



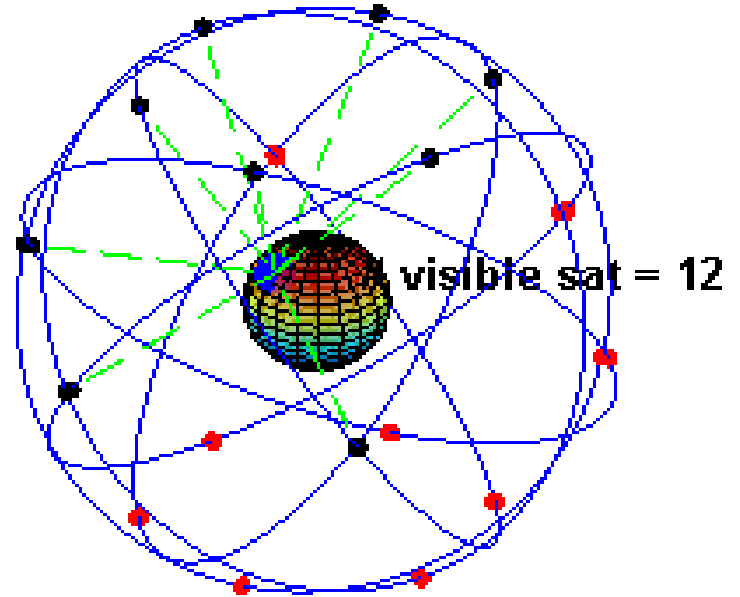
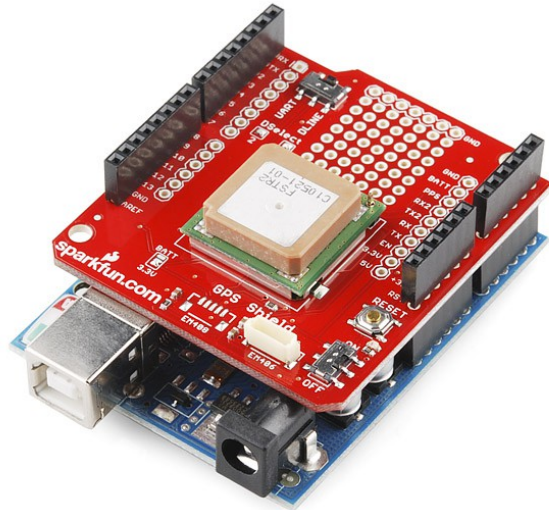
GPS - Global Positioning System

Mike Grusin

Engineer

Sparkfun Electronics

mike.grusin@sparkfun.com





Treasure hunt!

1. main sign	40.064367, -105.210019
2. e sign	40.064594, -105.209897
3. w sign	40.064596, -105.210328
4. light pole	40.065326, -105.210346
5. light pole	40.065324, -105.209903
6. light pole	40.065132, -105.209670
7. light pole	40.064975, -105.209425
8. pond pole	40.064908, -105.209527
9. tree	40.065082, -105.210384
10. tree	40.065090, -105.209854
11. tree	40.064664, -105.209868
12. tree	40.064669, -105.210390



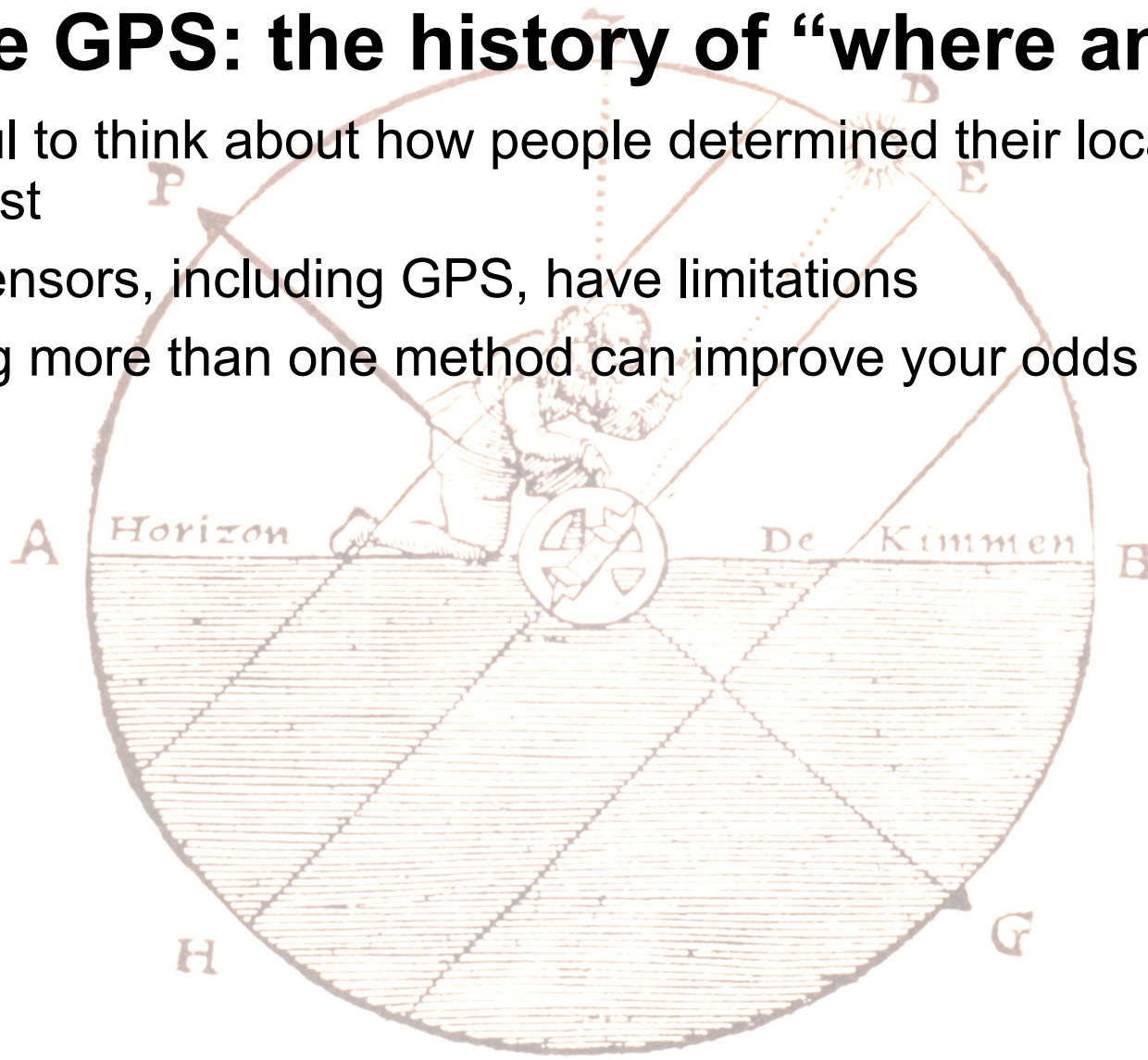
Install TinyGPSPlus

1. Go to <http://arduiniana.org/libraries/tinygpsplus/>
2. Click download, choose latest, click “source code (zip)”, save
3. Extract zip file and view contents
4. Rename e.g. “TinyGPSPlus-0.94b” to “TinyGPSPlus”
5. Drag “TinyGPSPlus” to your “Arduino/libraries” folder
6. Restart Arduino



Before GPS: the history of “where am I?”

- ▶ It's useful to think about how people determined their location in the past
 - ▶ All sensors, including GPS, have limitations
 - ▶ Using more than one method can improve your odds of success!

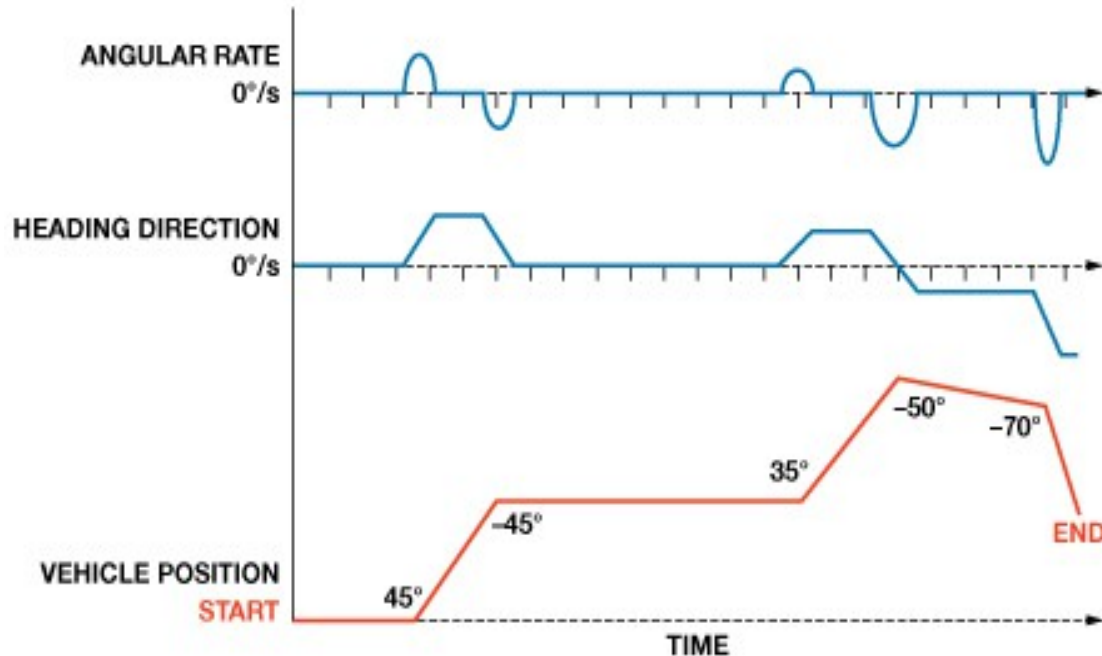




Before GPS: the history of “where am I?”

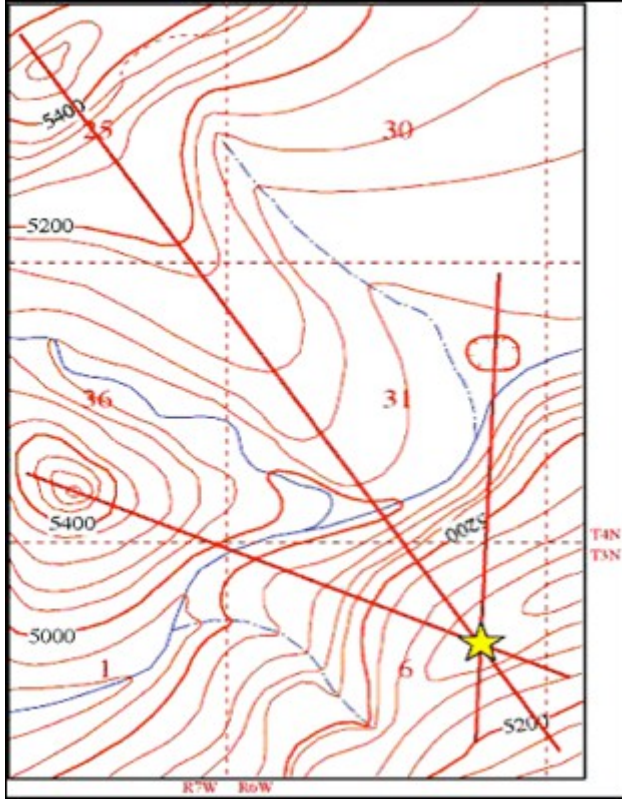
Dead Reckoning

- ▶ Keep track of how far you've gone (distance) and the angle you're traveling (azimuth)
 - ▶ Distance sensors:
 - ▶ Odometry (count wheel rotations)
 - ▶ Optical flow detectors (e.g. computer mice)
 - ▶ Angle sensors:
 - ▶ Magnetometer / compass
 - ▶ Gyro / IMU (Inertial Measurement Unit)
- ▶ Small errors will add up the longer you do this
 - ▶ Filtering is important (and complex)





Before GPS: the history of “where am I?”



Triangulation

- ▶ Measure the bearing angle (azimuth) from your (unknown) position to fixed landmarks with known positions
- ▶ Plot reciprocal angle from landmarks
- ▶ Your location is where the lines cross!
- ▶ Works well, but requires known fixed landmarks and a way to accurately measure bearing angles to them (compass), or the angle between them
- ▶ GPS uses a variant of this method!



Before GPS: the history of “where am I?”

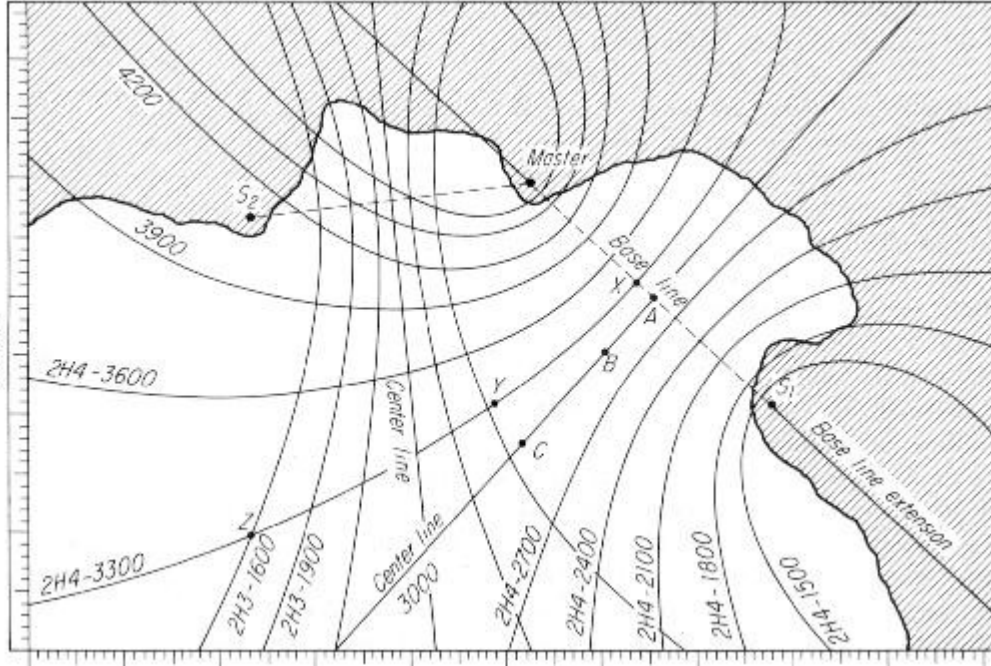
Celestial navigation

- ▶ A sextant measures precise angle (altitude) between a celestial object and the horizon
- ▶ By knowing the exact time and consulting tables, you can determine your location (to ¼ mile if skilled)
- ▶ You need the exact time (this was very difficult with mechanical clocks!) and a clear view of the sky
- ▶ Still used as a backup (salt water and electronics don't mix)





Before GPS: the history of “where am I?”

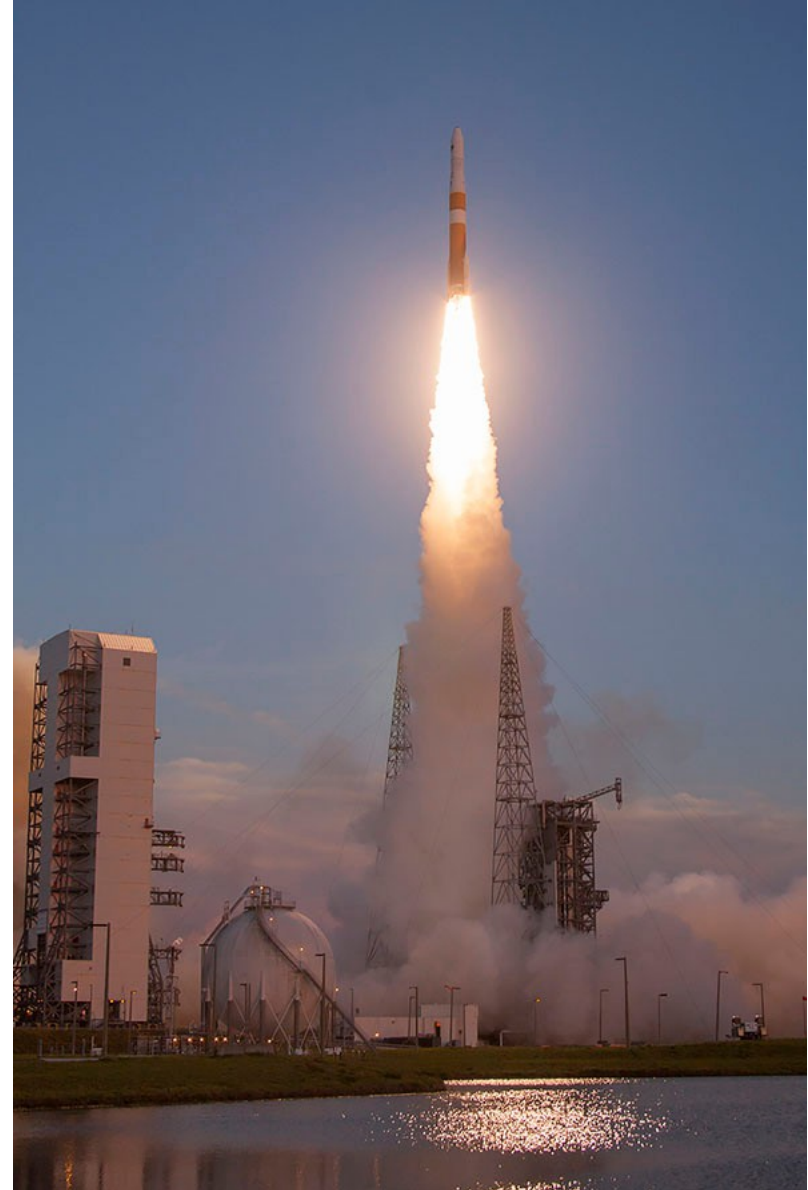


LORAN (Long Range Navigation)

- ▶ Developed in the 40s for military and eventually civilian navigation
- ▶ Used phase-linked fixed radio transmitters
- ▶ The phase difference between two transmitters locates you on a hyperbolic line
- ▶ Adding a third transmitter puts you at the intersection of two hyperbolic lines
- ▶ Accuracy of 0.1 to 0.25 miles
- ▶ Transmitters turned off and system decommissioned in 2010 in favor of GPS

GPS

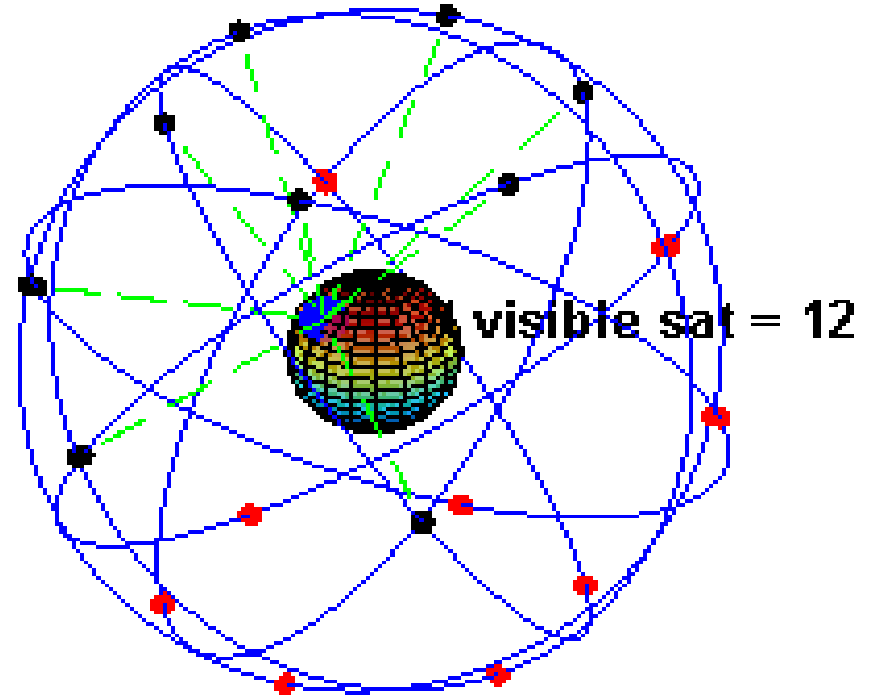
- ▶ Evolution of earlier military satellite experiments and navigation systems
- ▶ Requires 24 satellites for continuous global operation
 - ▶ First launch in 1989, fully operational in 1994
 - ▶ Launches continue today for upgrades and maintenance
- ▶ Run by USAF 50th Space Wing, based in Colorado Springs
- ▶ “One of the most beneficial and humanitarian results of military and aerospace technology”
- ▶ Other countries have their own systems: Russian GLONASS, EU Galileo, China, India





How does it work? (the simplified version)

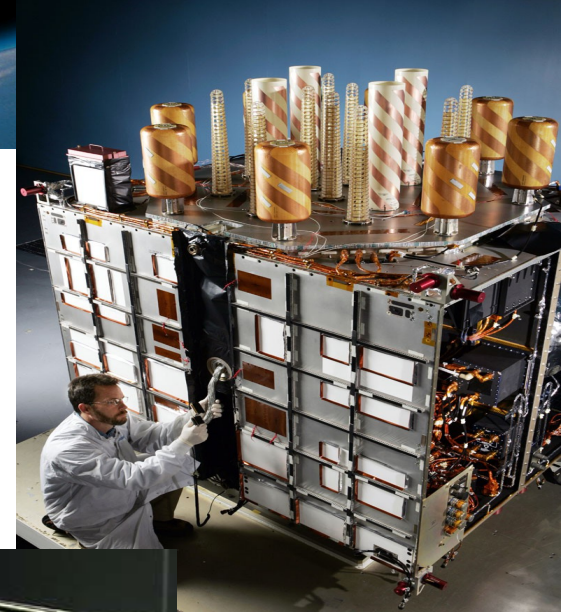
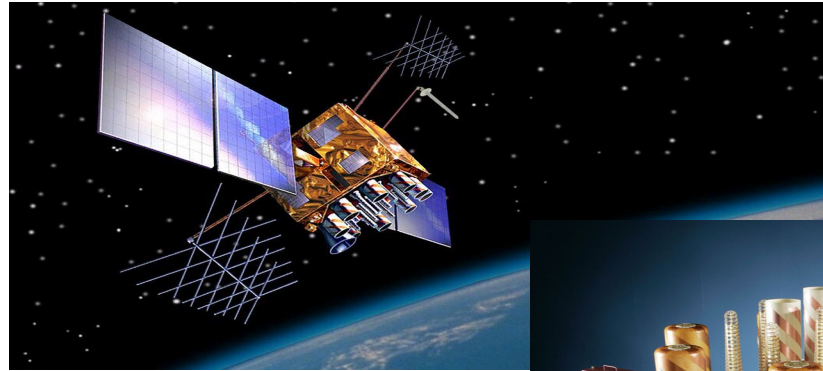
- ▶ At least 24 active satellites (plus spares and upgrades) in 6 orbital planes (at least 4 active satellites per plane)
- ▶ 12500 miles up, 12-hour orbit
- ▶ The geometry ensures that (depending on the local terrain) you can see at least four satellites from any point, and often more (8 or 9)





Atomic clocks

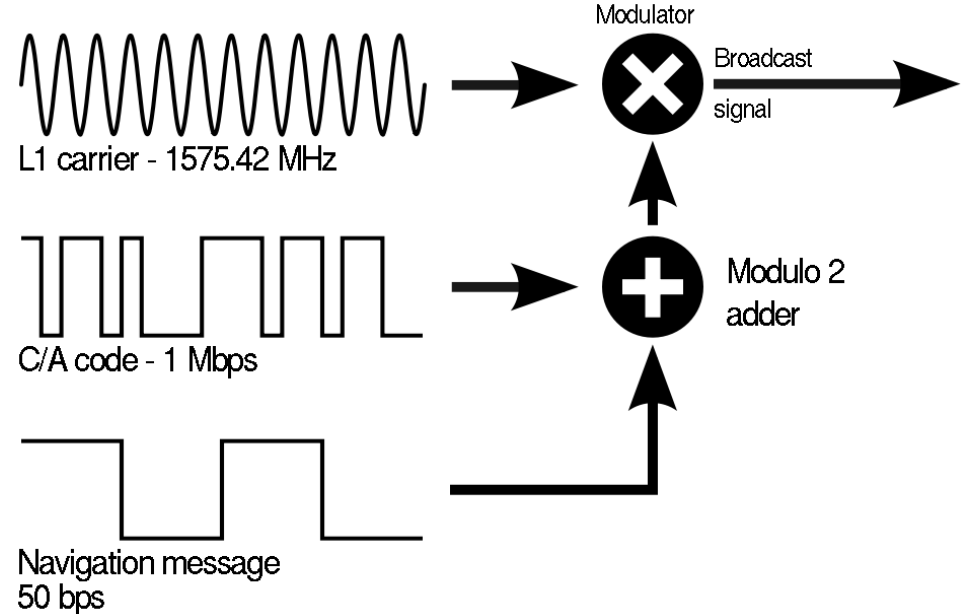
- ▶ Each GPS satellite contains four(!) atomic clocks
 - ▶ For redundancy and testing against each other
 - ▶ Accurate to better than 1ns (1 billionth of a second)
 - ▶ The clocks are designed to tick slightly slow to remove relativistic effects from the satellite's velocity
- ▶ Note that even the cheapest GPS receiver gives you access to extremely accurate time!





How does it work? (the simplified version)

- ▶ **Your receiver listens to a cacophony of signals and figures out where it is!**
- ▶ All active satellites constantly transmit precisely-timed signal patterns
 - ▶ Frequencies: 1228MHz and 1575MHz
 - ▶ Very low power: 50W
- ▶ The data contains fixed sequences unique to each satellite, plus (all) satellite orbits (ephemerides) and the atomic time
- ▶ 1500 bits, 30 seconds / frame
 - ▶ Low rate, but bit timing is extremely precise
- ▶ Note that all satellites transmit simultaneously on the same frequencies – everything gets mashed together!





“Channels”

- ▶ GPS receivers will have x “channels” (up to 60!)
- ▶ Why so many “channels” with only 24 satellites?
- ▶ These are not receivers (there are only two frequencies, so you would need at most two receivers)
- ▶ These “channels” (called “correlators”) do the mathematically-intensive job of trying to match the key patterns (different for each satellite) against the mashed-up signal being received (all the visible satellites mixed together)
- ▶ When a channel has a match, the satellite data can be

extracted and used to create or improve the current fix.

* Multiple correlators greatly speed up the initial fix, as they can all work in parallel to identify the satellites in-view.

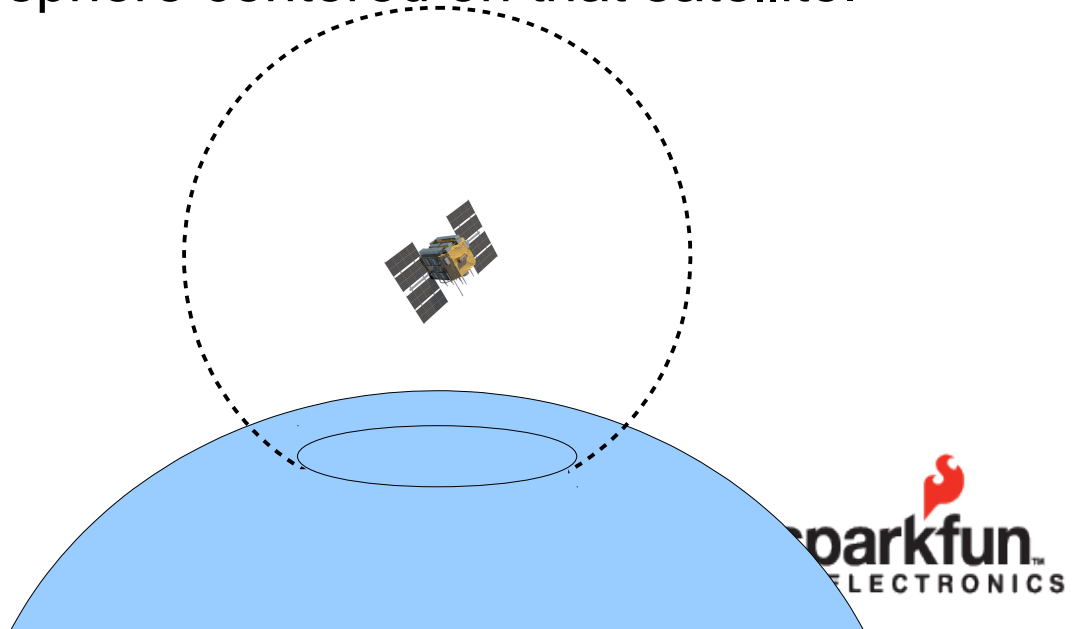
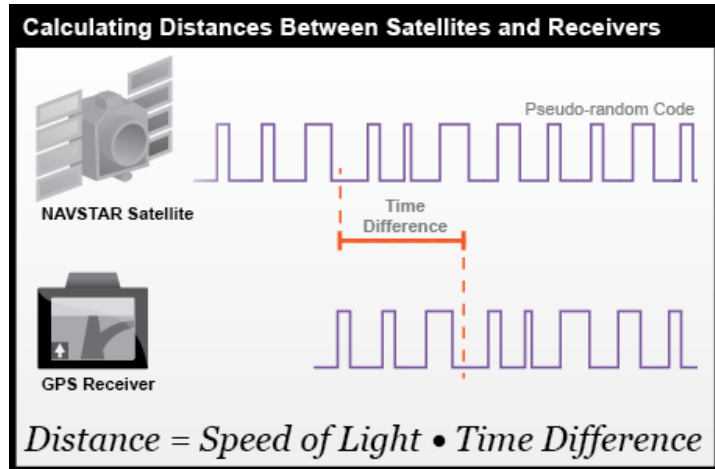
$\Sigma = 9$ $\Sigma = 25$

ELECTRA



How does it work? (the simplified version)

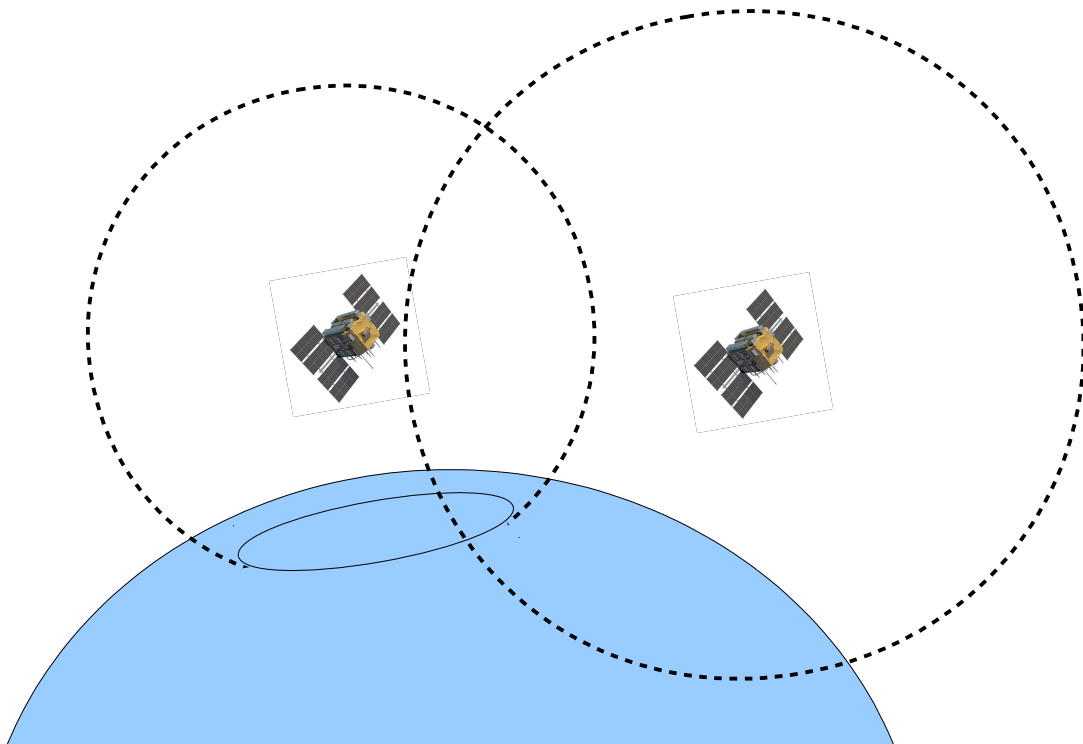
- ▶ When a satellite is found, the atomic clock time on the satellite (sent along with the data) is compared to the (possibly inaccurate) clock time in the GPS receiver, to guess the range (“pseudorange”) to the satellite.
- ▶ Calculating the range to a single satellite puts your position somewhere on the surface of a sphere centered on that satellite.





How does it work? (the simplified version)

- ▶ When you have the range to two satellites, your position will be somewhere on the circle where the two spheres intersect





How does it work? (the simplified version)

- ▶ When you have the range to three satellites, your position will be at one of two points where the three spheres intersect
- ▶ GPS modules assume you're near the surface of the earth; one of these points will usually be far off in space and can be discarded



How does it work? (the simplified version)

- ▶ When you have four (or more) satellites, you will not only improve the accuracy of the fix, but you can solve for the precise time at the GPS receiver.
- ▶ This is why and how the receiver does not need an (expensive!) atomic clock to function.



History: SA (Selective Availability)

- ▶ A military system which purposely added pseudorandom errors to the signal
- ▶ Degraded civilian GPS receivers to 100m accuracy
- ▶ Military used special receivers and a daily key to remove those errors
- ▶ Turned off during the entirety of the Gulf War (1990) because troops were buying civilian GPS receivers due to a shortage of military receivers
- ▶ The government was increasingly pressured to turn off SA to enhance civilian use of GPS (especially FAA)
- ▶ Permanently turned off in 2000
- ▶ The military has “methods” to deny GPS to hostile forces (while friendly use is unaffected)

Accuracy and error sources

- ▶ How accurate is GPS?
 - ▶ When viewing GPS on a map, you will see the fix “wander” around an area
- ▶ Error sources
 - ▶ Ionospheric effects ± 5 meters
 - ▶ Satellite orbits determination ± 2.5 meter
 - ▶ Atomic clock errors ± 2 meter
 - ▶ Multipath effects ± 1 meter
 - ▶ Tropospheric effects ± 0.5 meter
 - ▶ Calculation rounding errors ± 1 meter
- ▶ Total: ± 12 meters (40 feet radius)
- ▶ Satellite geometry at various times can also be a factor (HDOP – Horizontal Dilution of Precision is computed by the GPS module to tell you this)



Improving accuracy

- ▶ Use GPS outdoors with a clear view of the sky
 - ▶ Signals may or may not penetrate building construction, but will be degraded in all cases
- ▶ Avoid trees, rain
 - ▶ GPS frequencies are absorbed by water
- ▶ Avoid “multipath” situations
 - ▶ Urban (or rock) canyons cause signals to bounce, causing phase issues
- ▶ Use a good antenna
 - ▶ And if its directional, keep it pointed up (or it will make things worse!)



Improving accuracy

- ▶ You can buy commercial receivers that are accurate to several cm, but they are expensive (\$1000+)
- ▶ DGPS (Differential GPS)
 - ▶ Uses two GPS receivers, one at a known fixed location, the other one where you want precise position (AVC robot, etc.)
 - ▶ The fixed receiver measures the errors it's receiving (it's fixed, so any drift is error)
 - ▶ The fixed receiver transmits the errors to the moving unit. If they're fairly close, they should be seeing roughly the same ionospheric effects
 - ▶ Commercial DGPS systems can achieve accuracy of mm
 - ▶ Unfortunately this does NOT work by just putting two GPS units near each other (many people have tried). They must be using the same set of satellites at all times, which is not guaranteed



Improving accuracy

- ▶ WAAS (Wide-Area Augmentation System)
 - ▶ Less accurate than DGPS, but doesn't require additional hardware and works over the continental US (other systems available in other countries)
 - ▶ FAA measures fluctuations in the ionosphere
 - ▶ FAA sends that data to a geostationary satellite, which broadcasts that data as a “fixed” GPS satellite
 - ▶ Equipped GPS receivers can use this data to improve accuracy
 - ▶ Can improve accuracy to ± 2 meters (under ideal conditions)



ITAR restrictions

- ▶ To prevent their use in missiles, all consumer GPS units sold in the US have built-in restrictions to the maximum speed and altitude at which they'll operate
- ▶ 60,000' altitude
- ▶ 1000 knots speed (1150 mph)
- ▶ GPS modules are programmed to refuse to operate if one or both of those limits is exceeded
- ▶ “One or both” becomes important if you want to use GPS on a high-altitude balloon (to 130,000' or more), which only violates one of those limits
- ▶ Different manufacturers will interpret this rule in different ways (and this isn't often in the datasheet)

Check Sparkfun user comments and Google for real



Getting to know your GPS

- ▶ You have a GPS Shield and an EM-406 GPS module
- ▶ The EM-406 talks to your Arduino via normal serial port
- ▶ The Shield lets you route the EM-406's serial port to either:
 - ▶ The hardware port (“UART”) on the Arduino (RX and TX)
 - ▶ or two other pins (“DLINE”), preset to 2 and 3 (“SoftwareSerial”)
 - ▶ SoftwareSerial lets you talk to both the GPS and host computer simultaneously, which can be very useful
- ▶ For this class, we'll use SoftwareSerial
 - ▶ Set the “UART/DLINE” switch to “DLINE”
 - ▶ Remember that serial data will be sent to pins 2 and 3!
- ▶ Red LED on EM-406:
 - ▶ Off: powered off
 - ▶ On (steady): looking for satellites
 - ▶ On (blinking): has computed a position fix





Improving start-up time

- ▶ All GPS receivers need time to get the first fix (cold start)
- ▶ All channels are looking for any visible satellites
- ▶ Under ideal conditions this will take about a minute, but it could be many minutes (or never) if you're indoors, etc.
- ▶ You can improve start up time to seconds if you provide a back-up power source to keep the GPS receiver's internal clock running
 - ▶ This can be an extra connection to your power source, separate battery (coin cell), super-capacitor, etc.
 - ▶ Power draw is very low (microamps)
- ▶ By leaving the clock running, when the rest of the GPS unit powers up, it can immediately predict which satellites are in-view, and quickly look for those (warm start).
- ▶ Warm start performance will degrade the longer it's been



Talking to your GPS

- ▶ Almost all GPS receivers output a data format specified by (and generally called) **NMEA** (National Marine Electronics Association)
- ▶ This is ASCII (text) data, sent on a standard serial port, usually at 4800bps (you can change the speed)
- ▶ Data is sent as comma-separated values in single lines, or “sentences”
- ▶ There are about a half-dozen different sentences that have various data in them
- ▶ Example:

```
$GPRMC,161229.487,A,3723.2475,N,12158.3416,W,0.0,13.309,62,120598.00,A,0.0,M,0.0,0.0,0000.0,000.0,0000.0,000.0,0000.0,000.0
```




Talking to your GPS

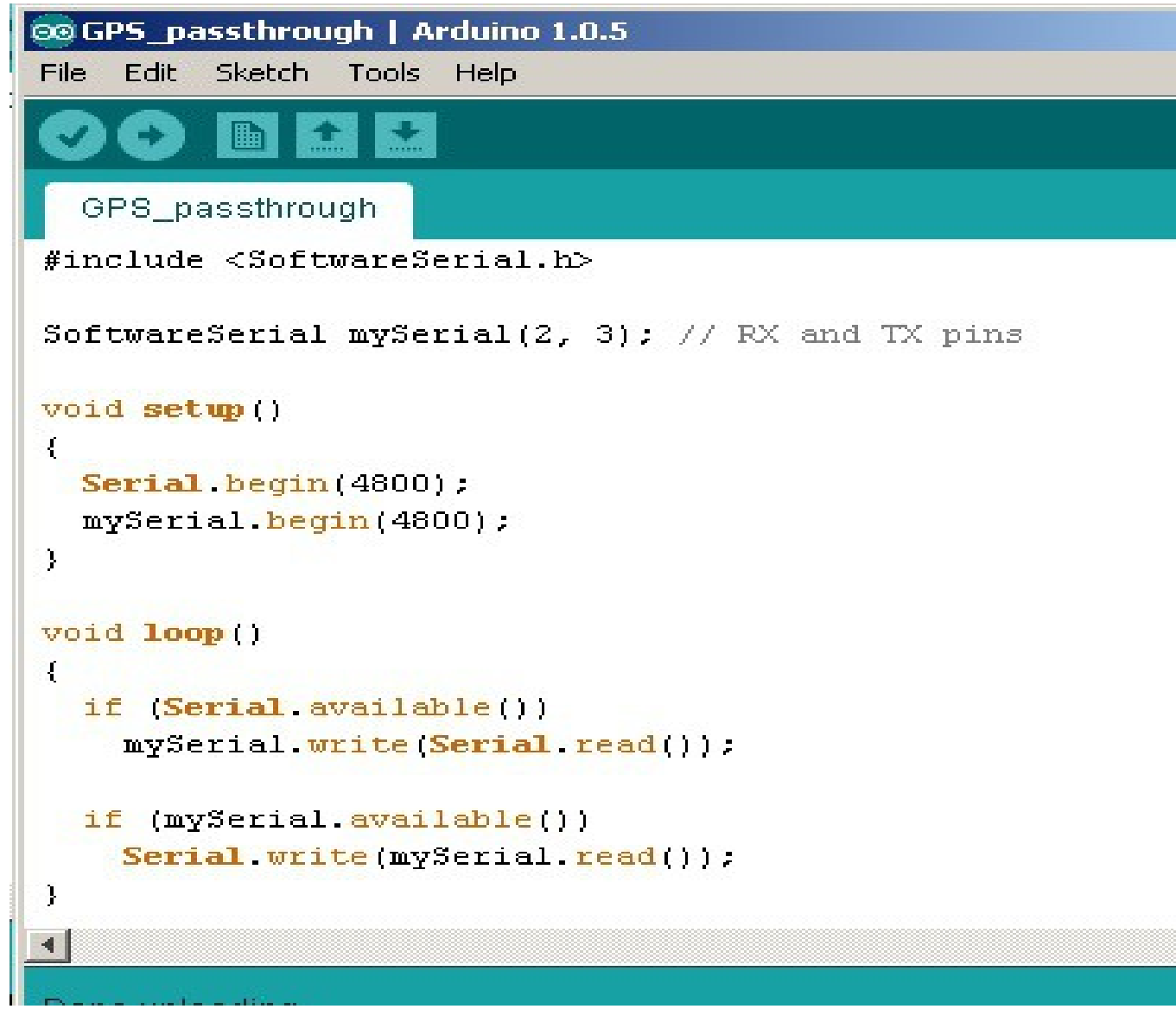
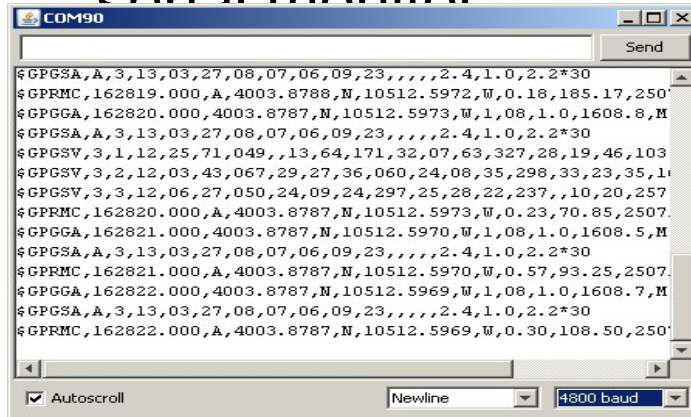
```
$GPRMC,161229.487,A,3723.2475,N,12158.3416,W,0.13,309.62,120598,,*10
```

Message:	\$GP	From GPS receiver
Sentence:	RMC	"Recommended Minimum" data
UTC Time:	161229.487	hhmmss.sss
Status:	A	A=valid or V=not valid
Latitude:	3723.2475	ddmm.mmmm
N/S:	N	N=north or S=south
Longitude:	12158.3416	dddmm.mmmm
E/W:	W	E=east or W=west
Speed:	0.13	knots
Course:	309.62	degrees true
Date	120598	ddmmyy
Mag Var:	NA	degrees E=east or W=west



Let's try it!

- ▶ Write and upload this sketch
- ▶ Open the serial monitor and set it to 4800 baud
- ▶ You should see NMEA data in the serial monitor





NMEA “sentences”

- ▶ You'll see that each line starts with a '\$' and a sentence designator “GPXXX”
- ▶ Each sentence has different data (some is duplicated in multiple sentences)
- ▶ What you see is the default set of sentences for this GPS module
- ▶ You can send commands to the GPS module to turn sentence types on and off, and control the rate at which they appear
 - ▶ For example, you can ask it to send “RMC” sentences once per second, and the “satellites in view” sentence once every 10 seconds



Saving and using raw data in Google Earth

- ▶ Start Coolterm (or the terminal software of your choice) and tell it to talk to your Arduino's serial port
- ▶ Tell Coolterm to save the data to a text file
- ▶ Take your GPS receiver somewhere interesting
- ▶ When you're done, stop saving the data
- ▶ Go to a website that can translate NMEA sentences to Google Earth .kml files
 - ▶ <http://www.h-schmidt.net/NMEA/>
 - ▶ Many others out there, Google "NMEA to kml"
- ▶ Give the site your saved NMEA file
- ▶ Load the output .kml file into Google Earth
- ▶ Profit!



Seeing live data

- ▶ Download and install the “SiRF Demo Software” available from the EM406 product page at Sparkfun.com
- ▶ Set the correct COM port and baud rate (4800)
- ▶ The software will graphically display the output from the GPS module (but does not have access to actual geographic data)



Using the data in your sketches

- ▶ You could write a sketch that digs through the NMEA data and extracts the values you need
- ▶ But as is common with Arduino, someone has done the hard work for you!
- ▶ Use the TinyGPS library by Mikal Hart
- ▶ [HTTP://arduiniana.org/libraries/tinygps](http://arduiniana.org/libraries/tinygps)
- ▶ Installation:
 - ▶ Open your personal Arduino sketch folder (on Windows this is “My Documents/Arduino”)
 - ▶ If there isn't one already, create a folder called “libraries” in the above folder. All your new libraries will go in this folder.
 - ▶ Download the TinyGPS library (TinyGPS12.zip) from the above web page (at the bottom, above the comments)
 - ▶ Unzip the file. and put the “TinyGPS” folder into the



Let's try it out!

- ▶ In the Arduino IDE, open:
File/
Examples/
TinyGPS12/
Examples/
test_with_gps_device
- ▶ IMPORTANT:
 - ▶ Change the line:
“SoftwareSerial nss(3,
4);” to the pins used by
our hardware (2, 3)
 - ▶ Upload the sketch, and
open the serial monitor at

The screenshot shows the Arduino IDE interface. The top bar indicates the file is 'test_with_gps_device' and the IDE version is 'Arduino 1.0.5'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for checking, running, and uploading. The main text area displays the code for 'test_with_gps_device.h'. The code includes `<SoftwareSerial.h>` and `<TinyGPS.h>`. A comment explains that the code demonstrates the normal use of a TinyGPS object, requiring SoftwareSerial and assuming a 4800-baud serial GPS device on pins 3(rx) and 4(tx). The code defines a `TinyGPS` object named `gps` and a `SoftwareSerial` object named `nss` with pins 2 and 3. It also defines several static functions for printing GPS data. The `setup` function is shown at the bottom. Below the code editor is the Serial Monitor window, which shows 'Done uploading.' and 'Binary sketch size: 15,830 bytes (of a 32,256 byte maximum)'. The status bar at the bottom indicates '13' and 'Arduino Uno on COM90'.

```
test_with_gps_device | Arduino 1.0.5
File Edit Sketch Tools Help

test_with_gps_device $
#include <SoftwareSerial.h>

#include <TinyGPS.h>

/* This sample code demonstrates the normal use of a TinyGPS object.
   It requires the use of SoftwareSerial, and assumes that you have
   4800-baud serial GPS device hooked up on pins 3(rx) and 4(tx).
*/

TinyGPS gps;
SoftwareSerial nss(2, 3);

static void gpstdump(TinyGPS &gps);
static bool feedgps();
static void print_float(float val, float invalid, int len, int prec)
static void print_int(unsigned long val, unsigned long invalid, int
static void print_date(TinyGPS &gps);
static void print_str(const char *str, int len);

void setup()
```

Done uploading.

Binary sketch size: 15,830 bytes (of a 32,256 byte maximum)

13 Arduino Uno on COM90

Let's try it out!

COM90

Testing TinyGPS library v. 12
by Mikal Hart

Sizeof(gpsobject) = 115

Sats	HDOP	Latitude (deg)	Longitude (deg)	Fix Age	Date	Time	Date Age	Alt (m)	Course ---	Speed from GPS	Card ----	Distance ---- to London	Course ----	Card ----	Chars RX	Sentences RX	Checks Fail
10	80	40.06470	-105.20989282		07/25/2013	16:53:49	302	1596.70	147.14	0.44	SSE	7527	40.52	NE	245	2	1
10	80	40.06470	-105.2098985		07/25/2013	16:53:51	109	1596.30	219.80	0.80	SW	7527	40.52	NE	791	5	1
10	80	40.06470	-105.20989157		07/25/2013	16:53:52	176	1596.30	51.25	0.37	NE	7527	40.52	NE	1080	8	1
10	80	40.06470	-105.20990365		07/25/2013	16:53:53	384	1595.80	44.63	0.52	NE	7527	40.52	NE	1289	10	1
10	80	40.06470	-105.20990584		07/25/2013	16:53:54	611	1595.80	25.54	0.57	NNE	7527	40.52	NE	1544	12	1
11	80	40.06470	-105.20990246		07/25/2013	16:53:56	265	1595.70	100.62	0.28	E	7527	40.52	NE	2130	16	1

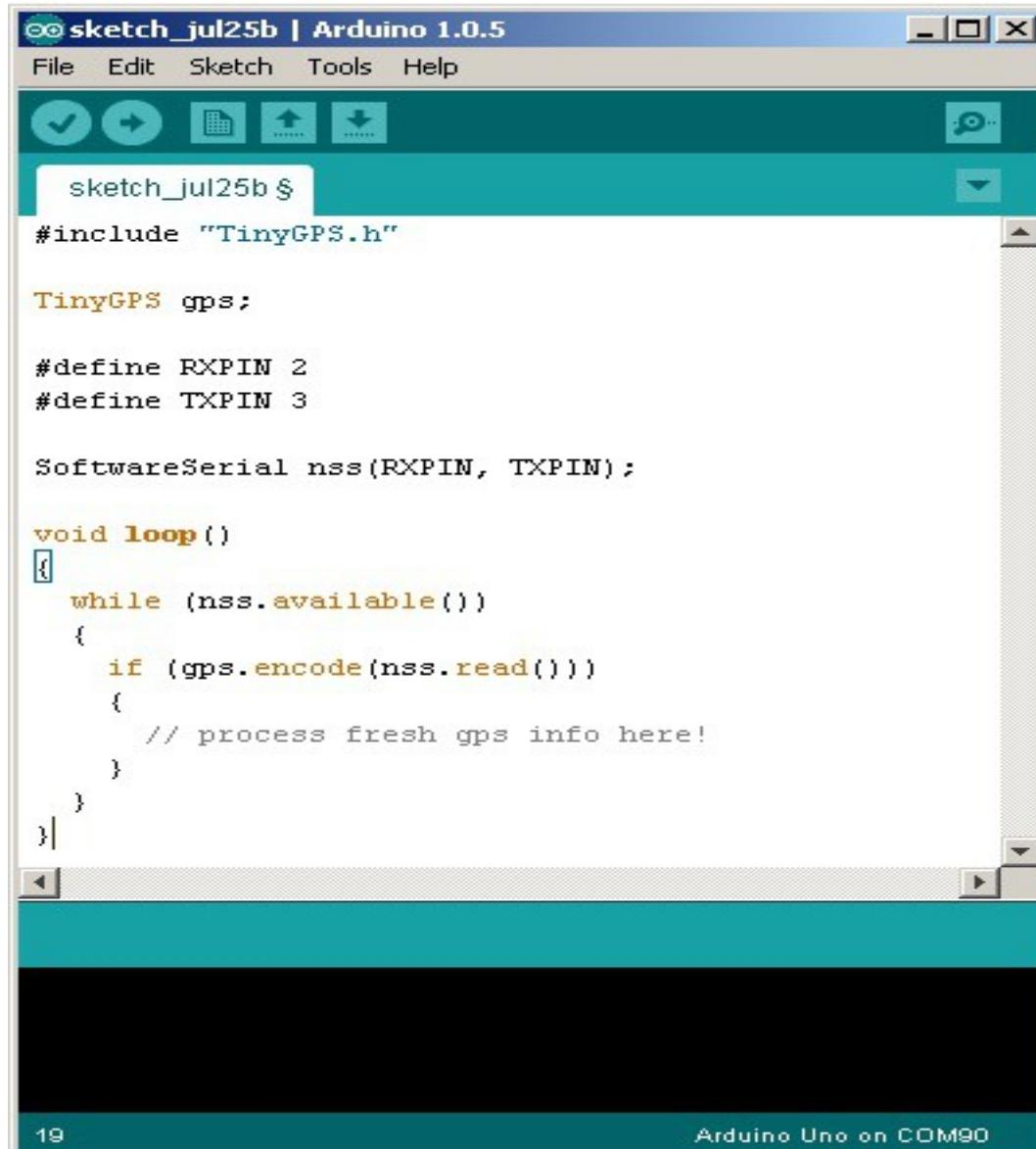
☒ Autoscroll

Newline 115200 baud



Using TinyGPS

- ▶ Very simple: when you see characters from the GPS module, feed them to `tinygps.encode(char)`
- ▶ If `tinygps.encode()` returns `true`, there's fresh position / time data available
- ▶ There are lots of commands to get out position, altitude, time, speed, course, etc.
- ▶ Plus utility functions that can determine course and distance from your position to any fixed position (great for AVC!)
- ▶ See the website for full details



```
sketch_jul25b | Arduino 1.0.5
File Edit Sketch Tools Help

sketch_jul25b $
#include "TinyGPS.h"

TinyGPS gps;

#define RXPIN 2
#define TXPIN 3

SoftwareSerial nss(RXPIN, TXPIN);

void loop()
{
  while (nss.available())
  {
    if (gps.encode(nss.read()))
    {
      // process fresh gps info here!
    }
  }
}
```

19 Arduino Uno on COM90



More information

► <http://www.kowoma.de/en/gps/>

