**GPS Installation Instructions**

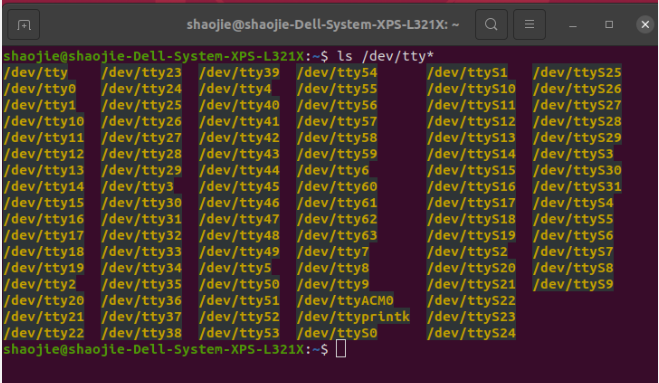
These installation instructions are a simplified version of the following tutorial:

<https://gpswebshop.com/blogs/tech-support-by-os-linux/how-to-connect-an-usb-gps-receiver-with-a-linux-computer>

Please note that all Linux terminal commands are to be run with the $ character omitted.

1. You should plug the GPS into one of the USB ports of the Raspberry Pi, and ssh into the Pi.
2. Now run the following command

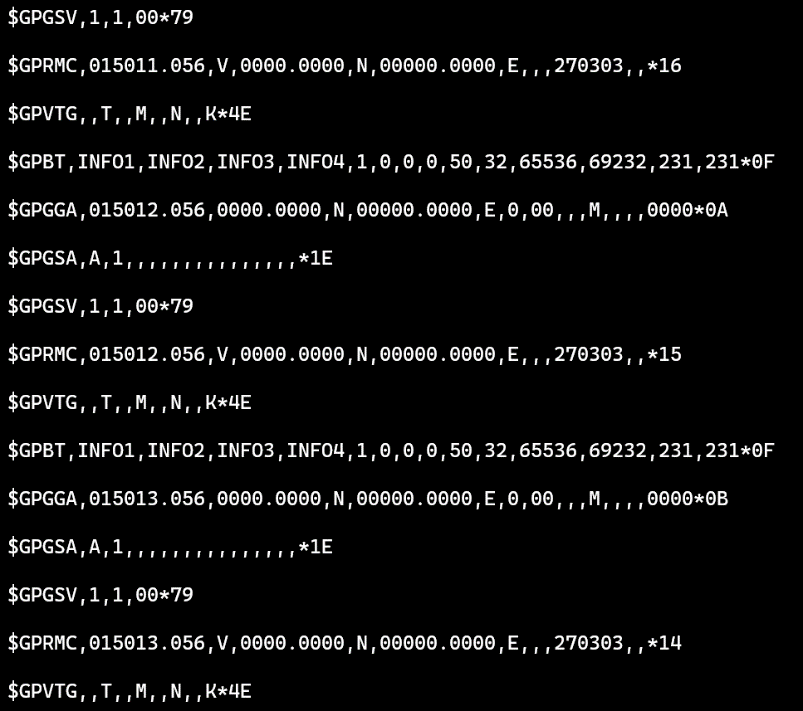
$ls /dev/tty\*



You should see a list similar to the one above. Of interest is the entry “/dev/ttyACM0”, which is the GPS.

1. To test that this is in fact the GPS, and working as intended, run the command:

$sudo cat /dev/ttyACM0

You should see the terminal print data which looks similar to the following image:

Use the keypresses “ctr + c” to stop printing.

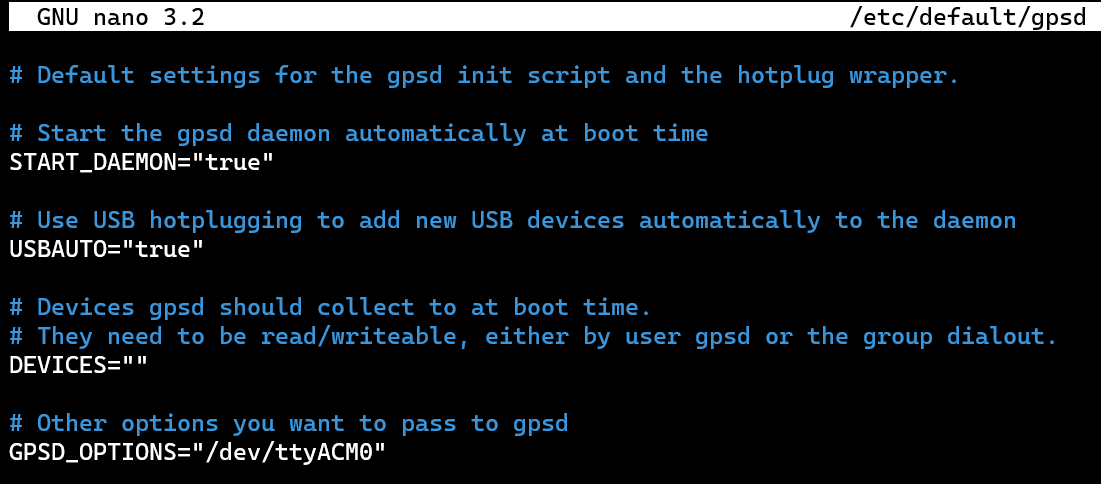
If this is not working, then there is some sort of issue, and your raspberry pi is unable to interface with your GPS module. One possible cause for this is that the baud rate is not setup correctly on the Pi. More troubleshooting instructions can be found on the linked tutorial, or you can contact us for assistance.

1. The GPSD daemon now needs to be installed. This is what your program will use to obtain GPS coordinates during its execution. The daemon can be installed with the following command:

$sudo apt install gpsd

1. The GPSD daemon needs to be made available from boot. Use the following command to enter the text editor and modify the gpsd file:

$sudo nano /etc/default/gpsd

You will see something similar to this:

You should modify the lines in the file so that they look identical to the image above. Please note the # are comment lines, these are not important and do not need to be modified. Only the lines in white need to match.

Use “ctr + s” to save your changes, and then “ctr + x” to exit the text editor.

1. Now that the setup part is complete, we can move onto writing the actual software. A library has been made for your convenience. This library contains two files, the c file and the header file. These should both be placed in the same directory (it would be good practice to place them in some “Libraries” folder). The library then has to be compiled into an object file so it can be used.

Ensure that the terminal is inside the directory which the library files are located at, and then use the following command to compile it:

$gcc -c -g gps\_library.c

To understand how to use the library, we will also share the following demo code and go over how it works.

//Written by Tiziano Wehrli

//With library, compile with:

//gccarm -g -o gps\_demo.out gps\_demo.c -lgps -lm ../Libaries/gps\_library.o

#include "eyebot.h"

#include <gps.h>

#define SECOND 1000000

int main() {

    struct gps\_data\_t gps\_data;

    int initReturn = initializeGPS(&gps\_data);

    if (initReturn != 0) {

        return 1;

    }

    //Init the GPS data variables

    float latitude;

    float longitude;

    float speed;

    printf("Running main loop now");

    //Main Run Time Loop

    while (true) {

        //Quickly poll to check if we have GPS data, 0.001ms max wait time

        if (isDataGPS(&gps\_data)) {

            //Data is available

            printf("GPS data available for us to read\n");

            int readReturn = readFromGPS(&gps\_data, &latitude, &longitude, &speed);

            if (readReturn == 0) {

                printf("Latitude: \t %.6f \n Longitude: \t %.6f \n Speed: \t %.6f \n ", latitude, longitude, speed);

            }

        }

        printf("Dropped into main loop, doing something for 0.3 seconds \n \n");

        usleep(0.3 \* SECOND);

    }

    // When you are done...

    (void)gps\_stream(&gps\_data, WATCH\_DISABLE, NULL);

    (void)gps\_close(&gps\_data);

    return 0;

}

**Code Analysis**

int initReturn = initializeGPS(&gps\_data);

    if (initReturn != 0) {

        return 1;

    }

This code segment here deals with the initialisation of the GPS. In the case that it fails, you should attempt to run the program a couple more times. If it fails every time, then there is an issue with your GPS, and you will need to troubleshoot it.

 //Init the GPS data variables

    float latitude;

    float longitude;

    float speed;

This simply initialises some variables which will hold the values the GPS returns. They will be important later in the program.

//Main Run Time Loop

    while (true) {

        //Quickly poll to check if we have GPS data, 0.001ms max wait time

        if (isDataGPS(&gps\_data)) {

            //Data is available

            printf("GPS data available for us to read\n");

            int readReturn = readFromGPS(&gps\_data, &latitude, &longitude, &speed);

            if (readReturn == 0) {

                printf("Latitude: \t %.6f \n Longitude: \t %.6f \n Speed: \t %.6f \n ", latitude, longitude, speed);

            }

        }

        printf("Dropped into main loop, doing something for 0.3 seconds \n \n");

        usleep(0.3 \* SECOND);

    }

This is the main loop of the demo code, and the part that is of most interest, as this is what you will modify when you are creating your program. The GPS returns data in intervals, and so it is poor practice to constantly wait for a GPS reading. You should instead check if GPS data is available, and if not, do something else and check again later (which is what this demo program does).

This program will loop indefinitely inside the “while (true)”.

The first if statement checks if there is GPS data available to read. If there is, it will then read the GPS data. The readFromGPS function uses pointers as parameters to update the 3 variables that were initialized previously. It is not important to understand how pointers work, just note that if the read function returns 0 (which means success), then the values of the variables “longitude”, “latitude” and “speed” have been updated with the current readings.

When you modify the program, you should delete the print line which prints out the 3 values, and instead replace this with whatever action your program should take whenever it obtains a GPS reading (for example, update the heading it should drive at, or check if it has reached its destination).

If the first if statement fails, this means that there is no GPS data available to read, and the program will simply continue on. This is when you should have your program do other things, such as check the IMU sensor, or run some sort of control loop. In this demo example, the program will simply print a line to the terminal and sleep for 0.3 seconds. When you modify the code, you should delete these lines, and replace them with whatever functionality you need.

// When you are done...

    (void)gps\_stream(&gps\_data, WATCH\_DISABLE, NULL);

    (void)gps\_close(&gps\_data);

    return 0;

These lines here should be called once just before the program finishes. They are the “cleanup” instructions for the GPS and should be run before terminating the program.

1. The last step is to compile the program so that it can be run. It would be a good idea to compile the test program without modifying it first, just to see that everything is working.

The compile instructions have been written at the top of the demo program, and are as follows:

gccarm -g -o gps\_demo.out gps\_demo.c -lgps -lm ../Libaries/gps\_library.o

The first term is the compiler command, the second and third term are compiler flags, the fourth term is the name of the compiled program (you can change this as you see fit), the fifth term is the name of the program you are compiling, the sixth and seventh term are important, as they are required to link the library we have created with the GPSD library.

Take close note of the eighth term, which is the final one. This is the path to the gps\_library object file you created previously. This path is the relative path from your current directory to wherever you have placed gps\_library object file. It is important that you correctly determine this path, or the compile will fail.

Furthermore, if you rename the demo file, or create a new one, you will need to change the fifth term (which reads “gps\_demo.c”) to reflect the new name of the code you are attempting to compile.

You can then run the program using:

$./gps\_demo.out

Please note that you may not receive any readings initially, as the GPS needs to receive a “fix”. After around 3-6 seconds, you should see the program working, where it will spend most of its time dropped in the main loop, and returning readings from the GPS more infrequently (whenever the GPS updates).

**Documentation**

The documentation of the functions we provide in our library are listed as follows. They can also be found within the gps\_library.c file.

**To initialize the GPS:**

/\*

    Function to initalize the GPS.

    @PARAM inStructGPS The address of the GPS struct.

    @return Exit status, 0 if successful.

\*/

int initializeGPS(struct gps\_data\_t\* inStructGPS) {

**To read from GPS:**

/\*

    Call this function when there is data available and you wish to read from GPS.

    Note, this will not check if data is available, and attempt to read regardless.

    @param inSructGPS The address of the GPS struct.

    @param inLatitude The address of the latitude float to modify.

    @param inLongitude The address of the longitude float to modify.

    @param inSpeed The address of the speed float to modify.

    @return

        0 on success.

        1 on gps\_read failure.

        2 on obtain mode failure.

\*/

extern int readFromGPS(struct gps\_data\_t\* inStructGPS, float\* inLatitude, float\* inLongitude, float\* inSpeed) {

**To check if there is data available:**

/\*

    Call this function to check if there is any data to read from the GPS.

    @param inSructGPS The address of the GPS struct.

    @return True if there is data to read, false otherwise.

\*/

extern bool isDataGPS(struct gps\_data\_t\* inStructGPS) {