Where have we been? - GPS maps on a boat without internet

Bill Ballard

Our Thursday sailing crew wanted a way to see where we actually went from week to week. While on the San Francisco Bay we don't have constant internet or cell service, so we needed a standalone system that could log our latitude and longitude for later viewing on a map. This prevented us from using the Initial State GPS logging version as the ability to upload data files needs at least a cell phone hot spot. This solution is free except for the hardware. As an additional benefit, we display the GPS speed over ground which is different from the boat instrumentation that shows our speed through the water. This lets us know if we are fighting a significant current from the tide.

I started with a Raspberry Pi Zero W, an Ultimate GPS breakout from Adafruit, and a Pimoroni Scroll pHAT. With the pHAT I used an extended header so I could both connect and move the GPS out of the way of the display with male-to-female extension jumper wires.

Hardware Needed

- Raspberry Pi Zero W (doesn't come with 2x20 GPIO header)
- Adafruit Pi Protector for model Zero or a case of your choice that allows use of the GPIO
- Ultimate GPS breakout (not the HAT version) from Adafruit, I also recommend getting the battery
- Pimoroni Scroll pHAT or HD version if you prefer
- Headers: 2x20 pin strip dual male for the Zero W, 40-pin stacking header for the Scroll pHAT, 4 male-to-female extension jumper wires for the GPS
- 8 GB microSD card with suitable adapter for your computer (adapter is usually included)
- USB – MicroUSB cable for power
- 12V USB adapter suitable for an automobile with USB cable to power the Pi
- Suitable adapters to set up the Pi and connect a display, mouse and keyboard for setup and for running Mathematica. The Adafruit Raspberry Pi Zero Budget Pack has all the right parts including the 2x20 header.

Assembly

For the Raspberry Pi Zero W, solder on the dual male header using standard directions. After soldering the header on, you can put the Zero W in a case.

For the Pimoroni Scroll pHAT, solder on the stacking header. Then cut off the header wires EXCEPT for pins 4, 6, 8, and 10 (https://pinout.xyz/#). Double check the pin numbering before you do this! I bent these four pins 90 degrees so the GPS would be out of the way when installed. You may come up with a better

Illustration 1: Assembled Raspberry Pi Zero W with header and case.

mouting method depending on your boat and where you will install things so you can defer cutting and bending pins until later.

For the GPS breakout you need to wire the Vin, GND, TX and RX pins to the appropriate pins on the GPIO. The appropriate pins on the GPIO are on the outside of the Pi, so the GPS would cover the display unless you use some method to move it out of the way. Fortunately, these are adjacent pins 4, 6, 8, and 10 . The TX and RX pins naturally cross over from the Ultimate breakout so a direct connection can be used. (In the picture, I used another section of header to allow for some other uses of the GPS.) The GPS can now be soldered onto the 4 male pins of the jumper cables. I also recommend installing the provided optional battery connector on the back of the GPS and installing a battery, it will speed up satellite acquisition significantly when starting the system. I just push the female ends of the wire onto the 4 pins of the pHAT being careful to align them properly. The final product is held in place with some Velcro on the back of the Pi and on the console.

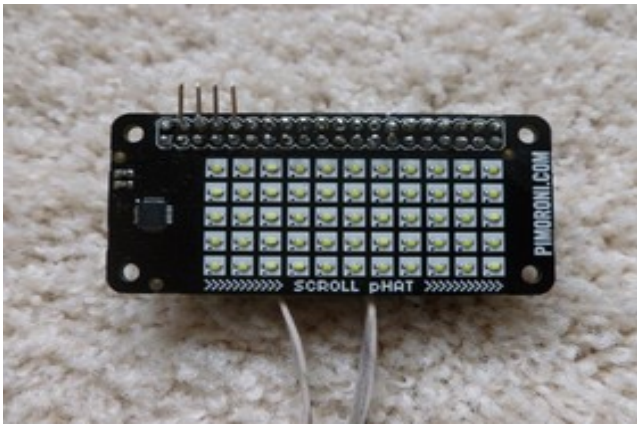

Illustration 2: Header installed on Pimoroni Scroll pHAT with unneeded pins removed.

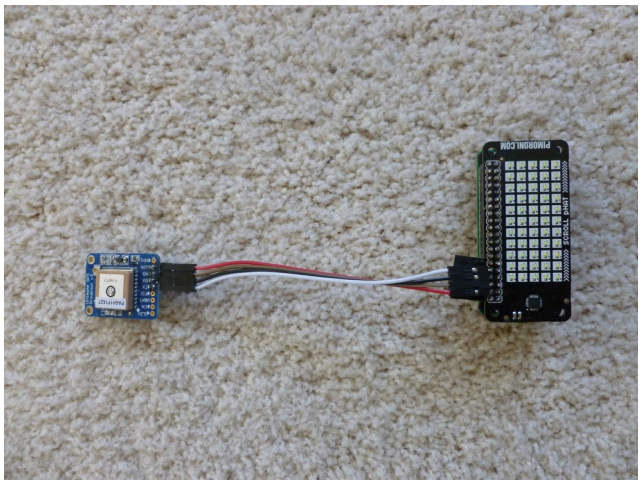

Illustration 3: Assembled Pi with GPS board

Software setup

Download the latest full Jessie distribution. Using your favorite method (Etcher, Win32Diskimager, etc.), load Raspbian onto an 8 GB or larger micro SD card, insert the card into the Raspberry Pi Zero W, attach a display and keyboard/mouse, and then boot up the Pi but don’t attach the GPS just yet. Do your usual configuration to initialize the Raspberry Pi for your locale (timezone, keyboard, Wi-Fi country). Under Interface Options be sure to enable SSH, the serial console (yes to everything) and I2C in the configuration. Also, change passwords, and allow boot to the command line (CLI). Rebooting now is helpful or you may find some keyboard items in strange places when trying to type in passwords for wireless, so:

$ sudo reboot

Next log back in and set up the wireless. My preferred way to set up wireless it to:

$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf

Change the network country if needed and add this section to the file with your appropriate information, with the quotes:

```
network={
   ssid=”your ssid”
   psk=”your password”
```

```
    key-mgmt=WPA-PSK
}
```

then control-O to write out the file and exit. You may prefer to use the GUI instead. Reboot again, log in and then do:

$ sudo apt-get update

$ sudo apt-get upgrade

to make sure your system is up to date. Now we load some needed software for the scroll pHAT and the GPS:

$ sudo apt-get install python-scrollphat gpsd gpsd-clients python-gps

The GPS uses the TX/RX pins that the console defaults to so we need to make some modifications to the system:

$ sudo nano /boot/cmdline.txt

and remove the console=serial0,115200 portion of the line. Then save the file and exit the editor. Now we need to redirect the GPS to the appropriate place. (Please note that if you are using a pi Zero with a WIFi dongle, replace the S0 with AMA0 everywhere in these instructions.)

$ sudo systemctl stop serial-getty@S0.service

$ sudo systemctl disable serial-getty@S0.service

Next shut down.

$ sudo halt

When the system stops, remove power, install the scroll pHAT and the GPS breakout, paying attention that the power line is correctly connected to pin 4 on the GPIO and power back up. The fix light on the GPS board will blink every 1 second with no fix and every 15 seconds with a fix. If you don't have a fix, move the system to a window with a good view of the southern sky (northern in the southern hemisphere) where it can see the satellites.

While you are acquiring a lock, we need to disable the standard gpsd socket for our uses so in a terminal window:

$ sudo systemctl stop gpsd.socket

$ sudo systemctl disable gpsd.socket

and set up how we want to use it.

$ sudo gpsd /dev/ttyS0 -F /var/run/gpsd.socket

$ cgps -s

That last command will test the GPS and populate a terminal screen with the current GPS data very nicely if you have a good fix. Control-C to exit cgps.

Program

Now for the python program gpsd-boat.py. Usually we will have this start at boot, but we should check everything out manually first. The system will not have a live internet connection while sailing, and so will not know the date right away. As a result, I wait for the GPS to get a fix and use the UTC date information to generate file names with the sailing date. The python program is as follows and can be obtained from https://github.com/wpballa/gps-boat:

```
#! /usr/bin/python
# Threading section written by Dan Mandle http://dan.mandle.me September 2012
```

```
# modified by Bill Ballard January 2017 for GPS tracking files
# and to add Pimoroni Scroll pHAT display of speed May 2017
# License: GPL 2.0

# load all the modules we will need

from gps import *
import datetime
import time
import random
import threading
import scrollphat

# initialize some variables

gpsd = None                     # setting the global variable
first = True                    # first pass will set filenames
tzfix = time.altzone/3600       # time zone doesn't change
scrollphat.set_brightness(128)  # bright for daylight display

# initialize thread, we don't use all data

class GpsPoller(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        global gpsd                     # bring it in scope
        gpsd = gps(mode=WATCH_ENABLE)   # starting the stream of info
        self.current_value = None
        self.running = True             # setting the thread running to true

    def run(self):
        global gpsd
        # this will continue to loop and grab EACH set of
        # gpsd info to clear the buffer and prevent overruns
        while gpsp.running:
            gpsd.next()

# the real part of the program

if __name__ == '__main__':
    gpsp = GpsPoller()          # create the thread
    try:
        gpsp.start()            # start it up
        while True:             # infinite loop

# It may take a a while to get good data, ignore until a fix is present
# and mode > 1

            if (gpsd.fix.mode) > 1:
                utc = gpsd.utc  # load the utc time

# create and open track files for today, use utc date as we don't have
# internet available to set date, correct date for your time zone

                if (first):     # first pass generate filenames

# correct date from UTC to local, grab correct characters from utc string
```

```
                    lclhrs = int(utc[11:13])-tzfix
                    lclday = int(utc[8:10])

                    if (lclhrs<0):      # might need to correct to yesterday
                        lclday=lclday-1
                    elif (lclhrs>23):   # or tomorrow
                        lclday=lclday+1

# now fix up local day string for filename

                    if (lclday<10):
                        todayname=utc[0:8]+str(0)+str(lclday)
                    else:
                        todayname=utc[0:8]+str(lclday)
# dat file for future uses has additional data
                    fname = ("/home/pi/"+todayname+"-track.dat")
# comma separated for Mathmatica or spreadsheet
                    fname2= ("/home/pi/"+todayname+"-latlon.csv")
# open as append in case we crashed and are restarting
                    trk = open(fname, 'a')
                    trk.write("date/time, latitude, longitude,"
                    " altitude, speed, track\n")
                    trk.close()
                    trk = open(fname2, 'a')
                    trk.write("latitude, longitude\n")
                    trk.close()
# done with first pass setup
                    first = False

# now load the data and convert some units

                lat = gpsd.fix.latitude
                lgt = gpsd.fix.longitude

# convert from meters to feet for this use

                alt = 3.281*gpsd.fix.altitude

# convert speed in m/s to knots as nautical use here

                speedkts = 1.944*gpsd.fix.speed

# display speed over ground on scroll pHAT fixed for speeds < 10 kts
# Speeds over 10 kts need to scroll the display. The initial offset of 10
# and extra 7 spaces in the string will allow a complete scroll.

                scrollphat.clear()
                kts = ("{0:0.1f}").format(speedkts)
                if (speedkts < 10.):
                    scrollphat.write_string(kts, 1)
                    scrollphat.update()
                    time.sleep(10)      # set to delta seconds between grabs

                else:
                    scrollphat.write_string(kts+"       ", 10)
                    len = scrollphat.buffer_len()
```

```
                    for i in range(len/2+1):
                        scrollphat.scroll()
                        time.sleep(0.3) #sets scrolling speed
                                        # no delay as scrolling takes time
                track = gpsd.fix.track

# open in append mode and write data, then close files so a system
# crash won't prevent output

# but first correct time to right time zone

                lclhrs = int(utc[11:13])-tzfix
                if (lclhrs<0): lclhrs=lclhrs+24
                lcltm = str(lclhrs)+utc[13:19]

# build formatted output string
# for data file

                outs = ("{0:s}, {1:0.9f}, {2:0.9f}, {3:0.1f}, "
                "{4:0.1f}, {5:0.0f}\n"
                ).format(lcltm, lat, lgt, alt, speedkts, track)
                trk = open(fname, 'a')
                trk.write(outs)
                trk.close()

# for comma separated file

                outs = ("{0:0.9f}, {1:0.9f}\n").format(lat, lgt)
                trk = open(fname2, 'a')
                trk.write(outs)
                trk.close()
            else:
                print "no fix"
                time.sleep(20)  # give it some time to acquire fix

    except (KeyboardInterrupt, SystemError, SystemExit): #reasons to stop
        gpsp.running = False
        gpsp.join()             # wait for the thread to finish
        scrollphat.clear()      # clear the display

# end of program
```

To test the program, do

$ python gpsd-boat.py

You should get one “no fix” line because things don’t start right up on the first pass. As set up, it takes a minimum of 30 seconds for the files to be created even if you have a GPS lock at the start. After a minute or two do a Control-C to end the program, and ls to see the files. You should now have both files like

2017-04-16.dat

2017-04-16.csv

but with the current day's date in them and they should contain some data. The .dat file contains a lot of data you might find useful, but the .csv only has the longitude and latitude data for the trip. Both files log data every 10 seconds.

Automating things

Now we want to automate things so that much of this occurs at boot.

$ sudo nano /etc/rc.local

and add the following two lines to the file just before the exit 0 line

gpsd /dev/ttyS0 -F /var/run/gpsd.socket
python /home/pi/gpsd-boat.py > /home/pi/gpsd-boat.log 2> /home/pi/gpsd-boat.err &

Control-O to save the file and Control-X to exit. The last line will start the python program as a background job, redirect output to a log file and redirect any error messages to an error file for later debugging.

Now you need to go get some sailing, or perhaps driving data. I use a 12V automotive plug in the boat (ok, technically a yacht as it has sleeping quarters) in one of the many 12V outlets. Above deck these will likely be corroded so be prepared to clean the contacts (Scotch Brite works well) and keep the Raspberry Pi and GPS in a plastic bag to ward off water (I lost one GPS to some unfortunate splashing). Just power things up and go for a sail. When done, just unplug the system and take it home.

Illustration 4: As installed and operational

When you get home and again have internet access, connect the Raspberry Pi to a monitor and keyboard so you can use Mathematica. The .csv output file is designed to be easily read with Mathematica. However, given that we crashed the Pi to power it off on the boat, you will need to edit the file with nano and remove the last line or two. Also if the application crashed and restarted at any point there will be extra headers you should search for (latitude search term will do it) and delete. These files were written as root, so typically you need to do:

$sudo nano 2017-05-04-latlon.csv

to remove these errant lines. Then save the file and exit.

Start Mathematica (the red starburst in the menu bar) and use the following notebook to visualize your sail, but replace *date* with the date identifier for your file. You can download the notebook from the earlier github project. The first line of the file imports the data from the comma delimited file and loads the header and data separately. The output of this command should be a partial list of all the latitude and longitude pairs. The second line converts the latitude-longitude data into a GeoPosition set of variables, and then a GeoPath construct for plotting. The output of this should be a small graph of the path taken, but with no map. The third line places the GeoPath on an automatically sized map and dumps the output to the file image.jpeg. You will also get a small version of this image in the output. The PlotStyle controls the color and thickness of the sailing path plot.

```
{hdr, data} = TakeDrop[Import["date-latlon.csv"], 1]

datap = GeoPath[GeoPosition[{data}]]
```

```
Export["image.jpeg",GeoListPlot[{datap}, ImageSize → Large, PlotStyle →
Directive[Red, Thickness[0.005]]]]
```

Hit Shift-Enter to force an evaluation and wait a bit (the header will show running), particularly if it was a long sail. It takes quite some time for Mathematica to load the map data over the internet, so be patient. If this step is too long for your taste, you can ship the csv file over to a Raspberry Pi Model 3B and get much quicker execution. In any caseyou should wind up with a sample of the data, a small graph of the path, and then a file named image.jpeg with a map of the sailing area with a red path line. The final result looks like this for our sail around the San Francisco Bay. You can see how many ferry routes we dodged.

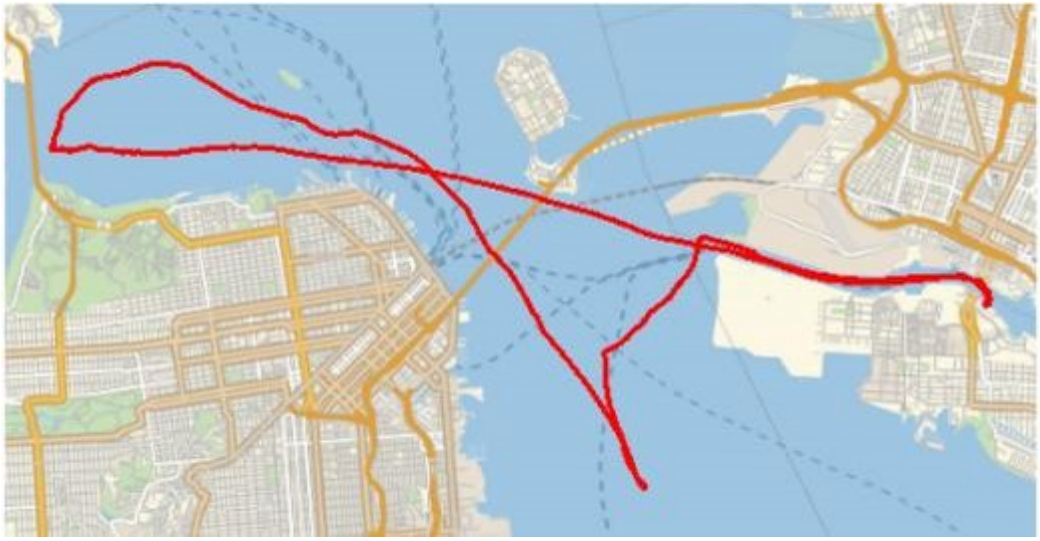

Illustration 5: Our route around the San Francisco Bay

If desired, you can omit the Pimoroni display and only log the data path. Just comment out the section using the Pimoroni scroll pHAT and use a short section of header to mount the GPS. I’m sure you can also use this for walking or cycling if you use a battery pack to power it.