POLITECNICO DI TORINO

Collegio di Ingegneria Informatica, del cinema e meccatronica

Corso di Laurea Magistrale
in Computer Engineering for
Embedded System

Master Degree Thesis

A Low-Power Embedded System for Outdoor Industrial Geolocation Applications

Relator
prof. Andrea Acquaviva

Co-Relator
Dott. Daniele Jahier Pagliari

Candidate
Igli Baruti

April 2018

Abstract

Embedded Systems are computers created for single purpose applications, optimized to offer great performance at very low cost. Today we talk about Internet of Things and ultra low-power devices at home as well as in factory since there are many possible applications. The real research in Embedded is to find the best solution, in terms of cost, size, performance, power consumption or complexity for the specific purpose to which it is dedicated. This thesis documents the implementation of an application able to get the position on GPS coordinates and transmit them to a server connected wirelessly on a wifi network. The system was though for automotive production area and boast as strong point the low-power consumption, low cost, size and complexity. The system. connected in a network, is able to communicate with a server and is based on the principle of vibrations analysis, thanks to an accelerometer, to implement a dynamic power management of the modules used and limit power consumption.

Index 1

1. Introduction

1.1Outline

The document is structured with an initial abstract with the recap of the thesis scope, six chapters and three appendixes for further information about modules, protocols and implemented code:

Abstract: brief discussion about the study case, the thesis and what is aimed to do

Chapters

- Introduction: short description of "Factory 4.0", what is and principles on which is based, description of FCA and introduction to the study case considered in this thesis.
- Background: the state of art of the technology adopted in this case, listing pros and cons and justifying the choice made.
- Contribution: the hardware and software implementation made to realize the prototype, so the configuration of the Raspberry, the project plan, so reasons of the adopted technologies, algorithms and protocols, the modules configurations, the implementation of the previous versions the FSM structure, the PSM for power analysis, the DPM policy adopted for the power optimization.
- Results: the final implementation, what is able to do, comparisons, considerations and application on field.
- Discussion & Conclusion: alternatives and possible updates, description of errors, problem found and solution adopted and indoor part introduction.
- Bibliography: list of websites and documents considered for the realization of the thesis.

APPENDIX A: small description of the modules adopted, for detailed description check bibliography

APPENDIX B: list of some protocols used for interconnections of modules with the raspberry board.

APPENDIX C: pieces of the python code implemented like code blocks of GPS data management and power management.

APPENDIX D: acronyms adopted in the document.

For further information about protocols and modules the reader is invited to check the bibliography or the appendixes for brief explanation.

1.2 Overview

The evolution of industries, in particular the manufacturing and service system, in the last years, followed the advancement of information technology in order to improve productivity and quality of manufacturing environment.

Nowadays with Industry 4.0, or “Smart Factory”, we denote a new kind of industry that join all the developments in Information Technology, such as the Internet of Things (IoT), Cybersecurity, cloud and cognitive computing, cyber-physical systems, to enhance industrial processes of automation and data communication. We can resume this new industrial revolution in four principles:

– **Interoperability**: the ability to interconnect in a network, typically an IoT, devices, sensors and people in order to communicate and exchange data in a straightforward way.

– **Information transparency**: the resolution of the surrounding environment through a set of sensors that provide a detailed context of information.

– **Technical assistance**: the ability to automate the problem fixing without human involvement or to support them in unpleasant and exhausting tasks**.**

– **Decentralized decisions**: the ability of cyber-physical systems to make decisions on their own and to perform their tasks as autonomously as possible. Only in the case of exceptions, interferences, or conflicting goals, tasks are delegated to a higher level.

The synergy between IT improvements and industries leads to an increase of productivity maintaining the overall cost (in terms of easiness of tasks and economic costs) low, reducing costs of maintenance, keeping trace of the product in its entire productive chain.

However, there is a negative side too, the digitalization of the industry introduces the cyber security issue, a factor that cannot be ignored, especially on embedded systems and low-power devices. Also, the almost full automation of the industry leads to a non-negligible risk, that’s why the human presence it’s always necessary.

FCA (Fiat Chrysler Automobiles) is one of the biggest auto-maker companies that is focusing a lot on the modernization of plants and factories, aiming to compete with other industrial realities that already embraced the dynamics of the “Smart-Industries”.

This thesis addresses a study case of FCA, regarding tracking of vehicles in the car park using a device able to give its position, regardless the environment (indoor or outdoor) and communicate with the server through a wireless network.

So we aim to create a network of cars in an environment where we keep trace of position, actual and previous, using an embedded system, with its own distinctiveness that we will see later, that communicate with a central server to fill a database (the back-end project of handling data sent by devices is the second part of the project not covered in this thesis) of position, car identifier, time and other info as validity flag or number of positions changed (to define during requirements phase). The indoor part was not covered in this thesis due to the different technology adopted for the 2 kinds of environment, the GPS was suitable for open and large areas, but questionable in closed context, where was used instead, the Bluetooth and the Beacons technology.

Figure 1: Outdoor car park

As already told, one of the goals of Factory 4.0 is the interoperability principle of products. A network of interconnected cars produces a database where is possible to know always the position and the background history of each car and its movement, making easier to find, to know in which production phase it is and so, making the technical assistance faster and untroubled.

1.3 Project Plan

So before starting the project we must take in consideration the Requirements of the study case:

1) a system mounted in a car that will send its updated position to a server.

2) a battery-powered system with no other power supply connected

-- Low power system

3) small size of the final system (although in the prototyping we used a complex board like the Raspberry)

4) low cost device, for mass-production

5) performance trade-off, real time is unnecessary, but positioning precision must be accurate.

6) able to connect and communicate data, wirelessly with a main server.

7) simple implementation, the device must not have complex hardware inside, in order to limit power consumption and system size

Then we can consider the Use Cases:

1) the device should use a GPS module able to get the latitude and longitude of the device

2) once it got the position it sends the data to the server that will handle it and put in the database

3) it should not consume too much, so we should limit the usage of GPS and Wifi transmission as much as possible

4) it should recognize the stationary and motion phase, to avoid useless data communication during motion but only when stationary

5) should distinguish from permanent stop to erroneous or short stops (position only when permanently stopped) with the help of an accelerometer.

After defined requirements and use cases, it's possible to set up the hardware and software technology to adopt, taking in consideration the previous points.

Hardware:

-- hosting board: we selected the Raspberry Pi 3 model B

-- accelerometer: we tried 2 devices, Sparkfun adxl345 and Berry-IMU LSM9DS0, we choose the adxl345

-- GPS, again we tried with 2 models, Adafruit Ultimate V3 and NEO-6M and choose the V3

-- Wi-Fi module: we used the internal module, embedded in the raspberry.

-- ethernet cable: connecting the rpi to a pc.

-- wires: to connect rpi GPIOs to modules

Software:

-- programming language: Python, because of the great amount of resources regarding programming on RPI

-- Operating System: Raspbian Stretch 4.9

-- Matlab: for analysis of vibration voltage to obtain thresholds.

-- PuTTY: to create an ssh connection from rpi to pc

-- Xming: to visualize on pc the OS of the rpi

-- python packages: we will see for each module the library imported and package adopted

Now that we have a first set of requirements, use cases and selected HW/SW component, we create the structure and the algorithm that is exposed in Chapter 3.

2. Background

Geolocator devices are already adopted in industries and many other fields, car GPS trackers are a great example of geolocation utilization.

These devices can be turned on/off manually or on car ignition and use intensively the GPS module inside to provide a constant updated position from the satellites. On the other hand, the device consumes so much energy that has to be constantly connected to a source of power, in this case the car itself, its size is not negligible, and cost is high.

For our purpose these devices are unsuitable, so another solution is the one adopted in the car sharing environment: cars are monitored with the GPS inside that gives the position and track the itinerary of the car in movement, until turned off. Although the solution adopts some low power strategies (turned off during long stationary periods), its cost is still too high, and the tracker feature consumes a lot of power, it can't be battery powered.

So, what is the kind of solution we are looking for?

The solution is an ad-hoc small low-power embedded system, battery powered, connected to the wireless network and able to communicate with the server.

Low-Power: the system must use as less as possible the high power consuming modules, like GPS and Wifi, to extend battery lifetime. Our solution is to reduce the transmissions of position only when the car is definitely stationary, avoiding uses of these modules during movement (no tracking).

Small size: the final product will be an embedded system that will mount a small processor (8-bit data), a memory (few Kbytes for temporary data), an E^2PROM for the code, a gps module, an accelerometer, a Wifi and a Bluetooth module and some GPIOs. The implementation with the Raspberry Pi3 is a prototype, the board has dozens of modules we don't need in the final implementation so in the calculation of the power consumption, we have to take into account this factor.

Wireless-Connection: positions of the cars are sent to the main server through a socket connection in a wireless network, data is split in TCP packets and sent to a server in constant listening.

Low-Cost: in our case performance is not particularly distinctiveness: we can choose low-cost GPS modules since we don't want tracking but positioning. And other modules (accelerometer and wifi) are cheap and makes this solution suitable for mass production.

Outdoor/Indoor: the final product should work both on external environments, with the use of the GPS and internal structure with Bluetooth localization through Beacon network. Our prototype implements the outdoor part.

3. Contribution

3.1 Raspberry PI 3

The Raspberry PI 3 (appendix A.I) is the board choose for the prototype, due to its easiness to program and the huge number of resources made available for programmers. It was preferred respect other the previous versions, mainly because it has an internal Wi-fi module (not present for example in the PI 2), which is better respect an external module, both in terms of power consumption and performance. The board has an OS, Raspbian Stretch 4.9.

3.1.1 Board Setup

Following, the steps to configure the board to the PC, the choice to connect the board to the pc and to peripherals through usb connection was made to not consider in our power consumption estimation their contribution and because they reduce portability and so the possibility to test it outside, moreover the detailed guide can be found on site given in bibliography, here we have the main points of our configuration:

1) Flash the SD card with the operating system Raspbian Stretch 4.9 using the SD Formatter 4.0, an open-source software available online and insert the SD card into the board
2) Connect the board to the pc through micro-USB as power supply and via Ethernet cable
3) We need 2 software on PC side to communicate with the RPI: PuTTY, an SSH client software that allows connection to the shell of the RPI and Xming, a X11 display server that combined with ssh connection allow to use the GUI of Rasbian. Then initiate a ssh session to the local address of the RPI. The prompt will ask for username and password which of default are:

   Username: pi

   Password: raspberry

Once connected we start enabling in raspi-config serial communication protocols like I^{2}C, SPI and UART, we change the configuration of network address switching from dhcp (that give the address automatically once connected) to static, defining which will be our address:

sudo nano /etc/network/interfaces

change the row **iface eth0 inet dhcp** to **iface eth0 inet static** and adding below the lines

address 192.168.1.10
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
gateway 192.168.1.254

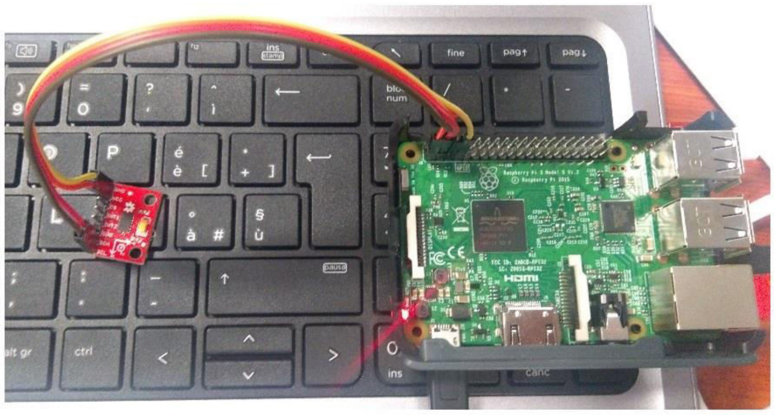
So that next connections will be to that specific address. The same thing for the wifi interface, but to the **wlan0** row.

Finally, after connection and configuration of the boards we start to initialize our modules preparing them for the project.

3.2 Accelerometer

The accelerometer adopted was the Sparkfun adxl345 (check A.III), a small and low-power module that we will use for our project. We need it because as told before, our goal is to detect the position of the car when it is stopped and we use the vibration of the car in motion to distinguish the two phases of motion/stop. This accelerometer can detect these small movements, gives information to decide when to turn on the GPS for capturing a position and send it to server. Furthermore, the module is the only one always on, that's why the power consumption factor is significantly important.

The module is connected to the board through I^2C connection in the appropriate GPIOs like in picture, (the 4 wires are GND, VCC (3,3V), SDA, SCL):



In Appendix C.I there is the code related to the initialization of the values of Range, from 2g to 16g, for small vibrations better to keep the lowest value so it is more sensitive, of Bandwidth, the frequency response to dynamic acceleration (in our case vibrations) that goes from 25 Hz to 1600 Hz depending on intensity of vibrations that we want to capture and of axes taking in consideration earth gravity.

In order to use the information given by the accelerometer, we implement a function getAxes (C.1) that will return the data of the three axes x, y, and z.

The idea is to compare consecutive values of the axes to see if there was a vibration, the procedure that follow is:

- If [x,y,z] consecutive values are equal the car is stopped or in constant acceleration, but since we are using a bandwidth high enough to sense dynamic acceleration, only the first case is possible
- When values are different we have vibrations: to distinguish transient vibrations due to accidental movement of the device, from the movement of the car, we adopt a threshold in terms of time and intensity so to not consider them

- When time and intensity are above the thresholds adopted, we can tell that the car is in movement

After that, when values turn equal for a given time, we can say that car is stopped and proceed with positioning.

So, the key point is the choice of the timeouts to detect motion/stop and the sensitivity to vibrations, because the main problem is to distinguish a temporary stop to a permanent stop to avoid useless activation of the GPS and WIFI module

3.2.1 Threshold sensibility

The choice of thresholds was made analyzing a typical car route, identifying 3 phases: stop with car Off, motion with car On and in movement and stop with car On. Distinction between stop with car On and Off is trouble because it depends on the type of car, for example electric car and cars with the Stop&Go system produce no vibration at all during temporary stops. The only distinction can be made between generic stop and motion using a timeout value.

Axes x : time
Axes y: vibration intensity on 1 of the 3-axis scale of the accelerometer

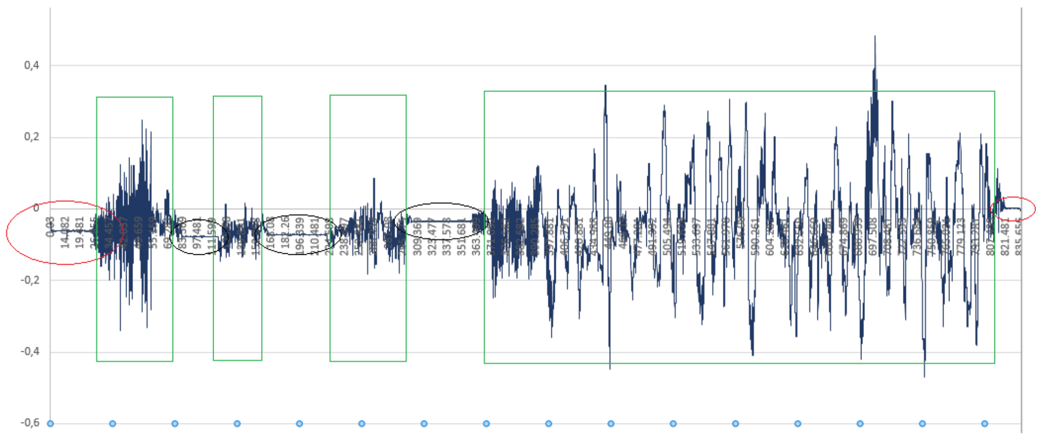


Figure 2: Vibration analysis made on car with Android: red circles are the stop phases with car Off, at the beginning and at the end; black circles are the ones with car On and green squares represent the motion phases.

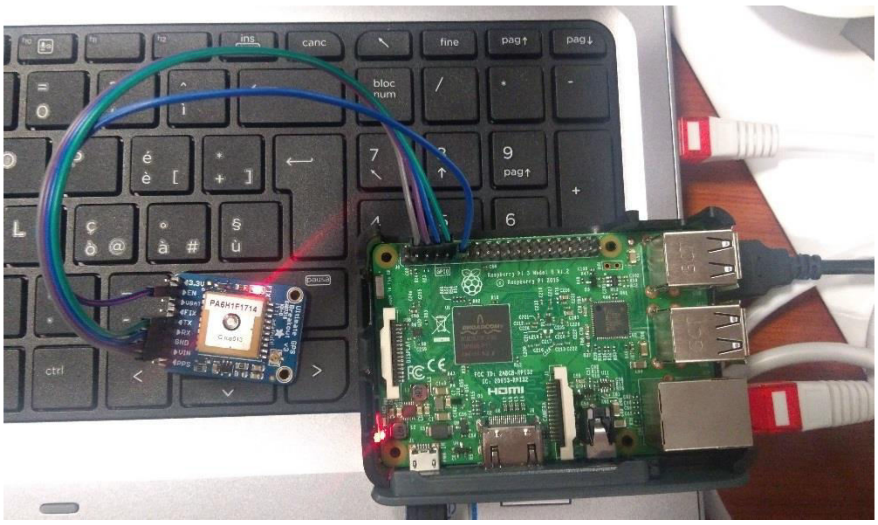
In the picture we can see the effective difference in terms of vibrations when the car is moving and when is not and, also the duration of these phases. But stops with car On and Off are indistinguishable.

The solution proposed is to apply a time value of stationarity and considering our particular case we can set it to several minutes in fact, the car during production phase is not moved frequently, we can assume a distribution where there is a lot of stationary time and a few movements of small duration. The analysis made in the picture is the worst scenario where the difference of motion and stop is clearly visible.

In this case the GPS is activated only several minutes after the last stop, in this way the GPS is turned On only when necessary, since it is the most critical module in terms of power consumption.

3.3 GPS

The GPS adopted is the Adafruit Ultimate V3 (Appendix A.II), a ultra low-power module, with just 20 mA of current draw, and great quality and sensitivity. We connect the module to the board with UART connection like in picture (the wires are the block of 4 for VCC (5 V), GND, RX connected to the TX of the board and TX connected to the RX of the board, plus a wire for the ENABLE pin):



and enable the UART connection on the board-side, adding at the config.txt file the line enable_uart =1:

```
1. sudo nano /boot/config.txt
```

and add at the end, the line

```
1. enable_uart=1
```

the next step is to download the libraries needed to convert and understand the rough data coming from the GPS to the board, so we install the package related to GPS with the command:

```
1. sudo apt-get install gpsd gpsd-clients python-gps
```

disable the services that can interfere with the application:

```
1. sudo systemctl enable gpsd.socket
2. sudo systemctl start gpsd.socket
```

and connect the serial port of the board to the GPS:

```
1. sudo gpsd /dev/ttyS0 -F /var/run/gpsd.sock
```

finally, we are ready to use the functionalities of the gps package to read data from gps.

In C.I there is the code of configuration of the GPIO responsible of turning On/Off the module, the V3 has a pin ENABLE that allow to disable and so to cut current draw of the module by simply 0/1 to that pin. Moreover there is the creation of a function that allow us initialization and data acquisition from the GPS: in our case "gpsd" is a data structure that contains the information of latitude, longitude, altitude and other information, once activated and prepared, with the gpsd.start() function, it returns values we wanted in string form like gpsd.fix.altitude, gpsd.fix.longitude ecc.

We need to make a consideration, the GPS once turned off, requires time to be ready and acquire a new position, this depend mostly on the quality signal from satellite. Using the device outside, on open environments, allow to a faster position fix, we are talking about 20 seconds to several minutes in case the antenna is covered and not directly looking in the sky.

3.3.1 Vibration Sensitive Analysis

The use of the GPS module is strictly related to the analysis of vibrations data given by the accelerometer. The application works like a positioning system and not like a tracker, which would be more expensive in terms of power consumption, so we need to get the position only when the vehicle is stationary, at least for predefined period and not when it's moving. The idea was to consider car vibrations as signal to distinguish stationarity and motion, as we will see later I made some tries with the only accelerometer inside a car to check the effective sensing of these 2 states and results were as expected, the difference was high enough to be recognized easily by the sensor and we apply this strategy for the DPM implementation (see C.II).

3.4 WI-FI

The last module to discuss is the wifi module built-in the RPI 3. To transmit the position to the server, we must be connected wirelessly through a wifi network. But data transmission on a wifi network leads to a non-negligible power consumption of the antenna, so of the overall system. The same consideration done for the GPS must be taken here, we must reduce as much as possible activity of the WIFI module.

This can be done acting a static DPM as before, turning On the module and open a connection to the server when we need to send data and turning it Off when unneeded. The communication is made using a client-server socket connection from the device to the main server, following the instruction on both the device-side and server side.

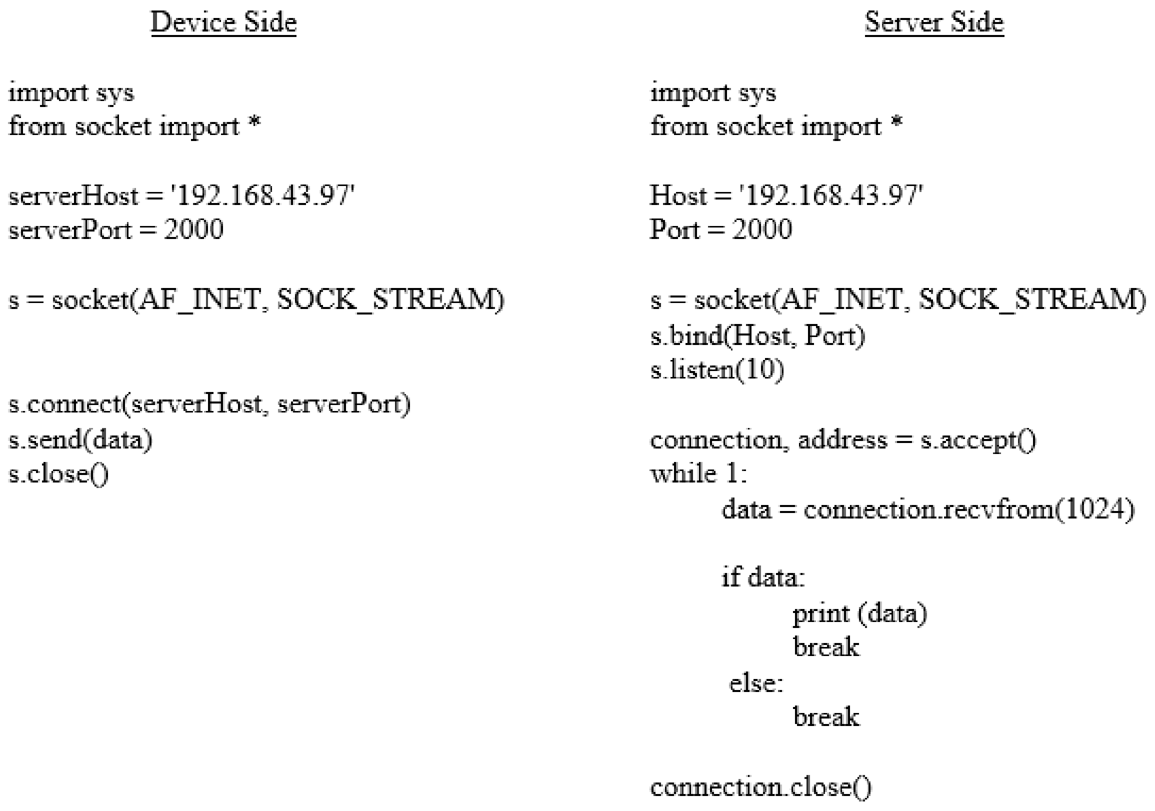
To disable the internal wifi module we used the command **sudo ifconfig wlan0 down**, the command that disable the wireless interface wlan0 and used the option **up** to enable it again, on the application the command was run thanks the function *system* imported from the module *os:*

os.system("sudo ifconfig wlan0 up") *os.system("sudo ifconfig wlan0 down")*

3.4.1 Client-Server communication socket

The stream socket is a connection-oriented mode based on TCP protocol, to transmit and receive data through a network among nodes inside it, used sometimes even for interprocess communication. And was the solution adopted due to its easiness of implementation, and speed of connection and data transmission.

Below we have the scripts used on the device and on server side, this one was running on the pc to simulate the server and the explanation of the python code:



Device Side

```
import sys
from socket import *

serverHost = '192.168.43.97'
serverPort = 2000

s = socket(AF_INET, SOCK_STREAM)


s.connect(serverHost, serverPort)
s.send(data)
s.close()
```

Server Side

```
import sys
from socket import *

Host = '192.168.43.97'
Port = 2000

s = socket(AF_INET, SOCK_STREAM)
s.bind(Host, Port)
s.listen(10)

connection, address = s.accept()
while 1:
        data = connection.recvfrom(1024)

        if data:
                print (data)
                break
         else:
                break

connection.close()
```

First, we import the package *socket* that contains functions and variable definitions that we will use

*from socket import **

we proceed to create a socket variable *s* on both sides, specifying the type of addressing and the type of socket

s = socket(AF_INET, SOCK_STREAM)

AF_INET	→ IP protocol-based socket
SOCK-STREAM	→ indicates a TCP socket

Then, on server side we tell the port that is listening with the instruction *bind()*

Server_socket.bind(Host_address, Port_number)

and put the port in waiting state specifying the maximum number of pending connections

Server_socket.listen(max_num_pending_connections)

On client side we are ready to connect to the server socket

Client_socket.connect(ServerHost, ServerPort)

and on server side to accept the connection

connection, address = Server_socket.accept()

accept() return a file descriptor of the opened socket and the address. Once the connection is successfully done the server goes on a loop ready to receive data with the command

data = connection.recvfrom(size_of_data)

and the client can send with

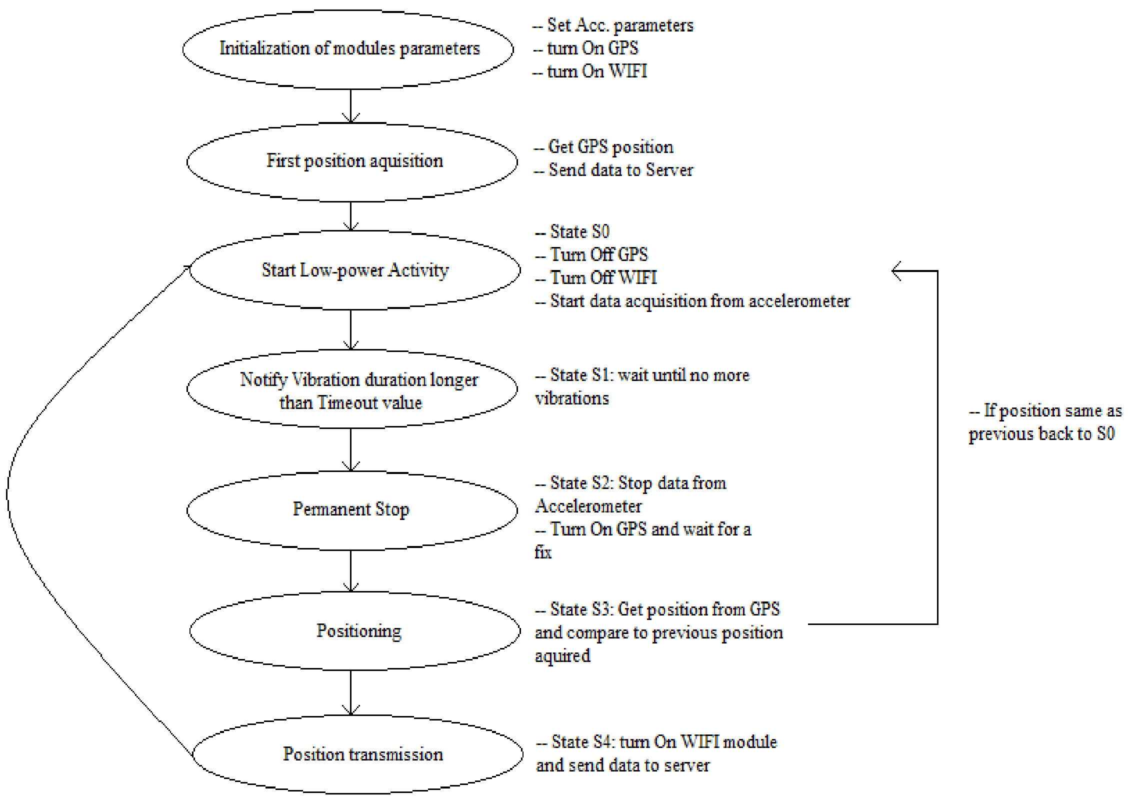
Client_socket.send(data)

The data to send is in String format, so is up to the server to parse the information and get the individual values. In Appendix C.II Data Management, there is the implemented code and the utilization of these instructions to send positioning information like latitude, longitude and altitude.

I proceeded with this type of connection for sake of simplicity and for not overloading the system with complex connections even if better and more secure.

3.5 Software FSM

In this section we will see the functional flow of the application that can be described in the FSM (Finite State Machine) picture below.



Initialization of modules parameters
– Set Acc. parameters
– turn On GPS
– turn On WIFI
First position aquisition
– Get GPS position
– Send data to Server
Start Low-power Activity
– State S0
– Turn Off GPS
– Turn Off WIFI
– Start data acquisition from accelerometer
Notify Vibration duration longer than Timeout value
– State S1: wait until no more vibrations
Permanent Stop
– State S2: Stop data from Accelerometer
– Turn On GPS and wait for a fix
Positioning
– State S3: Get position from GPS and compare to previous position aquired
Position transmission
– State S4: turn On WIFI module and send data to server
– If position same as previous back to S0

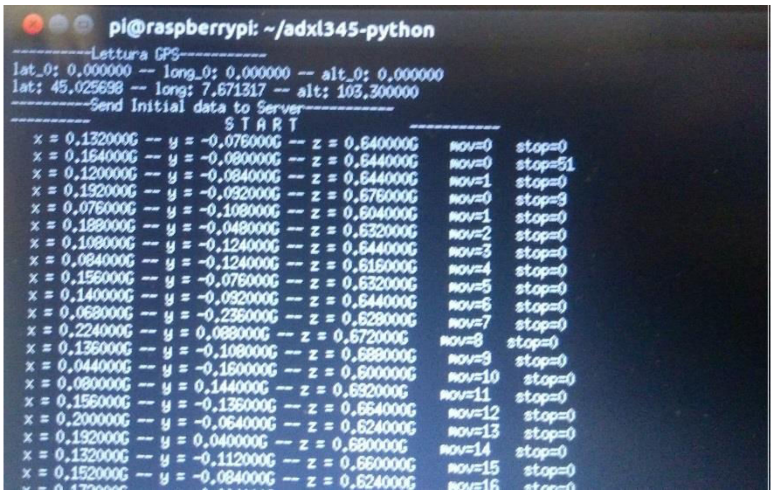
We adopted this solution to make an easier power management, as we will see later each state of the system has its functionality and so its relative power consumption, splitting the application in blocks help to estimate this measure.

The application starts with a configuration phase, to initialize variables and prepare peripheral modules, then we proceed with the first acquisition of data, even if not strictly necessary, an initial positioning is useful to check if all configurations went well.

After that all the module are turned off except for the accelerometer, the device is put in an idle low-power state waiting to sense vibrations long enough to determine a movement.

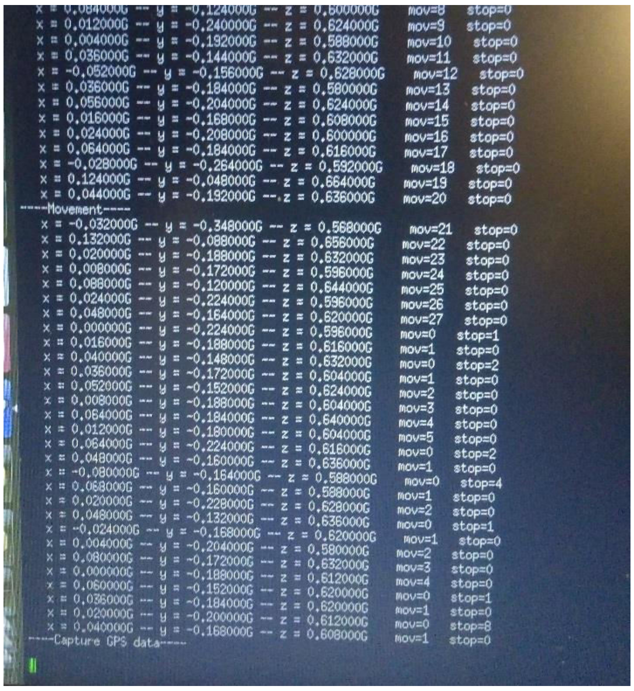
In the figure BELOW we can recognize the first position acquisition, transmission and beginning of idle phase (---- START ----). Values x, y and z are the axis of the accelerometer on which we are rely.

mov and *stop* are the counter flags used for timeouts: the first is used for vibration sensitivity, see in the second picture as it reaches a certain threshold, print --- MOVEMENT --- and from state S0 goes to state S1:



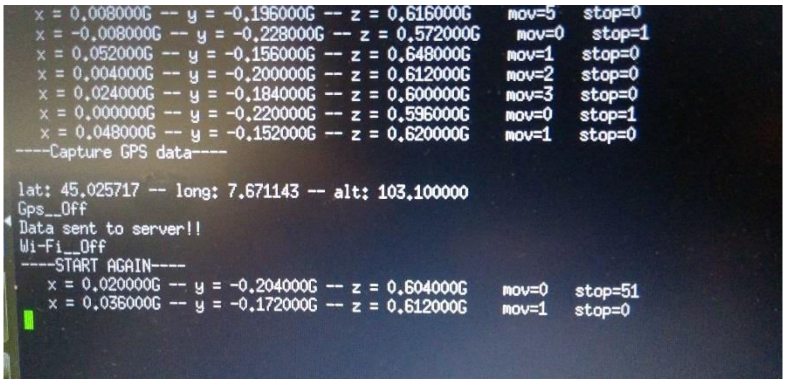
```
pi@raspberrypi: ~/adxl345-python
----------Lettura GPS-----------
lat_0: 0.000000 -- long_0: 0.000000 -- alt_0: 0.000000
lat: 45.025698 -- long: 7.671317 -- alt: 103.300000
----------Send Initial data to Server-----------
----------                START                 -----------
   x = 0.132000G -- y = -0.076000G -- z = 0.640000G     mov=0    stop=0
   x = 0.164000G -- y = -0.080000G -- z = 0.644000G     mov=0    stop=51
   x = 0.120000G -- y = -0.084000G -- z = 0.644000G     mov=1    stop=0
   x = 0.192000G -- y = -0.092000G -- z = 0.676000G     mov=0    stop=9
   x = 0.076000G -- y = -0.108000G -- z = 0.604000G     mov=1    stop=0
   x = 0.188000G -- y = -0.048000G -- z = 0.632000G     mov=2    stop=0
   x = 0.108000G -- y = -0.124000G -- z = 0.644000G     mov=3    stop=0
   x = 0.084000G -- y = -0.124000G -- z = 0.616000G     mov=4    stop=0
   x = 0.156000G -- y = -0.076000G -- z = 0.632000G     mov=5    stop=0
   x = 0.140000G -- y = -0.092000G -- z = 0.644000G     mov=6    stop=0
   x = 0.068000G -- y = -0.236000G -- z = 0.628000G     mov=7    stop=0
   x = 0.224000G -- y = 0.088000G -- z = 0.672000G    mov=8   stop=0
   x = 0.136000G -- y = -0.108000G -- z = 0.688000G     mov=9    stop=0
   x = 0.044000G -- y = -0.160000G -- z = 0.600000G     mov=10    stop=0
   x = 0.080000G -- y = 0.144000G -- z = 0.692000G    mov=11   stop=0
   x = 0.156000G -- y = -0.136000G -- z = 0.664000G     mov=12    stop=0
   x = 0.200000G -- y = -0.064000G -- z = 0.624000G     mov=13    stop=0
   x = 0.192000G -- y = 0.040000G -- z = 0.680000G    mov=14   stop=0
   x = 0.132000G -- y = -0.112000G -- z = 0.660000G     mov=15    stop=0
   x = 0.152000G -- y = -0.084000G -- z = 0.624000G     mov=16    stop=0
```

If this happens, the device waits for periods without vibrations long enough to determine a stop, exactly the contrary of the previous state.



```
   x = 0.084000G -- y = -0.124000G -- z = 0.600000G     mov=8    stop=0
   x = 0.012000G -- y = -0.240000G -- z = 0.624000G     mov=9    stop=0
   x = 0.004000G -- y = -0.192000G -- z = 0.588000G     mov=10    stop=0
   x = 0.036000G -- y = -0.144000G -- z = 0.632000G     mov=11    stop=0
   x = -0.052000G -- y = -0.156000G -- z = 0.628000G     mov=12    stop=0
   x = 0.036000G -- y = -0.184000G -- z = 0.580000G     mov=13    stop=0
   x = 0.056000G -- y = -0.204000G -- z = 0.624000G     mov=14    stop=0
   x = 0.016000G -- y = -0.168000G -- z = 0.608000G     mov=15    stop=0
   x = 0.024000G -- y = -0.208000G -- z = 0.600000G     mov=16    stop=0
   x = 0.064000G -- y = -0.184000G -- z = 0.616000G     mov=17    stop=0
   x = -0.028000G -- y = -0.264000G -- z = 0.592000G     mov=18    stop=0
   x = 0.124000G -- y = -0.048000G -- z = 0.664000G     mov=19    stop=0
   x = 0.044000G -- y = -0.192000G -- z = 0.636000G     mov=20    stop=0
-----Movement-----
   x = -0.032000G -- y = -0.348000G -- z = 0.568000G     mov=21    stop=0
   x = 0.132000G -- y = -0.088000G -- z = 0.656000G     mov=22    stop=0
   x = 0.020000G -- y = -0.188000G -- z = 0.632000G     mov=23    stop=0
   x = 0.008000G -- y = -0.172000G -- z = 0.596000G     mov=24    stop=0
   x = 0.088000G -- y = -0.120000G -- z = 0.644000G     mov=25    stop=0
   x = 0.024000G -- y = -0.224000G -- z = 0.596000G     mov=26    stop=0
   x = 0.048000G -- y = -0.164000G -- z = 0.620000G     mov=27    stop=0
   x = 0.000000G -- y = -0.224000G -- z = 0.596000G     mov=0    stop=1
   x = 0.016000G -- y = -0.188000G -- z = 0.616000G     mov=1    stop=0
   x = 0.040000G -- y = -0.148000G -- z = 0.632000G     mov=0    stop=2
   x = 0.036000G -- y = -0.172000G -- z = 0.604000G     mov=1    stop=0
   x = 0.052000G -- y = -0.152000G -- z = 0.624000G     mov=2    stop=0
   x = 0.008000G -- y = -0.188000G -- z = 0.604000G     mov=3    stop=0
   x = 0.064000G -- y = -0.184000G -- z = 0.640000G     mov=4    stop=0
   x = 0.012000G -- y = -0.180000G -- z = 0.604000G     mov=5    stop=0
   x = 0.064000G -- y = -0.224000G -- z = 0.616000G     mov=0    stop=2
   x = 0.048000G -- y = -0.160000G -- z = 0.636000G     mov=1    stop=0
   x = -0.080000G -- y = -0.164000G -- z = 0.588000G     mov=0    stop=4
   x = 0.068000G -- y = -0.160000G -- z = 0.588000G     mov=1    stop=0
   x = 0.020000G -- y = -0.228000G -- z = 0.628000G     mov=2    stop=0
   x = 0.048000G -- y = -0.132000G -- z = 0.636000G     mov=0    stop=1
   x = -0.024000G -- y = -0.168000G -- z = 0.620000G     mov=1    stop=0
   x = 0.004000G -- y = -0.204000G -- z = 0.580000G     mov=2    stop=0
   x = 0.080000G -- y = -0.172000G -- z = 0.632000G     mov=3    stop=0
   x = 0.000000G -- y = -0.188000G -- z = 0.612000G     mov=4    stop=0
   x = 0.060000G -- y = -0.152000G -- z = 0.620000G     mov=0    stop=1
   x = 0.036000G -- y = -0.184000G -- z = 0.620000G     mov=1    stop=0
   x = 0.020000G -- y = -0.200000G -- z = 0.612000G     mov=0    stop=8
   x = 0.040000G -- y = -0.168000G -- z = 0.608000G     mov=1    stop=0
-----Capture GPS data-----
```

And only the GPS module is activated to get the position of the vehicle, if the data is not suitable, it returns on idle state. The GPS module is turned off independently on the correctness of data to avoid further consumption and if data is good then the WIFI module is turned on, created a socket connection to the server and data is transmitted. Then it begins again from S0.



```
   x = 0.008000G -- y = -0.196000G -- z = 0.616000G     mov=5    stop=0
   x = -0.008000G -- y = -0.228000G -- z = 0.572000G     mov=0    stop=1
   x = 0.052000G -- y = -0.156000G -- z = 0.648000G     mov=1    stop=0
   x = 0.004000G -- y = -0.200000G -- z = 0.612000G     mov=2    stop=0
   x = 0.024000G -- y = -0.184000G -- z = 0.600000G     mov=3    stop=0
   x = 0.000000G -- y = -0.220000G -- z = 0.596000G     mov=0    stop=1
   x = 0.048000G -- y = -0.152000G -- z = 0.620000G     mov=1    stop=0
----Capture GPS data----

lat: 45.025717 -- long: 7.671143 -- alt: 103.100000
Gps__Off
Data sent to server!!
Wi-Fi__Off
----START AGAIN----
   x = 0.020000G -- y = -0.204000G -- z = 0.604000G     mov=0    stop=51
   x = 0.036000G -- y = -0.172000G -- z = 0.612000G     mov=1    stop=0
```

3.5.1 Configuration

We must distinguish between the calibration and the configuration phase. The first is the analysis of the vibrations of the vehicle made before the application is launched, we need to specify the ideal timeout value and vibration sensitivity in order to optimize the performance of the system.

For example, making a study of the typical path of the vehicle to check the number and durations of temporary stops, or choosing a low vibration sensitivity in case of trucks.

While the configuration phase is the initialization at the beginning of the application, of the 3-axis of the accelerometer and its relative parameters, the address and port of the server to connect, and the functions and values of the GPS, like the enable pin adopted switch it On/Off. All values are stored outside the source code, in a JSON file, check the Appendix C.I for the code.

3.5.2 Acquisition

Position acquisition is made with the GPS module introduced before and is the most critical phase due to its large power consumption when active. But we must consider even the latency of the provided data: once turned On, the module needs some time, from 30 seconds to few minutes to be ready to get a fix from satellite, it depends on sky coverage and environment.

A solution adopted was to plug an external antenna to the module to magnify the signal acquisition and reduce this time of latency. In the test on field, inside a car the module can be ready for positioning in 1 minute without the external antenna, thanks to the high performance given by the adxl345.

3.5.3 Transmission

As already said before communication is made with a client-server socket connection from the application to the main server. GPS position (latitude, longitude, altitude) plus ID of the device and time with hour and minutes are put in a string and send through TCP connection. On server side then, there is the parsing of the data to retrieve all the information. Of course, this solution was adopted since other best options in terms of speed and security were complex to implement that could need more computational power and leads to an increase of power consumption and overall complexity.

With socket the server can handle up to 100 client connections, but it has to be implemented on server-side and it was out of the scope of this thesis, so the example proposed (3.4.1) was for just 5 pending client connections. To sum it up, the board was connected via ethernet cable to the pc, to print data and help to check the correctness of the application and when data was ready, it was sent to another pc, or just the same one, wirelessly through wifi network. On the pc was implemented a script simulating the server (see 3.4.1) and made it run so we could see that the data position sent by the board, was received.

3.6 Power Management

The most important property to preserve for this application is the low-power factor. Below there is a table showing the differences between few possible solutions that were considered, evaluating Power consumption, the power dissipated by the overall system, Complexity, the difficulty of realization, Performance, the amount of information, latency and accuracy of data, Cost, the overall cost including modules:

Solutions	*Power consumption*	*Complexity*	*Performance*	*Cost*
Tracking	*Very High*	*Low*	*High*	*High*
Positioning: Based on GPS	*High*	*Normal*	*High*	*Normal*
Based on Vibrations	*Low*	*Normal*	*Low*	*Normal*
Based on Acceleration	*Low*	*Very High*	*Low*	*Normal*
Positioning based on vibrations with DPM	*Very Low*	*Normal*	*Low*	*Low*
Positioning based on vibrations without DPM	*Low*	*Low*	*Normal*	*Normal*

The Tracking solution, is the solution to maintain all the modules active to keep always an updated position of the vehicle. The best solution in terms of performance, thanks to the always updated data and complexity, no need for DPM, but the worst in terms of power consumption, precisely because all modules are always active, even when unneeded. To know the position of the car in motion is useless for our purpose.

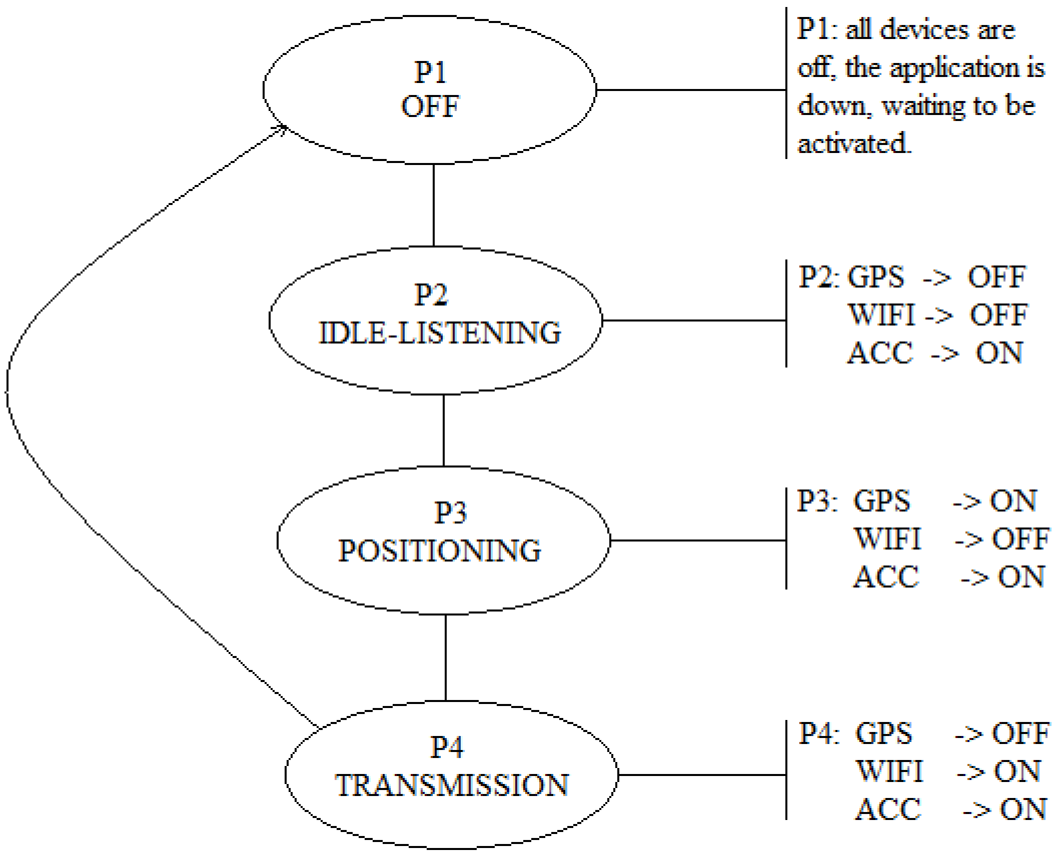
So, the best solution is to limit the use of GPS only when car is stationary. And we can decide on 3 methods: to distinguish between motion and stationarity we could use the GPS position to know when effectively the position changed and when not with interspersed acquisitions, better than tracking in terms of power consumption but still high, or we could use the accelerometer to sense acceleration of cars, but what about vehicle in constant speed? Before starting to find a solution, we can try to assume that cars in motion make vibrations, even if very small and considering this principle we can determine the different state shown in the previous FSM. So, the best solution is the positioning based on vibrations that allow us to identify the right moments to take GPS data.

To improve this method, we can use the DPM technique, simply turning on/off modules without even find possible intermediate low-power states. It will allow to choose low-cost modules, since performance is not strictly needed. Of course, switching on/off a module will create a great latency in data acquisition and transmission: as said before 20 seconds to 5 minutes to activate the GPS and 5 to 10 seconds to turn on the wifi module, connect to the wireless network, create a connection with the server and transmit data. On our study case the worst-case scenario was a vehicle moved 4-5 times per day with break of hours so timing to give a GPS position is not important, so worse performance in exchange of extreme low power consumption is a great deal. The DPM solution was made turning on/off modules according to the Power State Machine elaborated below.

3.6.1 Static DPM

The PSM is finite state machine that represents the different power states of the workload. We identified 4 states of different energy consumption, each related on the module that is active.

Power State Machine



Essentially the difference between power states is the module active in that moment; the state P1 is the state off. In a possible future development, we thought turn off manually the application when not necessary to improve its lifetime, but this imply the physical attendance of someone for each device. So, we preferred to leave the application in an idle-listening state P2 where the accelerometer is the only module active, since it is a ultra-low power module, this state can last for very long periods.

ADXL345 current draw: 40µA with voltage supply: 3,3V -> electric power = 132µW

On P2, we are in the state of positioning, where the GPS is active and getting data, in this case we should add the consumption of the GPS module.

Adafruit V3 current draw: 20mA with voltage supply: 3,3V -> electric power = 66mW

And on P3, the GPS is turned off and the internal wifi module is enabled and prepared to transmit data.

WIFI current draw: 33mA with voltage supply: 5V -> electric power -> 165mW

Of course, energy required of wifi is higher than the GPS, transmission is always critical in terms of power consumption, but the module in our case is activated for just 10 seconds and then turned off again and it does just a single transmission so compared to energy required by the GPS is a lot lesser:

<u>GPS -> Average Power = 0.066W Average Time = 180 s Energy required = 11.88 J</u>

<u>WIFI -> Average Power = 0.165 Average Time = 10 s   Energy required = 1.65 J</u>

<u>ADXL345 -> Average Power = 0.000165W Average Time = 3600 s Energy required = 0.59 J</u>

For 1 hour of activity (3600 s) and 1 activation (1 positioning with GPS active for 180 s and 1 transmission with wifi module active for 10 s) of the application we see the great difference of contribution in terms of power consumption and understand why turning off the accelerometer during positioning and transmission will not give enough power saving to compensate the loss in performance of switching On/Off the module.

3.6.2 Power Saving Optimization

In the following we have the strategies adopted to obtain a low-power application:

- selection of hardware: the choice of modules was made considering their power consumption and capability to be turned on quickly once turned off, in order to avoid long periods of active modules.
- DPM: turning off devices when unneeded
- Activating only one module at time to avoid overcharging and burst of consumption
- Turning off other useless modules as Bluetooth, HDMI, LEDs and (adopted for the raspberry prototype). Use the accelerometer as a vibration sensor to recognize stages where to turn on modules.
- Compare the acquired GPS position with the previous, so that redundant data are not transmitted and wifi is not activated. This happens in case of error or unwanted vibrations that activate the process.

  Allow to save energy and not overload the server with useless data
- Prevent activation of modules due to temporary vibrations, setting a timeout large enough for active period, ex. if the device sense vibrations for not longer than 10/20 seconds, doesn't activate the modules and continue to keep the idle-listening phase.
- Another possible solution is to turn off the Accelerometer module during positioning and transmission but considered the extremely low consumption of the module the gain in terms of power saving is minimal (see 3.6.1).

3.7 Power Estimation

Measurement of power consumption were made with estimations according to the specifics of all the module connected and the board adopted. Of course, since these are estimation measurements, that can be different from the real consumption, we cannot fully trust them, but this is a prototype (the raspberry board is pretty like a small computer), on the final product the embedded system that we will create will mount only the modules that really needs, an 8-bit processor and memories for code and data, and power consumption will be totally different.

Ex the board mount a quad-core Cortex-A53 at 1.2 Ghz that is way over the 8-bit processor that was thought for the final device and dozens of modules not necessary for our scope.

Another reason to make estimations instead of measure directly is that consumption of the board was way too higher of consumption of single modules, so we could not make a clear distinction from the different power states seen previously and determine the effective power of all of them, the raspberry pi3 power consumption is of 1.4W and modules are of order of mW and µW.

4. Results

The final prototype consists on a raspberry pi3 board connected to the pc via microUSB for power supply and via ethernet for the ssh connection that allow to print data to the pc. On the board the accelerometer was connected to the GPIOs relative to I^2C connection and the GPS to the GPIOs related to UART connection, see the below picture of the full mounted board. On "server-side", that in our case was simulated by the pc before launching the application, is necessary to run the server script seen before, able to prepare the connection on the predefined socket, for the board. After setting up everything, we can run the python program and check its functionality.

The test-on-field was made placing the board with the application inside a car to check if it works and after few tries (in order to make it works we had to choose the right sensitivity to vibrations and a timeout value enough to distinguish between temporary stops from permanent stops, that was not too long) the system demonstrated its operations.

The wireless network adopted to include the board and the pc was the one shared by the phone and actually we saw that positioning was made respecting the rules of the algorithm and transmission was always successful to the pc.

Another result checked was the effective gain in power reduction using our method respect other methods, such as using the GPS as a tracker or not applying a DPM and leaving the modules always on, the difference on these two methods was that the first always get gps position and send to server, while the other restrict the number of positioning and transmission but leaves the modules active.

5. Discussion & Conclusion

Our goal is to prove that an application embedded in a device able to handle GPS positioning, for an outdoor environment car park is possible, maintaining a low-power profile that lead the system to a long lifetime even if battery-powered. And with the analysis of vibrations made by an accelerometer, we can minimize the activation of expensive operations, such as GPS position and data transmission through wifi network and confirm that our goal is possible.

We have to take in consideration the possible use cases, since the analysis of vibrations is strictly related to the kind of environment, to the vehicle tested and its path along the car park and structure of wireless network.

So, a pre-configuration step before the application is initiated, must be taken in order to enhance the vibrations threshold adopted, to decrease/increase the timeout value for motion and stationarity and to agree the right parameters according to the wireless connection implemented.

Eventually, the system is able to launch the application and perform its operation without any external help and last long periods thanks to the low power idle-listening state.

The complete application will involve another feature, that is the relative geolocation in an indoor environment using a network of Bluetooth Beacons and about the overall project, the implementation, on server side, of the application that will handle the data received. For this part of the project, I used a pc running a python script to simulate the server.

5.1 Future Development

There are other possible developments to take in consideration like for example security:

the position data is sent wirelessly through socket connection without any sort of protection, data are not critical so securing the connection is useless considering even that a more complex system is needed, but if we want to guarantee a certain security we can consider the idea of cryptographic communication made with an HSM or a cryptographic accelerator, always taking into account the possibility to increase complexity and power consumption.

The possibility for the server to ask the position any time he wants, of course all modules should be always connected to the wireless network and that means that their wifi module would be always On, so a lot of power consumption.

Always on server side, in case of earthquake or defects distributed all over the network of devices, we can consider issuing an error if more than a certain number of devices sends simultaneously data to the server and act accordingly cutting off connection or simply ignoring those who come later, which is good to prevent a denial of service. In the second case on application side the device should be able to send back data after some time if server have ignored it.

Or the possibility to assign to the device more activity modes to switch depending on the request (secure connection, fast mode with high power consumption, sleep mode to turn off even the accelerator and disable the functionality for a certain period), keeping in consideration that each feature leads to an increment of complexity and power waste.

6. Bibliography

Raspberry configuration

https://pihw.wordpress.com/guides/direct-network-connection/

https://www.raspberrypi.org/documentation/

Sparkfun ADXL345 datasheet

https://www.sparkfun.com/datasheets/Sensors/Accelerometer/ADXL345.pdf

Programming accelerometer with RPI

http://ozzmaker.com/guide-to-interfacing-a-gyro-and-accelerometer-with-a-raspberry-pi/

http://www.instructables.com/id/3-Axis-Accelerometer-ADXL345-With-Raspberry-Pi-Usi/

Adafruit Ultimate Breakout GPS V3

https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf

Socket programming

https://www.python.it/doc/howto/Socket/sockets-it/sockets-it.html

https://docs.python.org/2/library/socket.html
www.tutorialspoint.com › Python › Python - Networking

Programming Raspberry GPIO

https://learn.sparkfun.com/tutorials/raspberry-gpio/python-rpigpio-api

Develop with Python

https://pythonprogramming.net

https://www.python.org/about/gettingstarted
https://www.programiz.com/python-programming

APPENDIX A

I Raspberry PI 3 - Model B

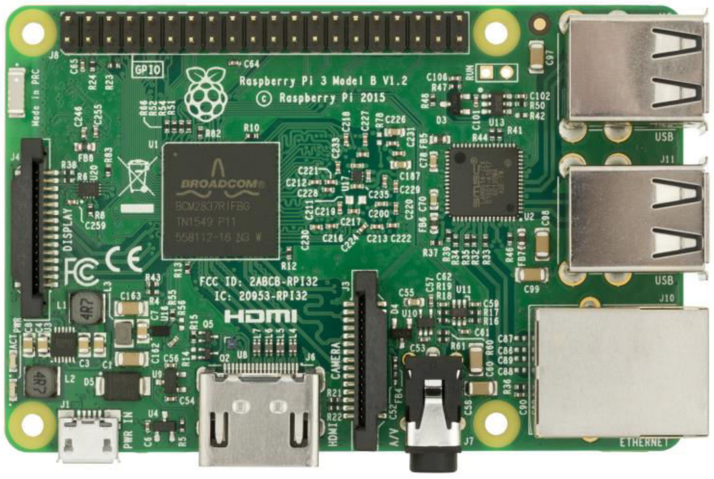


Figure 3: Raspberry Pi 3 model B

The Raspberry Pi 3 model B is a very common single-board computer used for development of small project in electronic and robotic fields. The reasons that makes it popular are the cheap price of the system, the easiness to use and the great HW performance.

It includes a quad-core superscalar ARM Cortex-A53 processor, with 64-bit and 1.2 GHz, an L2 shared cache of 512 KB, a 1 GB RAM and several internal modules as:

- 4 USB 2.0 port
- MicroSDHC slot
- BCM 43438 Wireless LAN and Bluetooth Low Energy (BLE)
- Full size HDMI
- 40-pin GPIO
- CSI/DSI camera and display port

II Adafruit Ultimate GPS Breakout V3 / GPS Neo 6M

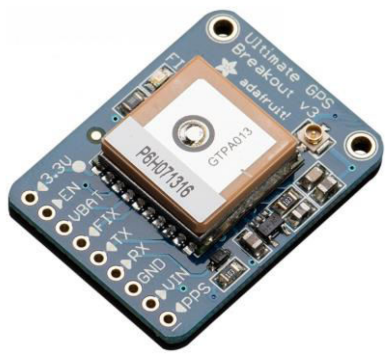


Figure 2: Adafruit Ultimate V3

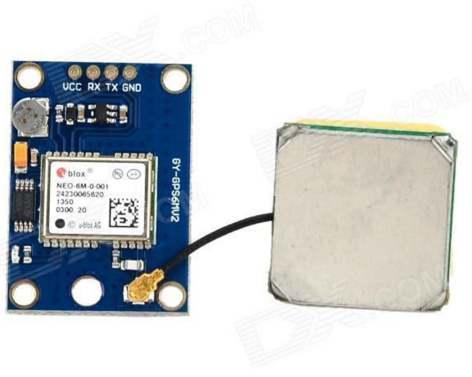


Figure 3: NEO 6-M

The Global Positioning System (GPS) receiver is a device that receives from satellites (at least four) data about time and position and compute, using navigation equations, its own position in terms of latitude and longitude coordinates. The 2 GPS taken in consideration are the V3 and the neo-6M.

Adafruit Ultimate V3

The Adafruit Ultimate Breakout V3 GPS has 66 channels to track up to 22 satellites and so to speed up satellite acquisition, has a maximum of 10 Hz Updates, -165 dBm of receiver sensitivity. Then a low dropout 3.3V regulator, an internal led that indicates the state of the GPS, alternatively can use the FIX pin connected to an external led and an EN (enable) pin that allows to turn on/off the GPS. Internal antenna of 15mm x 15mm x 4mm.

The 20-25 mA during navigation and tracking respectively, makes it suitable for low-power and battery operated applications.

NEO 6-M

The u-blox NEO 6-M has 50 channels, 5 Hz update rate and sensitivity of -160 dBm, is a valid alternative to the V3 with same configuration and an external stronger antenna respect the other, useful for semi-closed environment where satellite signal is weak.

III ACC

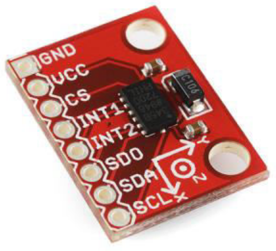


Figure 4: SparkFun adxl345

The Digital Accelerometer ADXL345 of Sparkfun is a 3-axis acc. with 13-bit resolution for measurement in range of ±2g, ±4g, ±8g and ±16g. The device is ultra low-power with current draw of 40μA in active mode and 0.1 μA in standby mode and acquisition frequency in range of 25-1600 Hz. Provide serial interfaces as SPI and I2C.

IV ACC BERRY IMU

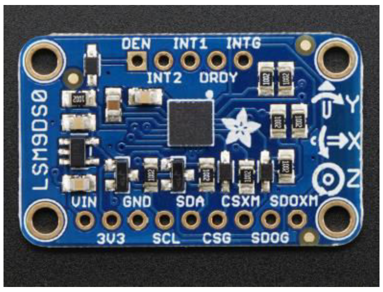


Figure 5: LSM9DS0 Acc/Gyr/Mag

Is a multifunctional module that act as accelerometer, gyroscope, magnetometer. Serial interfaces with SPI and I2C, current consumption in normal mode (accelerometer and magnetometer) of 350 μA and 6 μA in power-down mode. Range of accelerometer from ±2g to ±16g, magnetic scale of ±2 / ±4 / ±8 / ±12 gauss and angular rate of ± 245 / ±500 / ± 2000 dps.

APPENDIX B

I SSH

The Secure Shell is a network protocol for remote connection on computer systems in a unsecured network.

Mostly used for remote login, this protocol allows to create a secure channel of communication among nodes with client or server SSH mounted on them with authentication made available with passwords or cryptographic key pair, public and private (SSH keys)

II I2C

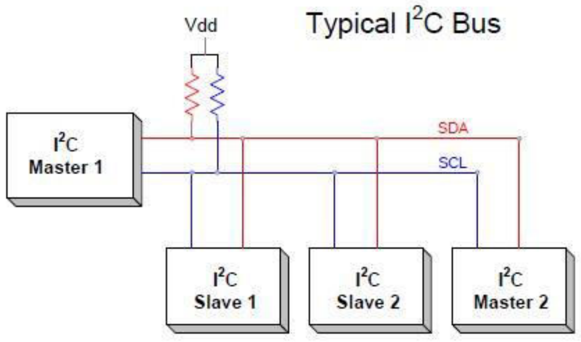


Figure 6: Example of I2C interconnection

I2C (Inter-Integrated Circuit) is a serial bus communication protocol for multi-slave and multi-master systems. It needs only 2 bidirectional open-drain lines, one for data called SDA and one for clock called SCL. Works on 7-bit address space and speeds of 100 kbit/s, arbitration occur comparing the SDA level with the expected one, if different means that another master is sending data.

This protocol is mostly used to reduce wires overhead and allow peripheral devices connection with discrete bus speed.

III SPI

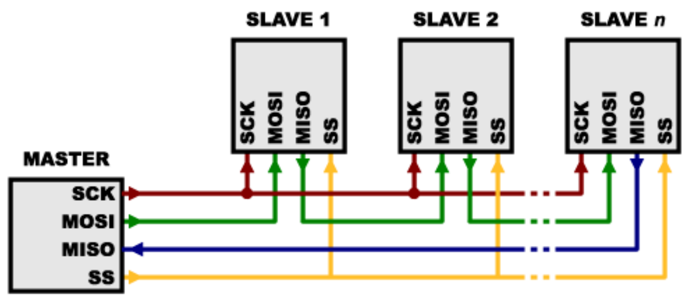
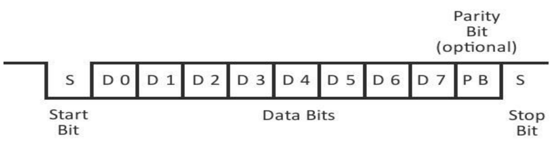


Figure 7: Example of SPI interconnection

SPI (Serial Peripheral Interface) is a synchronous serial protocol for single-master multi-slave system that adopt 4 wires for connection: one for clock, SCK, one for selecting the slave to communicate with, SS, two half-duplex channel, a Master Out Slave In, MOSI and a Master In Slave Out, MISO that allow to a full-duplex communication among master and slaves.

Bus speed is of order of Mbits/s so better than I2C, but with more space overhead. For this application were tried both solution to interconnect modules to the RPI and I2C was chosen

V UART



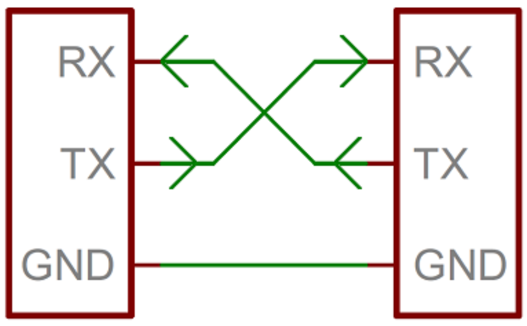


Figure 8: Example of UART interconnection

UART (Universal Asynchronous Receiver-Transmitter) is an asynchronous serial communication protocol with configurable speed transmission. Allow to communicate in simplex, half duplex and full-duplex. The transmitter split the bytes of data in single bits and send them sequentially while the receiver reassembles them with a shift register. As seen in the picture, data sent by the transmitter starts with a special start bit (e.g. logic 0) and ends with a stop bit (logic 1), with eventually a parity bit.

APPENDIX C

I Modules Configuration

JSON data

```
import json
json_file = open("filename.txt", "r")
data = json.load( json_file )
```

GPS Configuration

```
from gps import *
import time
import threading
import RPi.GPIO as GPIO

gpsd = None
GPIO.setmode(GPIO.BCM)
GPIO.setup(17,GPIO.OUT)
GPIO.outuput(data["pin"],1)

Class GpsPoller(threading Thread)
        def _init_(self):
          threading.Thread._init_(self)
          global gpsd
          gpsd = gps(mode=WATCH_ENABLE)
          self.current_value = None
          self.running = True
        def run(self)
          global gpsd
          while gpsd.running:
                gpsd.next()
```

Accelerometer Configuration

```
class ADXL345:
    address = None
    def __init__(self, address = 0x1d):        #IB: changed 0x53 for 0x1d new i2c address
        self.address = address
        self.setBandwidthRate(BW_RATE_25HZ)
        self.setRange(RANGE_2G)
        self.enableMeasurement()
    def enableMeasurement(self):
        bus.write_byte_data(self.address, POWER_CTL, MEASURE)
    def setBandwidthRate(self, rate_flag):
        bus.write_byte_data(self.address, BW_RATE, rate_flag)
    def setRange(self, range_flag):
        value = bus.read_byte_data(self.address, DATA_FORMAT)

        value &= ~0x0F;
        value |= range_flag;
        value |= 0x08;

        bus.write_byte_data(self.address, DATA_FORMAT, value)
        def getAxes(self, gforce = False):
        bytes = bus.read_i2c_block_data(self.address, AXES_DATA, 6)

        x = bytes[0] | (bytes[1] << 8)
        if(x & (1 << 16 - 1)):
            x = x - (1<<16)

        y = bytes[2] | (bytes[3] << 8)
        if(y & (1 << 16 - 1)):
            y = y - (1<<16)

        z = bytes[4] | (bytes[5] << 8)
        if(z & (1 << 16 - 1)):
            z = z - (1<<16)

        x = x * SCALE_MULTIPLIER
        y = y * SCALE_MULTIPLIER
        z = z * SCALE_MULTIPLIER
        if gforce == False:
            x = x * EARTH_GRAVITY_MS2
            y = y * EARTH_GRAVITY_MS2
            z = z * EARTH_GRAVITY_MS2
        x = round(x, 4)
        y = round(y, 4)
        z = round(z, 4)
        return {"x": x, "y": y, "z": z}
```

II Data Management

```
axes = adxl345.getAxes(True)


if ( (int)(appx*val_vib) != (int)(axes['x']*val_vib) ) or ( (int)(appy*val_vib) != (int)(axes['y']*val_vib) ) or
( (int)(appz*val_vib) != (int)(axes['z']*val_vib) ):
        print ("   x = %fG -- y = %fG -- z = %fG    mov=%d   stop=%d" % ( axes['x'],axes['y'],axes['z'],
vett_mov, vett_stop ))
        vett_mov = vett_mov + 1
        vett_stop=0
        if(flag==0 and vett_mov > move_value):
                flag=1
                print("----Movement----")
else:
        vett_mov=0
        if(vett_stop < (stop_value+1)):
                vett_stop=vett_stop+1
        if(flag == 1 and vett_stop > stop_value):
                …C.III WIFI/GPS Power Management
```

III WIFI/GPS Power Management

```
GPIO.output(data["pin"],1)                                  #ENABLE GPS
print("----Capture GPS data----\n")                 #START handling gps/transmission
gpsp = GpsPoller()                                          #start polling gpsd
gpsp.start()
while lati == 0:                                    #wait until a valid value is get
        lati = gpsd.fix.latitude                            #get latitude
longi= gpsd.fix.longitude                                   #get longitude
```

```
alti = gpsd.fix.altitude                                                  #get altitude
print ("lat: %f -- long: %f -- alt: %f " % (lati,longi,alti))             #print GPS data on screen
GPIO.output(data["pin"],0)                                                #DISABLE GPS
print("Gps__Off")
if (lati != prev_lat or longi != prev_lon):                         #check if different from previous value
        cmd = 'sudo ifconfig wlan0 up'                                    #prepare command
        os.system(cmd)                                                    #ENABLE WIFI
         time.sleep(8)                                                    #wait connection
         s = socket(AF_INET, SOCK_STREAM)                                 #create socket
         print("Data sent to server!!")
         s.connect((serverHost, serverPort))                              # connect to server on the port
                                                                          #connect to remote server
         s.send(str(i)+' Position: LAT= '+str(lati)+' LONG= '+str(longi)+' ALT= '+str(alti))           # send
the data
         s.close()                                                        #close connection
         cmd = 'sudo ifconfig wlan0 down'                                 #prepare command
         os.system(cmd)                                                   #DISABLE WIFI
         prev_lat = lati
         prev_lon = longi
         print("Wi-Fi__Off")
lati=0
print("----START AGAIN----")
```

APPENDIX D

GPS	Global Positioning System
RPI	Raspberry PI
PSM	Power State Machine
FSM	Finite State Machine
DPM	Dynamic Power Management
GPIO	General Purpose Input/Output
OS	Operating System
TCP	Transmission Control Protocol