Yinquan Hao

CSE 523/524

Prof. Jennifer Wong-Ma

Brief Idea

In smart transit project, we use raspberry pi 2 as GPS clients for school buses. The GPS clients are response for sending GPS packet to the smart transit server, which is written in python twisted. The packet should be follow Arcom GPS packet format. I would cover it in this write up later.

This setups the Rasperry PI to send GPS coordinates.

This setups a cron job to load the client.py upon bootup.

Waiting bc of GPS – why this necessary?

**Hardware**

Raspberry pi 2:

<https://www.adafruit.com/products/2358>

Adafruit Ultimate GPS HAT for Raspberry Pi A+/B+/Pi 2 (not for pi 3):

<https://www.adafruit.com/products/2324>

SMA to uFL/u.FL/IPX/IPEX RF Adapter Cable:

<https://www.adafruit.com/products/851>

SD/MicroSD Memory Card (8 GB SDHC)

<https://www.adafruit.com/products/1294>

MiFi® 4G LTE Global USB Modem U620L

https://www.verizonwireless.com/internet-devices/verizon-mifi-4g-lte-global-usb-modem-u620l

Additional soldering job:

For the GPS HAT, you need sold 2x20 GPIO header to the GPS HAT

**Install operation system for raspberry pi 2**

Step One:

Download NOOBS from <https://www.raspberrypi.org/downloads/noobs/>

Step Two:

Unzip the NOOBS file and paste it to the SD card.

Step Three:

Insert the SD card in to raspberry pi 2, and plug in the power cable.

Step Four:

Choose the operation system to install (Raspbian), and wait until the installing is complete.

**Configure the MIFI 4G Dongle**

The MIFI Dongle would not work out of box with raspberry pi 2. You need to construct an udev rule that will automatically reconfigure the device to use cdc ethernet mode whenever it is connected to the Raspberry Pi.

Step One:

Create a new file:

sudo nano /etc/udev/rules.d/99-vzw\_u620l.rules

Step Two:

Add following attributes to the file:

SUBSYSTEM=="usb", ATTR{idVendor}=="1410", ATTR{idProduct}=="9020", ATTR{bConfigurationValue}=="1", ATTR{bConfigurationValue}="2"

Step Three:

Save the file (ctrl-x, "y", enter)  
  
Step Four:

Reload the udev rules:

sudo udevadm control --reload  
  
Step Five:

Reboot the Pi

**Pi GPS Set Up**

Step One: Edit /boot/cmdline.txt

sudo nano /boot/cmdline.txt

Step two:

Remove console=serial10,115200 in cmdline.txt

Step Three:

Run the following two commands to stop and disable the tty service:

sudo systemctl stop serial-getty@ttyAMA0.service

sudo systemctl disable [serial-getty@ttyAMA0.service](mailto:serial-getty@ttyAMA0.service)

Step Four:

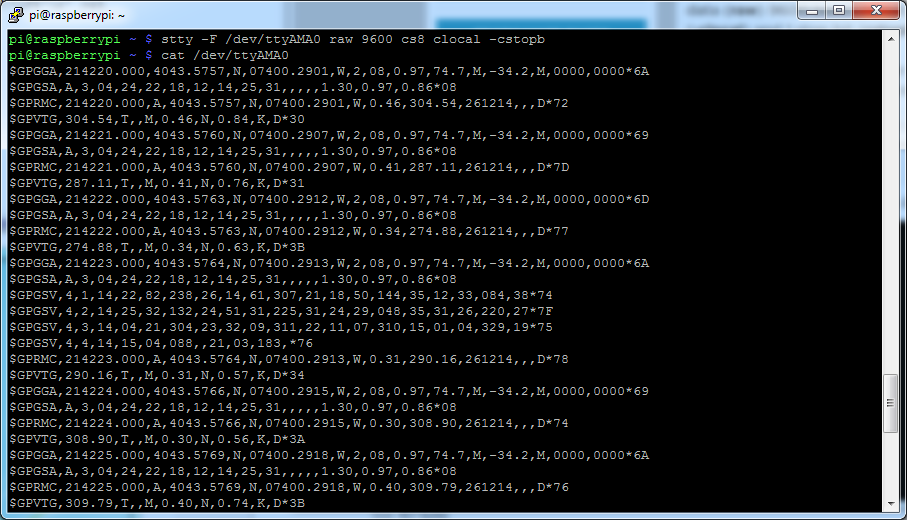
Reboot your Pi

Step Five: Test the GPS HAT

Run:

cat /dev/ttyAMA0

You should see something like this!



**Install GPSD**

For the GPS client created for the Raspberry Pi it is necessary to install a GPSD daemon which will run in the background of a UNIX system. This daemon will be polling the GPS interface on the Raspberry Pi hat and will translate the information into an easy to query format, which can then be massaged as necessary before being sent over the network.

To install gpsd, make sure your Pi has an Internet connection and run the following commands from the console:

sudo apt-get update

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

Note if you're using the Raspbian Jessie or later release you'll need to disable a systemd service that gpsd installs.  This service has systemd listen on a local socket and run gpsd when clients connect to it, however it will also interfere with other gpsd instances that are manually run (like in this guide).  You will need to disable the gpsd systemd service by running the following commands:

sudo systemctl stop gpsd.socket

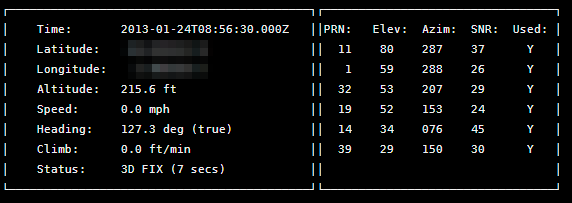
sudo systemctl disable gpsd.socket

Test gpsd:

sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock

cgps –s

You should see something like this!



Note:

Don't forget, you do need to have FIX to use these tools! If you have FIX and cgps always displays 'NO FIX' under status and then aborts after a few seconds, you may need to restart the gpsd service. You can do that via the following commands:

sudo killall gpsd

sudo gpsd /dev/ttyAMA0 -F /var/run/gpsd.sock

**GPS client python implementation**

The GPS client consist with a single python file (client.py) and a configuration file (config.cfg).

config.cfg

The configuration file used to set the server address, port and the BUSID. If you want to modify the settings, just change the values after the parameters.

Inside the config.cfg:

[config]

ADDRESS = 130.245.186.15

PORT = 8503

BUSID = 0000000099

**Client.py**

The GPS client reads the smart transit address and port from the configuration file. Then it establishes a socket connection to smart transit server. After the socket has been established, it polls GPS data from the GPSD daemon and packet it in Arcom format. Finally, send a UDP packet to the server.

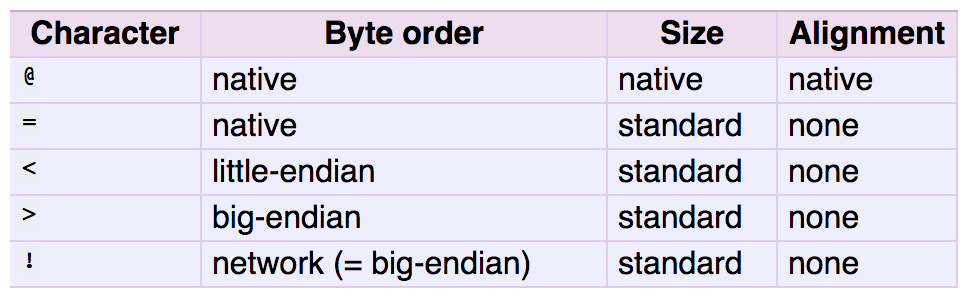
A few Notes:

Line 15: configFilePath = r'/home/pi/Desktop/config.cfg'

This intakes the directory of your configuration file. Be sure you have placed the config.cfg in this directory. If you want to change the direction you place the configuration file, please uses the absolute path. Because of the python script would automatic execute when pi start up, the term variable may not been set.

Line 121: packetData = struct.pack('!b10cBBiBbiibBh', 126,ID[0],ID[1],ID[2],ID[3],ID[4],ID[5],ID[6],ID[7],ID[8],ID[9],1,sval,time.time(),stat,1,convert(latitude),convert(longitude),71,1,0)

We use it to pack the data into Acrom format data packet. The struct.pack takes a string of format byte code format. In our case, '!b10cBBiBbiibBh'. And the method would case the input to the byte code format given.





**Shell Script & Crontab**

I have included a Shell Script in the file, which is used for setting the GPS service and executing the python script. You will need to add a Crontab job, in order to execute the shell script each time the pi startup.

Step One:

Go to the directory where you place the Shell Script:

cd /home/pi/Desktop

Step Two:

Make the launcher script an executable:

chmod 755 launcher.sh

Step Three:

Open Crontab file:

sudo crontab -e

Step Four:

Enter: @reboot sh /home/pi/Desktop/launcher.sh >/home/pi/ Desktop /cronlog 2>&1

This would create a cronlog file, which records anything supposes to show up normally in STDOUT.