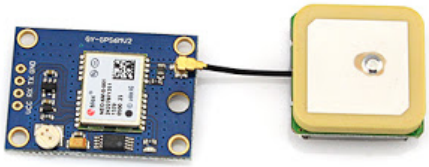


The fourth satellite is required to solve for the four unknowns: the three spatial coordinates of the receiver plus the time offset.

More than four satellites over-determine the system of equations and then a best fitting method is required.

2.- GPS module GY-GPS6MV2 (also NEO6MV2)

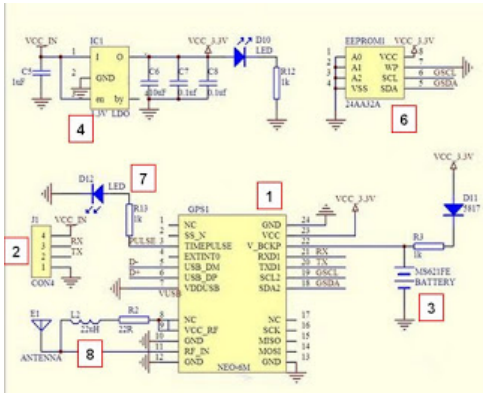
I bought this module some time ago, likely from Alibaba, maybe this one.

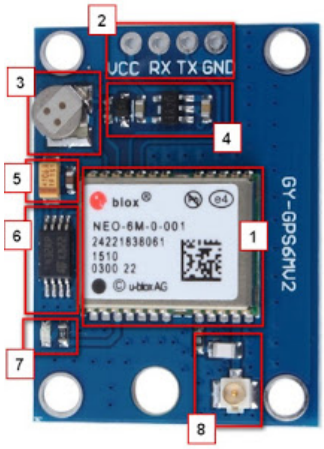


The Module

Here there is a rather detailed description of a GPS module which seems to be like mine, item by item- but unfortunately I can not guarantee.

The schematics:





U-blox' NEO-6M

The core of the module [1], the "GPS unit", is from u-blox, a company specialized in wireless communications and industrial positioning.

My chip is a NEO-6M-0-001 (and this probably is why I have seen the module also labelled as GY-NEO6MV2). Funnily enough, the NEO-6M is not any more in the manufacturer's list of products. Anyhow, the datasheet is here. The receiver description, including the protocol specification is here.

3.3 vs 5 V

This latter datasheet (and some webpages) warn about burning the module by connecting it to the 5 V of Arduino: it should rather be connected to 3.3 V, or to an adequate resistor connected to the VCC but also, apparently, to the communication pins between the Arduino and the GPS module.

Fact is that, at least my module, has an on board voltage regulator (KB33 MIC 5205; [4]) and I have worked with no problem both with 3.3 and 5 V.

However, it is true that I cannot be sure about the long term effect of the 5 V on the communication connections. Indications on two possible solutions are provided herewith diodes and here with resistors as voltage divider.

u-center

U-blox provides a software named u-center, which I find quite amazing. The company's page has a detailed User Guide, and a quick glimpse can be found here.

Ports

I have connected the module to the software in my PC through my USB to UART converter (chip CP2102).

(Please, just remind: RX is connected to TX, not to RX; otherwise it, of course, won't work.)

UART is not the only possible serial communication port. As indicated in the datasheet the receiver is "truly multi-port and multi-protocol [see below on this]". It has the following ports:

Electrical Interface
DDC (I2C compatible)
UART 1
UART 2
USB
SPI
reserved

Although the baud rate defined by the standard NMEA protocol (see below) is 4800 b/s, and the Arduino sketches work at 9600 bauds (see below), the UART ports can be configured up to 115200

b/s.

Battery

The module has a small battery [3], which is, apparently, used for the secure storage of configuration data on a EEPROM [5].

The battery is rechargeable, but it may be faulty. Mine, for instance has dropped from 2.9 volts last night to 0.9 V this morning. Upon re-connection it quickly grows again to 3.1 V, and one hour later, off the GPS, is at 2.5 V. (Maybe there is a small leakage somewhere in the UART converter and the USB extender cable).

This may create some difficulties: drain most of the current at the starting and delaying the actual operation of the GPS, and, perhaps, losing the information in the EEPROM.

See here, for instance, in Spanish.

LED

There is a small green LED [7] which blinks when the location is fixed.

The schematics above shows another LED, which I did not find in my module.

Antenna

The GPS antenna [8] should be kept with the ceramic part upwards, looking to the skies!

(I have managed to break the shielded cable at the connector. Amazingly I have managed to recompose the connection, although the cable is approx. 1/20th of my fingers).

3.- NMEA

What is it

The data received from a GPS module like NEO 6 follows the standard 0183 from NMEA (National Marine Electronics Association), which supports "one-way serial data transmission from a single talker to one or more listeners".

The standard seems to be complex enough as it covers a wide range of equipment, but, on the other side, it follows a strictly defined coding with simple ASCII text.

NMEA 0183 is copyrighted, and not easy to find (but it can be bought at the association's webpage). Latest version is 4.10 (2012), and the previous version (3.01, dated 2002) can be found here.

Sentences for Communication

There are many sources of information on NMEA 0183 standard, but I have found thisone particularly useful.

The key points, as far as I understand this issue, are:

- The communication from the GPS is done in sentences. One sentence is a line of data self-contained and independent from other sentences.

- There are standard sentences for each device category. There is also the possibility to define proprietary sentences for use by an individual company.

- The standard sentences have a two-letter prefix that defines the device. For GPS receivers the prefix is GP. This prefix is followed by a threeletter sequence that defines the sentence contents.

- Proprietary sentences begin with the letter P and are followed with 3 letters that identifies the manufacturer controlling that sentence: a Garmin sentence would start with PGRM.

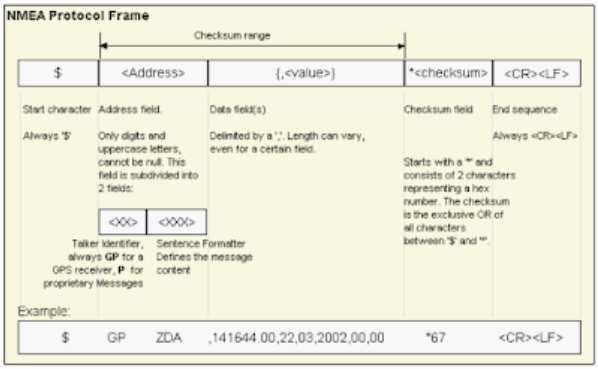
- Each sentence begins with a '$' and ends with a carriage return/line feed sequence (<CR><LF>).

- Sentences can be no longer than 80 characters of visible text (plus the line terminators).

- The data is contained within this single line. Data items are separated by commas. If data for a field is not available, the field is omitted but the delimiting commas are still sent, with no space between them.

- The data itself is just ASCII text. It may extend over multiple sentences in certain specialized instances, but it is normally fully contained in one variable length sentence.

- There is a provision for a checksum at the end of each sentence which may or may not be checked by the data receiver. The checksum field consists of a "*" and two hex digits representing the exclusive OR of all characters between, but not including, the "$" and "*".



Among the different sentences included in the standard and which can be of use for a GPS, I have just selected here one example:

$GPGGA,123519,4807.038,N,01131.000,E,1,08,0.9,545.4,M,46.9,M,,*47

Where:

GGA	Global Positioning System Fix Data
123519	Fix taken at 12:35:19 UTC
4807.038,N	Latitude 48 deg 07.038' N
01131.000,E	Longitude 11 deg 31.000' E
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 mode 8 = Simulation mode
08	Number of satellites being tracked
0.9	Horizontal dilution of position
545.4,M	Altitude, Meters, above mean sea level
46.9,M	Height of geoid (mean sea level) above WGS84 ellipsoid
(empty field)	time in seconds since last DGPS update
(empty field)	DGPS station ID number
*47	the checksum data, always begins with *

Implementation of Communication

With regard to hardware implementation:

- The interface speed can be adjusted on some models but the NMEA standard is 4800 b/s (bit per second rate) with 8 bits of data, no parity, and one stop bit.

- At 4800 b/s only 480 characters per second can be sent. As an NMEA sentence can be as long as 82 characters the communication can be limited to less than 6 different sentences (the actual limit is determined by the specific sentences used but this shows that it is easy to overrun the capabilities with rapid sentence response).

- There is s HS (high speed) NMEA standard.

- The receiver of the data can watch for the data sentence that it is interested in and ignore other sentences. There are no commands in the standard to indicate that the GPS should do something different. There is also no way to indicate anything back to the unit as to whether the sentence is being read correctly or to request a re-send of some data. Instead the receiving unit just checks the checksum and ignores the data if the checksum is bad figuring the data will be sent again sometime later.

Further details can be found in, for instance:

- NMEA Data

- NMEA0183 Protocol

- The NMEA 0183 Protocol

- GPS-NMEA Sentence Information

- NMEA Library

NEO 6

According to the protocol specification, the NEO 6 GPS receiver uses version 2.3 of the NMEA standard. Version 2.1 can also be selected by configuring the receiver.

The GPS receivers use NMEA protocols, but they can also communicate with the "proprietary UBX protocol".

The receiver can be configured in a number of ways, for example:

- dynamic platform settings, according to the use of the receiver that can improve the performance and accuracy: portable, stationary, pedestrian, automotive, at sea, and airborne.

- Navigation output filters, which control the validation of calculated fixes.

- Static hold, which I guess is an intermediate to the stationary and pedestrian settings above.

- Control of degraded navigation, with less than 4 satellites in use.

- SBAS (Satellite Based Augmentation Systems).

- Filtering of output data.

- Power management

The configuration of the receiver is done with the UBX configuration messages.

The UBX protocol is proprietary of u-blox, and it is described here.

The configuration of the GPS receiver with the UBX protocol can be done with u-center. Some pages on this: here, here.

(When you're done, go to Receiver>> Action>> Save Config. Otherwise, every time you unplug the GPS, it will revert back to the original configuration.)

4.- Arduino

Simple Sketches

A bare-bone sketch is GPS_bas_01, (here) which simply reads as text the data from the receiver and prints it down.

The resulting output is something like this:

```
$GPGGA,165512.00,3928.19043,N,00023.07436,W,1,05,1.61,107.0,M,50.1,M,,*43
$GPGSA,A,3,29,21,27,25,20,,,,,,,,2.42,1.61,1.80*01
$GPGSV,4,1,14,04,61,326,,05,06,049,,14,03,219,18,16,24,303,*77
$GPGSV,4,2,14,18,01,154,17,20,17,095,32,21,60,160,33,23,02,313,16*7A
$GPGSV,4,3,14,25,29,107,34,26,52,314,15,27,09,252,22,29,45,047,45*76
$GPGSV,4,4,14,31,60,221,18,37,39,147,28*76
$GPGLL,3928.19043,N,00023.07436,W,165512.00,A,A*77
$GPRMC,165513.00,A,3928.19045,N,00023.07460,W,0.304,,230417,,,A*6E
$GPVTG,,T,,M,0.304,N,0.562,K,A*25
```

As per the NMEA standard (see above) this would correspond to:

- $GPGGA (fix data): 16:55:12.00 (time UTC), 39º28.19043' N (lat), 0º23.07436' W (lon), 1 (data valid, GPS Quality Indicator), 05 (satellites in view), 1.61 (HDOP, horizontal dilution of precision), 107.0 m (altitude), 50.1 m (geoidal separation), 0 (age of differential GPS data), 0 (differential reference station ID), *4 (checksum).

- $GPGSA (GNSS receiver mode): A (automatic mode), 3 (3D), 29, 21 ,… (ID numbers of satellites in solution), 2.42 (PDOP, position dilution of precision), 1.61 (HDOP, horizontal dilution of precision), 1.80 (VDOP, vertical dilution of precision), *01 (checksum).

- $GPGSV (GNSS satellites in view): 4 (total number of GSV sentences), 1 (sentence number), 14 (total number of satellites in view), 04 (satellite ID number), 61 (elevation in º), 326 (azimuth in º), 0 (SNR, signal to noise ratio), 05, 06, … (data for up to 4 satellites in view per line), *77 (checksum).

- $GPGLL (geographic position latitude, longitude): 39º28.19043' N (lat), 0º23.07436' W (lon), 16:55:12.00 (time UTC), A (data valid), A (autonomous mode), *76 (checksum).

- $GPRMC (recommended minimum specific specific GNSS data): 16:55:13.00 (time UTC), A (data valid), 39º28.19045' N (lat), 0º23.07436' W (lon), 0.304 (speed over ground), 0 (course over ground), 230417 (date, ddmmyy), 0, 0 (magnetic variation), A (autonomous mode), *6E (checksum).

- $GPVTG (course over ground and ground speed): 0, T (course over ground, º true), 0, M (course over ground, º magnetic), 0.304, N (speed over ground, knots), 0.562, K (speed over ground, km/h), A (autonomous mode), *25 (checksum).

The sketch is quite simple, but still it uses a library (SoftwareSerial).

A side note on this: many of the sketches and libraries that I have seen use pins 3 and 4 for serial communication with the GPS module. However, not all Arduino boards allow those pins to be used for that purpose: check SoftwareSerial and avoid silly headaches on why the dammed GPS is not working with your Mega or Pro.

Some pages mention that Ublox modules may have timing issues with SoftwareSerial, and they recommend using the hardware UARTS on the Arduino for communication with the GPS. I have not gone beyond this.

It is possible to avoid SoftwareSerial and directly connect through the hardware serial pins RX/TX (D0 and D1 in Pro Mini). GPS_bas_00, (here) does the same as the previous sketch but directly with Serial (not with the SoftwareSerial library).

In this case the bytes read from the GPS receiver are placed into a char array and the serial print to the monitor is used only once for each NMEA sentence.

In Arduino Pro Mini the pins RX/TX D0/D1 are connected to the UART RX/TX pins, so both sets of serial pins are actually the same and cannot be used simultaneously. It is therefore necessary to disconnect (at least un-power) the GPS when loading the sketch, otherwise the board will receive input from the GPS when trying to close the communication with the Arduino IDE, and the uploading will not work. Once the sketch is loaded, the GPS can be re-connected to the Arduino (power and RX/TX on pins D0/D1) and the results from the GPS can be read through the serial monitor (RX/TX of the UART-UBS). Although in this case there are two "serial units" connected to a single serial port,

they are not interfering: the GPS is acting as the serial transmitter and the monitor as the serial receiver.

Other Arduino boards are easier on this regard, as they have several separated serial connections. See Serial or the different board specs for details.

Of course, SoftwareSerial avoids these problems.

GPS_bas_00 is quite simple, but is it possible to make it even simpler?

Here there is a project on high altitude balloons. They use Arduino and a different model of Ublox GPS module (MAX6). From them I got the idea that maybe the (almost) empty sketch might work:

```
void setup() {


Serial.begin(9600);

}
void loop()
{
}
```

As far as the GPS is powered, it just sends the NMEA sentences which it has been configured to spit out. So, if we just read from the serial port, no instruction involved, it might work.

Well, it does not work ...

The balloon project goes well beyond this. For instance, here they have prepared a sketch to dynamically configure the GPS from Arduino (which can normally be done from Ublox's u-center software, as mentioned above).

In this otherpage, also dealing with high altitude stuff, they configure the Ublox GPS from Arduino (in this case module NEO-6Q/MAX-6Q).

GPS Arduino Libraries

The main library for (this) GPS module is, as far as I have investigated, TinyGPS from Mikal Hart. The library is also in GitHub.

There is a TinyGPS++ (TinyGPSPlus) library (here the GitHub) The main differences, compared to TinyGPS and according to the author (also Mikal Hart), are that its "programmer interface is considerably simpler to use than TinyGPS, and the new library can extract arbitrary data from any of the myriad NMEA sentences".

NeoGPS is an Arduino library which is "fully-configurable Arduino library uses *minimal* RAM, PROGMEM and CPU time, requiring as few as *10 bytes of RAM*, 866 bytes of PROGMEM, and less than 1mS of CPU time per sentence.

There are libraries for specific GPS receivers. For instance, Adafruit_GPS library for the MTK33x9 chipset, is "the ultimate GPS library for the ultimate GPS module!".

GPSNEO-6M is another Arduino library for GPS interfacing, whose creator affirms has been tested with the NEO-6M.

Another library that claims to be used with "any GPS receiver that uses standard NMEA sentences" can be found here.

Maarten Lamers wrote the NMEAlibrary "for easy decoding of GPS data on the Wiring i/o board". As the points in the webpage, the Wiring was a predecessor of the Arduino, and the library has not been tested on Arduino. Still, he mentions that TinyGPS is based on this library (duly acknowledged by Mikal Hart).

UBX-GPSlibrary is designed for "fastest and simplest communication with u-blox GPS modules".

5.- Handling GPS Data

Once the GPS receiver has transmitted the NMEA sentences to the Arduino, and they have been properly read and stored in nice and "simple format", then what follows?

There are several options to convert the "simple format" data into more general format (GPX, KMZ, etc.). For instance:

- GPS Visualizer is a free online utility "that creates maps and profiles from geographic data". You can input a CSV or tabbed file, a spreadsheet, or drag and drop the data. The appearance of the page is a bit odd (so un-Apple!) but the content is good.

- GPS Prune is intended to view, edit and convert GPS data. It allows to load text files as well as NMEA files (with .nmea extension?), among quite a number of other options.

- Finally, GPS Babel seems to be the most known GPS data converter. It reads text files with NMEA sentences.

Publicado por RiGonz en 9:08

Etiquetas: Arduino, Arduino Pro Mini, GNSS, GPS, GPS6MV2, NEMA 0183, NEO 6, NEO6MV2, PRN, RX/TX, Serial, SoftwareSerial, TinyGPS, trilateration, U-blox, u-center, UBX

5 comentarios:

meghanasmiley03 25 de septiembre de 2017, 23:19

Nice post,i found lot of information.I updated my knowledge with this blog.it can help me to crack GIS jobs in Hyderabad.

Responder

RiGonz 29 de septiembre de 2017, 4:05

:-) Very glad to see that you found things worthy!

Responder

Unknown 22 de noviembre de 2018, 3:08

can i know where the tantalum capacitor connected to in the no 5??

Responder

Respuestas

RiGonz 22 de noviembre de 2018, 3:32

I'm very sorry. I deleted (by mistake) the files in Dropbox, and now I cannot recall the original source. Apologies

Responder

RiGonz 22 de noviembre de 2018, 3:31

Este comentario ha sido eliminado por el autor.

Responder

Introduce tu comentario...

Comentar como: Cuenta de Goo

Publicar Vista previa

Entrada más reciente Página principal Entrada antigua

Suscribirse a: Enviar comentarios (Atom)