**EA-467 Telemetry II (GPS)** (rev c) 21 Sept 2018



##### Part H: Serial Telemetry with GPS Data

This lab shows how a device with a serial output such as a GPS can be added to your processor and the data included in the downlink telemetry. The GPS unit shown here outputs repetitive serial data at both RS232 and TTL levels at 4800 or 9600 baud that can be connected to an input pin on the Arduino. We have two models, the larger one shown here and a smaller GPS that works the same ***but has the green and white wires swapped.***

**View Raw GPS data**: First in this section ***each student*** will use the Arduino to simply echo the GPS data so you can see the raw serial data output of the GPS on your PC screen. Keep most components on the Arduino board from previous labs. All we are doing here is adding the GPS in the upper right corner and running a serial wire to the Arduion pin 12.

1. **CAUTION**: Handle the GPS module with care. **Touch** the metal case of the GPS **first** and also **touch** the metal on the LABsat before handling the GPS.
2. Connect the GPS to the board using the pin header as shown, with the black wire at the top and connected by a jumper to the Ground bus. The red GPS pin also has a short jumper to +5V also on the right.
3. The GPS will output RS232 data at on its green wire (large GPS) or the white wire if it is the small GPS). Both will output TTL level data on the yellow wire at 4800 baud. We use the yellow TTL wire.
4. The red LED on the GPS will flash while it is powered and searching for lock, and then will stay lit when locked on the larger GPS units. On the smaller units the blinking will change from a short 0.2s/1.8s on/off duty cycle while searching and then to a steady 1s/1s blink when locked.
5. Reload your working Telemetry code from before and simplify it by deleting everything in the loop function. *Rename it for this GPS section and save again*. Use the same SoftwareSerial library you used for MySerial Ouput in the previous lab but use the MySerial.read() to input the GPS data on pin 12 and display it on your PC via the USB cable. Change the ‘mySerial.begin(4800);’ in your setup() function to 4800 baud. We will again use the SoftwareSerial mySerial(12,3,false); statement so that the input is on pin 12, output is on pin 3, and TTL(false) instead of RS232(true). Then use the following statements in your Loop to read each character from the GPS (one character at a time) and display it on the PC one character at a time.

if (mySerial.available()){ // First check that a character is available

 char a = mySerial.read(); // If so, then assign it to variable a

Serial.write(a); // Then write the character to the PC
} // close if (myserial.available())

1. Observe and note that the repetitive GPS serial data on your PC is updated every 2 seconds. Some sentences are updated every cycle, and some are updated only every 5th cycle. How many different “sentences” of data do you notice in each update group? How often is the data updated? You can Click Autoscroll OFF to pause operation so you can closely observe the data.
2. Record a complete 2-second data set which should include an example of each sentence type for potential use in your report. For terrestrial applications we mostly use just the $GPRMC sentence which we can parse to find not only LAT/LONG but also Course and Speed (although they will be zero at your non-moving desk). Many APRS devices use these values to transmit position and movement for tactical situational awareness.
3. Lookup the meaning of the other sentences and briefly describe them in your report. For example, the GPGSV gives a list of the 12 satellites being tracked and their AZ/EL and signal strength.

**Part I. Parsing the GPS data:**

So far you are just displaying on the PC every character you receive from the GPS without any parsing or filtering of the data. In this section, you have to actually look at every character to find the data you want. To extract the LAT/LON data you first have to find the right data sentence ($GPRMC) and then count commas out to the beginning of the LAT field. Then skip the right number of commas and grab the Longitude field. Here are some suggestions for a large IF…ELSE statement:

1. Check every incoming byte (a) for ‘R’ then check for ’M’ and then check for ’C’.
2. Then keep getting bytes and count commas until the 3rd one.
3. Then get the next 7 bytes and increment them into a LAT[8] string array
4. Then look for next 2 commas
5. Then get the next 8 characters and increment them into a LON[9] string array.
6. Get a copy of Bruninga’s $GPRMC parsing code ***GPSpin\_parseRMC\_2017*** from blackboard and use it as the baseline for your own code for this part of the lab. Instead of implementing about 8 nested IF statements, it uses a single loop of IF…ELSE statements and keeps track of the progress of success with a flag variable F. If F gets all the way to “23” then all of the correct elements have been found.
7. Run it and verify it is parsing out the LAT and LON values into the LAT[i] and LON[i] arrays and that the data is being displayed on the PC.

This demonstrates how you can write code to parse serial data from an instrument such as the GPS here.

**Part J – Transmitting GPS data**

To demonstrate an example of how GPS data can be used, we will transmit a GPS position via the Amateur Radio APRS network (the same network supported by our PCSAT, PSAT-1, PSAT-2 and QIKCOM satellites) and see how it is plotted. So, to make this more interesting, makeup your own LAT/LON within about a mile of USNA as indicated below:

1. Go to APRS.FI and zoom into Annapolis to see the existing APRS activity in the area. Select a place you want to report your position. Record at the lat long of the cursor.
2. Change the setup for the MySerial from 4800 back to ***9600, false*** because the serial port will now be used not to read the 4800 baud TTL GPS but to transmit data at 9600 RS232 to the ground via the labsat packet radio transceiver.
3. Now, Comment out everything in the loop() function and simply insert two copies of the literal GPS formatted string in APRS format below. One using MySerial command to output to the Transmitter on pin P3 and the other using Serial command so you can also see it on your PC via the USB and serial monitor.

 Serial.println(“)PC\_ \_!3859.yyN/07629.xxWScse/spd/free text”)

The first character “)” is the APRS protocol byte indicating this is an ITEM format. The PC\_\_ is the name of the item (your PC number for example) and then the “!” begins the APRS position format with Lat/Long and the “S” is the symbol byte to show what symbol to show on the map. You can choose any other symbol character (such as Y, C, O, R, n, a, X, S, [, <, F, ‘) to represent other vehicle types. See <http://aprs.org/symbols.html> and use the letter shown in the left box.

1. Change the two underscores to name your PC such as PCB3 if you are in the B row and at the 3rd PC and insert your own hundredths of minutes in place of the yy, xx of LAT/LONG to put your transmitted position at your selected location on the map. Also insert your own 3 digit course and speed for example, to fly over Bancroft in the next few minutes, and add your name at the end
2. As you did in Telemetry I lab, now connect your Arduino Transmit data on Pin 3 to the white or yellow TXD serial data line going to your PC serial port and running PUTTY to verify the data is being sent on that line at RS232 9600 baud.
3. When data is verified. Ask the instructor for a LABSAT. Plug in the power lead from the COMM board and from the Arduino into the Battery board as shown here. Be very careful that the black wire is on the bottom.
4. Disconnect your Red Arduino board and using an appropriate full size USB connector, plug into the Arduino on the LABSAT. Use TOOLS to switch com ports when you switch Arduions like this. Download your program into the LABSAT and verify that everything is still working.
5. We cannot send such a high repetition rate onto the local shared VHF APRS channel, so you need to reduce the rate to once every 30 seconds. INSERT a **delay(30000).**
6. Once the performance and the 30 second delay has been verified by an instructor, plug the white pigtail wire from the Arduino into pin 2 of the Comm board and it should begin to transmit data every 30 seconds.
7. The LABSATs transmit on the 145.825 space frequency. To get to the APRS network, the LABSATs are set to transmit via a relay by the USNA Groundstation W3ADO and from there to the Annapolis area APRS digipeater on the gate-8 water tank and then out to the APRS network. There it should be picked up by someone’s home IGATE and fed into the APRS system. If successful, you should see your location show up on the APRS map shown at <http://aprs.fi>. Search for your PCxx and use the PC cursor to check its position and see if it is what you are transmitting. Record your observations.
8. Change your LAT/LON in your code and see yourself move. (Note, the APRS network only allows updates once every 30 seconds max since it is shared by hundreds of other Ham radio operators in the area so you will have to be patient.

**Post Lab Analysis:** Identify each of the GPS reported LAT/LON fields. Some of the fields should be obvious, while others may require you to look them up on the web. Search for NMEA $GPRMC and the other sentences. Find particularly the one that indicates the GPS satellites being used for the fix. Where are they relative to our windows? See if you can identify what information they include.

**Laboratory Report:** This is the second of three telemetry labs. You will be combining your results from these three labs as Part II of the formal Lab Report 1. Your data, results, figures, plots and analysis should lead you (in the results and discussion section) to specific conclusions regarding telemetry and communications design. Summarize your key takeaways and learning points regarding telemetry in your conclusion.