"Integration of a GPS sensor board for a drone localization application"

second year internship report

14 June 2011 - 19 August 2011 (10 weeks)

Maxime France-Pillois

Internship supervisors : Soraya Arias and Roger Pissard-Gibollet

INRIA Rhône-Alpes
Service **E**xpérimentation et **D**éveloppement
655 avenue de l'Europe
38330 Monbonnot Saint Martin

August 19, 2011

Acknowledgment

First of all, I would to thank the “Service Expérimentation et Développement” of INRIA Grenoble for their reception in the service. They allow me to have a nice internship in a welcoming team.

I specially thanks Guillaume Roche, Christophe Braillon and Clement Burin des Roziers which helped me to overcome some software or hardware problems and which were on hand when I faced issues.

I wish express my gratitude to Roger Pissard-Gillobet and Soraya Arias for allowing me to take part in this captivating internship. Thanks to them, for following my work and for helping me all along this period. Without their devotion I was not able to reach the same internship results. Not forgetting their valuable help during the writing of this report. At last, I would to thank them for the confidence which gave me.

Finally, I wish thank my schools Ensimag and Phelma for giving me the opportunity to discover the software engineer work during internship period. And I thank the internship department of Ensimag and Claudie Marchand (SED) to carry out the administrative tasks, in particular the internship agreement, allowing me to performed this internship.

Contents

I Introduction

As part of the 2nd year internship of Ensimag school, I took part in an INRIA project within the "Service Expérimentation et Développement" of the Grenoble lab.

So this 10 weeks internship takes place in an engineers service in charge to help research project to set up experimentation platforms or to develop source code.

In this context, my mission aims to provide support to a doctoral student working in the NeCS (Networked Controlled Systems) research team. To validate her control algorithms of robots fleet she needs an experimentation platform. Due to its low cost, the selected robot was the AR.Drone Parrot. But these algorithms need absolute positions of the robot whereas the AR.Drone Parrot provides only relative positioning information. Thus, to set up an adapted physical platform with this kind of robot, the absolute position of the drone have to be provide. To reach this goal, I worked on the integration of two additional sensors on the drone : a GPS[1] and an accelerometer. With this two sensors, the absolute position of the robot can be gotten and so, the platform can fulfill the doctoral student expectations.

Actually, this internship subject was composed of two main parts : the first one was the writing and the testing of a GPS driver dedicated to the chip available in INRIA lab. And the second part was the carrying out of wireless driver communication based on a real-time operating system[2].

This report will present briefly the work carried out during this internship and the solutions implemented to fulfill the lab expectations. To do this, this document is divided in six main parts. In a first time, I present the environment of this internship : the company, the work organization, etc. Secondly, I introduce my internship subject and the expectations of my supervisors. Then, I present the choices made and the software architecture set up to fulfill the internship aim. Next, I describe more precisely the work I had done and the implantation problems faced. After that, I tackle to the validation tests issues and I summarize the internship results. And at last, I conclude with the balance of acquired experiences during this 10 weeks.

[1]**G**lobal **P**ositioning **S**ystem (**GPS**) is a space-based global navigation satellite system that provides location and time information in real-time.

[2]see Glossary

II Internship environment

My internship took place at the “Institut National de Recherche en Informatique et en Automatique” of Monbonnot(38), most precisely in the “Service Expérimentation et Développement”. INRIA is a public research laboratory specialized in computer science. Set up in 1967, INRIA is nowadays divided to 8 labs located all over France (Paris, Lille, Bordeaux, Grenoble, Nancy, Rennes, Saclay and Sophia Antipolis) hiring 3350 scientific people (1300 researchers, 1200 doctoral students and 250 post-doctoral students). I did my internship in the lab of Grenoble which hired 650 persons divided in 34 research projects. The lab of Grenoble is promoting the work with other public labs of the area like the “**L**aboratoire d’**I**nformatique de **G**renoble” or the “ laboratoire **G**renoble **I**mages **P**arole **S**ignal **A**utomatique”. Some research fields studied in INRIA of Grenoble are for example :

- software programming : especially in the software reliability.
- modeling and simulating : in biology field in particular, with modeling on cells life.
- human machine interfacing with the interactive 3D TV or the developing of smart cars for example.

More specifically, for this internship, I joined the “Service Expérimentation et Développement”, SED. This service was created to fulfill three main goals :

- to maintain a network of expertise to disseminate best practices of software development and use of community tools within project teams
- to set up, develop and maintain the experimental platforms with project teams
- to take part in software development within project teams

During this internship, I was supervised by two members of SED department : Soraya Arias and Roger Pissard-Gibollet. Both are research engineers specialized in Real-Time software and robotics. Soraya Arias, is being in charge of the drones use in INRIA, followed my work very closely. Roger Pissard-Gibollet, the head of the department, had more a decision-making role and he validated our choices for the project.

In the SED department, the schedule of interns is not strictly defined. All interns have to do 35 hours per week, but can organized their work as they want. Personally, I start working around 9 am and stop around 6 pm. Moreover, to facilitate the monitoring of my work, I write, each week, and publish on the SED-wiki Web site, a summary of the work carried out during the week. Thus, my supervisors or other SED people were able to know my progress.

To be able to organize easily my work, at the beginning of my internship, my supervisors provided me a schedule of tasks I had to do during this period. This schedule allowed me to see the work breaking structure and to understand easily the aim of my work. You can find this forward-looking schedule, represented as a Gantt diagram, in appendix A.4.

III Internship work presentation

III.1 Work overview

My internship subject aims to integrate a GPS and accelerometer sensor node within a system controlling a drone[3] robot.

This work should allow INRIA's researchers to develop easily drone positioning based algorithms. Indeed, presently, drones provide only relative information of his motions. The reachable information for users are relative orientations and velocities. But, algorithms designed need absolute positions of the robot. That's why, the integration of additional sensors is required. These sensors enable to get, in "real-time", the absolute position of the drone. Therefore, a positioning control of drones is feasible. Indeed, at the end of my internship, the system I am working on, should send positioning GPS frames and accelerometer frames in "real-time". Thus, researchers could be get the absolute position of robots and check their algorithms on a real platform thanks to drones integrating these sensors.
Note : Due to project hardware limitations, position data are not provided in strict real-time. A latency delay, of few milliseconds, is existing between the data recovery by users and the actual position of the drone. These limitations will be explained in the section V.

The whole required system is split in two different parts as described in the following diagram 1.

1. One part, is built-in on the drone. This is the sensor part where drone motions and positions are estimated. This part is called the node in this report.
2. The other part is called the sink. It is linked to a computer which drives the drone fleet.

The two parts have to communicate together. The drones are driven by the computer via a Wifi connection link and the system I designed communicates via radio frequency. This system can not use the same medium link as the drone because the drone mother board is protected and it is not possible to connect the node part directly to the drone system and used the Wifi link to send node data.

On the diagram presented in figure 1, you can see an overview of the entire project. The right part of the diagram is the mobile part composed of a drone and the additional sensors. The left part, is in charge of drive the drone and collect information. The sink gets the absolute positioning information provided by sensors (number 1 on the diagram). The computer has to get relative information sent by the drone (number 2) and to recover, through a serial link, the absolute data provided by the sink. With these two kind of data, the computer compares the real trajectory of the drone to the initial wanted trajectory. The computer computes the errors done, and sends a new relative command of motors velocities to correct that, and reach the expected position or trajectory.

[3]a drone is a flying system needing no human pilot

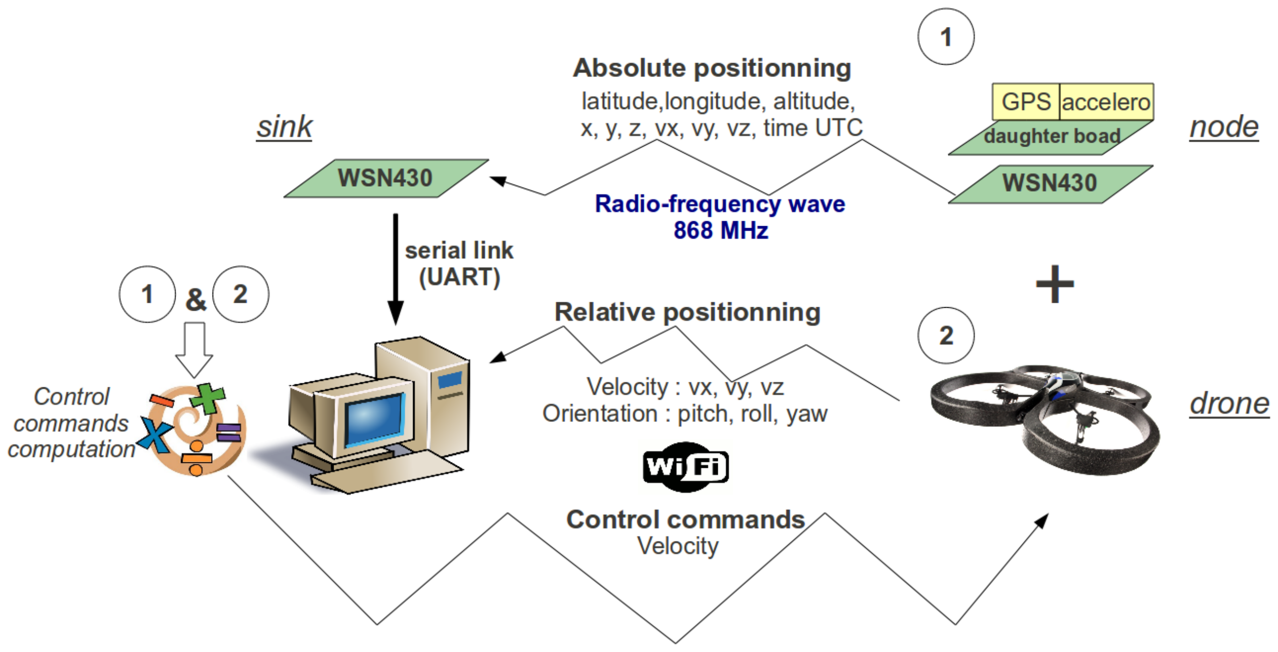


Figure 1: Internship subject overview.

Regarding the hardware support for my project, INRIA has already a large operational sensors network based on electronic boards called WSN430 board. These boards are mainly composed of two *Texas Instrument* chips : a Micro-controller [4] "MSP430" and a wireless radio chip "CC1100". This large scale sensors network is called "Senslab". This is a amount of 1024 environmental sensors nodes distributed in 4 INRIA sites (Rennes, Lille, Strasbourg and Grenoble). Each sensors node is made up of 2 boards : a WSN430 board [2], also called the mother board, and a daughter board with specifics sensors build-in. There are different kind of daughter board with different kind of sensors like temperature sensor, humidity sensor,...
To keep this standard, I had to work on this model. Actually, the two parts of the project will be based on this board. The node part will be composed of a WSN430 board and a daughter board made up of two sensors : a GPS sensor (lea-5H) [1] and a 3-axis accelerometer sensor (lis3lv02dq). An overview of the daughter board is available in appendix A.3. And the sink part will be only composed of a mother board WSN430 to be able to receive radio frequency frames.

To summarize, the work I performed has been divided in two main stages :

- The first one is relative to the GPS sensor : implement a dedicated Driver[5] in C language and perform tests to get the characteristics (accuracy, errors, ...) of this device. The driver has to meet a specific API [6] to be compliant with the others INRIA sensors drivers already built up, and to simplify the reuse of the GPS device.
- The second one focus on the communication and applicative part : implement a program where the both data sensors (GPS and accelerometer) are sent periodically to a sink node via a wireless communication and then these data are available on the sink computer to be integrated in a high level application.

[4]a micro-controller is a "small computer" on a single integrated circuit, containing a processor core, memory, and programmable input/output peripherals like analog-digital converters, serial ports communication,...
[5]see glossary
[6]see glossary

III.2 Existing work

At the beginning of my internship some works were already developed in the "Service Expérimentation et Développement" of INRIA. INRIA engineers have already designed a set of drivers and libraries dedicated to the WSN430 board. Theses programs provide features of the board including processor (MSP430) facilities, wireless communication module, drivers for timers [7], UART [8] or i^2c communication [9] [4]. These drivers have been already checked and used in many other projects, especially the "Senslab" project.

As for the GPS module, using the μ-blox component, a driver has already been implemented. But it was not suitable as it has been designed for a different hardware target board and a different processor. The previous project using the GPS module was focused on energy saving, whereas the actual project is interested mainly in reactivity. Thus, the GPS configuration has to be different. Moreover, the driver that I have to implement needs to be as general as possible to be used in different cases of study. Nevertheless, the GPS modules are the same, so I could learn from previous work how to handle the GPS module features.

The driver of the accelerometer sensor has already been developed. It has already been checked and used in an human body motion analysing application. So concerning the accelerometer sensor, I only have to focus on how to use the driver to retrieve accelerometer data on the application part programs.

Regarding the wireless communication, the hardware board (WSN430) uses a built-in radio chip (cc1100). So the easiest to set up the wireless communication between the sink and the node parts is to go through this chip to send frames via a radio link. An other way to communicate could be to send data to another board which should be in charge of the wireless communication, but this solution being more expensive in resources and time consuming we give up quickly. Different standards of communication have already been designed and implemented for the WSN430 board built-in radio chip, so the use of another board type would imply more development time for a feature already developed.
Three standards of communication have already been released : the TDMA [10] [3], the CSMA [11] and a homemade standard : the XMAC [12].

Concerning the drone and how to control it, programs, using the Parrot SDK (Software Development Kit) APIs, have been implemented by the SED engineers. I have to integrate my work to these programs to check the validity of the entire system.

As for the hardware daughter board, made up of GPS and accelerometer sensors, it has been designed but no functional tests have been performed on it before my internship work. So, I need to perform these tests myself concurrently with the software development.

[7] a timer is a counter module allowing to count precisely a predefined time slot. This is a essential feature in computer science to measure time.

[8] UART or Universal Asynchronous Receiver Transmitter is a serial communication module.

[9] $\boldsymbol{i^2c}$ or Inter Integrated Circuit is a serial synchronous communication bus made of only three wires, used to communicated between two or more integrated circuits or boards in our case. See appendix A.1 for more details

[10] **T**ime **D**ivision **M**ultiple **A**ccess (**TDMA**) is a channel access method for shared medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots. See appendix A.2 for more details.

[11] see Glossary

[12] The XMAC communication protocol is a INRIA home-made communication protocol designed especially for low-power consumption. This kind of communication is suited for low traffic network only.

III.3 Work choices

At the beginning of the internship, the work specifications have been strictly defined by my supervisors. The hardware boards and components are imposed as the main steps of the software implementation.

Regarding the wireless communication protocol between the drone fleet and the sink board I use the TDMA standard. This standard is the best suited for multiple nodes communication.

Concerning the GPS driver implementation, the critical point was the data exchange between the micro-controller MSP430, on the WSN430 mother board, and the GPS module located on the daughter board. To set up this data exchange three different communication buses can be used : the i^2c[13] bus, the serial bus (UART) and the SPI bus. For restricted resource reasons, my supervisors ask me to work on the UART communication. The micro-controller (MSP430) has two ports of communication but each port accepts only some kinds of communication. As the accelerometer data are sent via the i^2c bus through the first port, the easiest solution to send the the GPS module data seemed to use the UART, through the second port, to avoid data sending conflict within the two sensors.
Nevertheless, when working on the wireless communication, I notice that the communication between the micro-controller MSP430 and the radio chip CC1100 is performed using the SPI bus. This bus goes through the second communication port as well as the UART communication used to retrieve the GPS data. So I have to review the previous solution made by my internship supervisors because the use of the UART bus is severely compromised. The TDMA protocol needs constant synchronization frames to be operational, so the second communication port (and the SPI) is always busy with TDMA synchronization frames. So no sharing of this port is possible. Therefore, the UART bus can not be used as suggested and I have to change the communication between the micro-controller and the GPS device. The only solution remaining is the i^2c bus. Nevertheless, this implies to share this bus between the accelerometer data and the GPS data. Thus, these data can not be read exactly at the same time as it should be for an high accuracy localisation and control of the drone. But, as there is no other way to handle on the same board both wireless communication and sensor data retrieving, high level algorithms should take into account this characteristic.

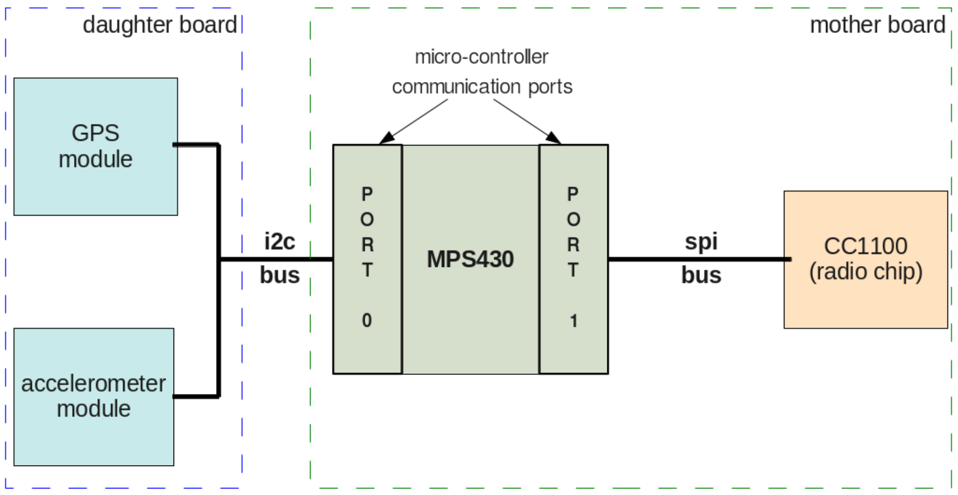


Figure 2: Summary of communication buses used between each chip.

[13]see appendix A.1

IV Internship work description

IV.1 Overview

An overview of the softwares designed is available in appendix A.6.

IV.1.1 GPS driver architecture

In order to be as general as possible, we decide to design the GPS driver for the two microcontrollers which should be able to be linked with the GPS module, according with existing boards in INRIA labs. These boards were the WSN430 board, use in the "Senslab" project and being the principal target of my work, and the "Marathon des Sables" board. The second one was the board for which the first GPS driver was designed.

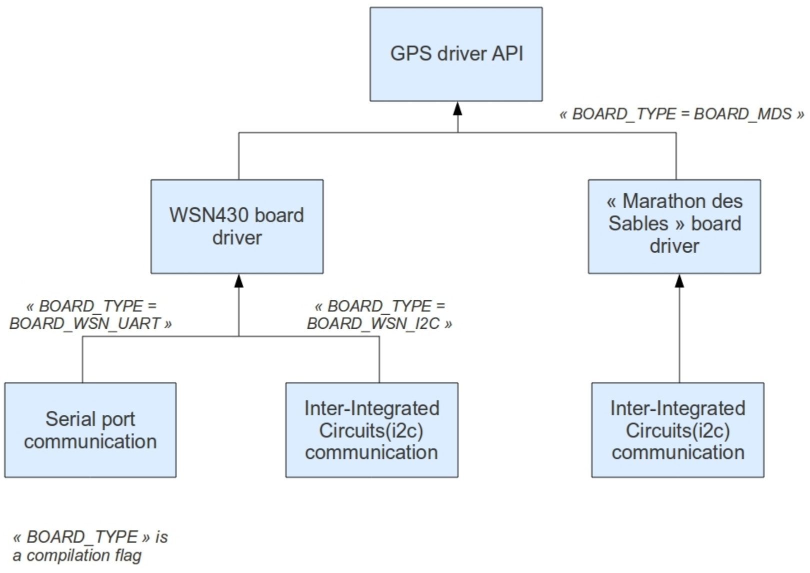

GPS driver API
« BOARD_TYPE = BOARD_MDS »
WSN430 board driver
« Marathon des Sables » board driver
« BOARD_TYPE = BOARD_WSN_UART »
« BOARD_TYPE = BOARD_WSN_I2C »
Serial port communication
Inter-Integrated Circuits(i2c) communication
Inter-Integrated Circuits(i2c) communication
« BOARD_TYPE » is a compilation flag

Figure 3: Global architecture of the GPS driver.

As shown at the top of the figure 3, the designed driver has a single API. it is independent of the targeted board types. To simplify the use of the driver and to meet the standard driver API of INRIA, a single API was required. Then, below this API, a differentiation is made at compilation time depending on the targeted board (a dedicated flag is used, see arrow labels on figure 3). This allows memory space saving, only useful memory is allocated. Moreover, it is not too restrictive for users as they know at the compilation time what board is targeted. Finally, as shown at the bottom of the figure 3, in case of the WSN430 board, GPS driver can handle two communication buses to retrieve the data : the UART and the i^2c. The choice is done using a specific compilation flag. As for the "Marathon des Sables" board only one communication bus is available, i^2c bus.

IV.1.2 Applicative part architecture

Concerning the application part, namely the GPS driver use case programs and program using TDMA communication for sensors data sending, I do not define a proper architecture. Application programs can be seen as a main function, no peculiar architecture needs to be set up. However, to allow the periodic reading of GPS frames, I have to add a software over-layer. This is a supplementary layer on top of the driver API. This layer is explained in subsection IV.2

IV.2 Software implementation

IV.2.1 GPS driver implementation

To write the GPS driver, I do not need to know all the theory about the Global Positioning Systems but mostly how the μ-blox GPS module works. I search for the main principles of GPS on Web and I read the μ-blox GPS data-sheet to understand how the module works. A summary of the basics information needed to design the driver are the following :

- GPS modules have all the same operating principle.
- Thanks to an antenna the GPS module can receive satellite signals.
- With at least three different satellite information, it is able to compute its current position. Generally, the position is given with the standard : latitude [14], longitude [15], and altitude.
- The μ-blox GPS module can provide other kind of information related to the position : a 3-D coordinates x,y,z point relative to the position, the speed vector v_x, v_y, v_z, time from satellite, ...

Concerning the implantation of the driver I design, I organize the implementation in layers sorted by hardware abstraction. As a result, I divide the driver in two layers where each layer is defined by a couple of files (the header file.h and the body file.c). The lower layer (files lea5x.h and lea5x.c) handles the features closest to the hardware. The upper layer (files gps.h and gps.c) is in charge of the most abstract features of the driver.

The basic features of the driver is to configure a GPS module and to retrieve the GPS data. These tasks are realized by the upper layer which set up an user-friendly API to manage all the driver features. To do so, the upper layer needs the lower layer. This one is totally dedicated to the μ-blox GPS module and to the communication between motherboard and GPS on daughter board. The GPS module configuration must set the device to an appropriate mode in order to get position and timestamped data. The principle of the configuration setting is dependant to GPS module used. In our case, the driver targets a single GPS module, so the configuration mechanism is the same for any version of the driver whatever the board or the communication medium wanted.

μ-blox GPS module configuration

The configuration of the GPS module is the big part of the driver. As described in subsection III.3, the device can handle different kind of communication to exchange data : (*i^2c, UART, SPI and USB*). To be able to manage these communication links the GPS module has to be configured differently. Generally, communication with most of computing devices is realized writing or reading data in some specific device registers. But for μ-blox GPS module, the configuration technique is closest to the network frames processing. Namely, the device is able to receive frames, matching a defined frame protocol, and computing them to take into account the received data. To set the configuration, the frame protocol used is a proprietary protocol designed by μ-blox, the device manufacturer. This protocol is composed of synchronization bytes, followed by a header, then the body of the frame and at last the check-sum bytes as shown on figure 4.

[14]the latitude of a location on the Earth is the angular distance of that location south or north of the equator. The latitude is an angle usually measured in degrees. The equator has a latitude of 0°, the north pole has a latitude of 90°north (written 90°N or +90°), and the south pole has a latitude of 90°south (written 90°S or -90°).

[15]the longitude is a geographic coordinate that specifies the east-west position of a point on the Earth surface. It is an angular measurement, usually expressed in degrees, minutes and seconds like the latitude position. The Greenwich meridian has the longitude 0°.

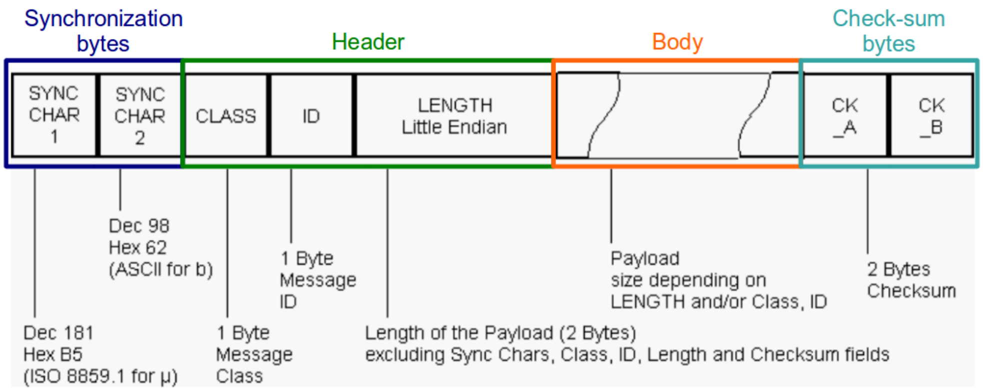

Synchronization bytes
Header
Body
Check-sum bytes
SYNC CHAR 1
SYNC CHAR 2
CLASS
ID
LENGTH Little Endian
CK _A
CK _B
Dec 98 Hex 62 (ASCII for b)
1 Byte Message ID
Payload size depending on LENGTH and/or Class, ID
2 Bytes Checksum
Dec 181 Hex B5 (ISO 8859.1 for µ)
1 Byte Message Class
Length of the Payload (2 Bytes) excluding Sync Chars, Class, ID, Length and Checksum fields

Figure 4: Example of a $\mu - blox$ frame.

As presented on the figure 4 the header is composed of one byte called *class* and one byte call *id*. By analogy, a *class* can be considered as a memory sector and an *id* as a register within this memory sector.

The configuration process is identical for any version of the μ-blox GPS module daughter board. But the retrieving of GPS frames from daughter board to mother board depends obviously of the communication bus selected. To ease the readability and to avoid redundancy in code, the routines in charge of configuring the device produce the related body (header + payload) frame. And another routine is in charge of the sending of the frame according to the communication bus chosen (it adds the correct synchronization and check-sum bytes to the body). This way configuration routines can be used for any GPS module daughter board, and only the sending method depends on the target. So only the sending function, using communication medium drivers, differs with the board or the communication protocol targeted.

Reading GPS frames

The μ-blox GPS module can send data according to two different frame protocols :

- the $\mu - blox$ protocol already presented in figure 4;
- the NMEA [16] protocol, a standard for GPS.

The big advantage of the $\mu - blox$ protocol is that it is a binary protocol, so data encoding is optimal and the rate can be more important. The NMEA protocol sends ASCII [17] character, using 1 byte for encoding one digit. For instance, to encode number 200, the NMEA protocol needs 3 bytes while $\mu - blox$ protocol needs only 1 byte). Nevertheless, the NMEA protocol is the standard protocol independent of the GPS module manufacturer, whereas the $\mu - blox$ protocol is a proprietary one only available for μ-blox modules. The two protocols having their pros and cons, we do not constrain the choice of the data protocol in the driver. The driver can be used to retrieve encoded data by both protocols. However, concerning the program designed to be integrated with drone application, we prefer the $\mu - blox$ protocol for its higher rate.

Once GPS data have been retrieved, the next step is the parsing depending on the chosen protocol. For NMEA frames no computing is required, they can be directly sent to the high level part. As for $\mu - blox$ protocol the parsing process is described below.

[16]see Glossary
[17]see Glossary

Let start with the data retrieving process depending closely of the communication bus. The implemented driver only handles two bus : the UART and the i^2c.

UART communication

When using the UART communication the GPS device sends frames directly on the bus at the frequency selected during the GPS module configuration. The motherboard has to pick the data on bus when there are available. The driver part dedicated to the UART bus, is able to detect data on bus and to raise a software interruption[18] when data are available. Then it is easy to recover GPS frames : when data are received on the bus a flag is raised (interruption), then the data are read and stored in a buffer to be parsed afterwards.

The retrieving and parsing of GPS data must be performed periodically. To fulfill this requirement I decide to take advantage of the interruption raised by the UART to get the data. However, an interruption is raised for each byte received, so whether I call the data parsing method at each interruption the data process is wrong. If the parsing is running while a new data is received, without specific restriction, the parsing is recalled on the same buffer. To resolve this problem, I set up a mutual exclusion when calling the parsing function. Namely, the handler can call the data parsing method only when the first data is received; for the other data being received, the handler only stores them in the buffer. Meanwhile, the parsing function manages to process every data of the buffer.

Note: the mutual exclusion can be performed easily because the driver is designed for mono-processor hardware support. Thus, within the interruption handler (which is, initially, executed in a masked interrupt mode) we can certify that only one execution flow can reach the mutual exclusion barrier at a time.

i^2c communication

When using the i^2c bus the GPS module does not send data anymore. It writes the data frames in an internal buffer, and this is the process interested in the data that needs to retrieve them once written.

As explained in appendix A.1 the i^2c standard works as a master/slaves communication : the master asks a slave to send data frames on the bus. The slave can not send data on bus without the master request. In our case, the master is the micro-controller MSP430 on the motherboard and the slave is the μ-blox GPS module (or the accelerometer module). So the driver (controlled by the MSP430) must send a request to the GPS device to retrieve a data. The mechanism consisting in interrogating a device continually to obtain data is called the polling.

Alike the UART solution, a periodic polling of the GPS module must be settled by the driver to retrieve GPS position information. The easiest solution to get something periodically is to use timers. A timer is a resource used to count time delay precisely. For the record, drivers managing the MSP430 timers have designed in SED department and are ready to use. An essential feature of timers is to raise an interruption at a predetermined moment. The existing timer drivers use this feature and allow a periodic interruption raising. Thus, to periodically retrieve and parse GPS data frames, I use timer driver functions to configure the desired period and the handler to be called (to retrieve and parse data) when an interruption is raised.

The micro-controller (MSP430) has only two timer modules. When the driver is running on bare machine (without any Operating System[19] there is no problem because the two timers are available and the periodic polling can be performed with any timer. But, my whole application needs to be implemented over FreeRTOS (as operating system). And FreeRTOS requires one timer for task scheduling. Moreover, another timer is required by the TDMA task to synchronize data

[18]An interrupt is an asynchronous signal indicating the need for attention.

[19]see Glossary

between receivers. So I can not use timer for the periodic data polling on i^2c bus. According to my supervisors, we decide to perform the periodic data polling for the i^2c in an over-layer part detailed in paragraph IV.2.2. Thus the driver remains independent of the application constraints but an over-layer can be called to ease the use of the driver for i^2c communication.

Concerning the retrieving of data from i^2c bus, I reuse an existing method written in the MSP430 driver set developed by the SED. But I have to modify the bus frequency value. I spend time in this adaptation because in the end the maximum frequency tolerated by the GPS module is different from the value given in the constructor data-sheet (in practice the maximum frequency is 70kHz versus the theoretical 100kHz).

GPS data frames parsing

The data parsing analyzes data frames according to a specific protocol to extract useful information. As presented in figure 4 a well-formed GPS frame respects a strict byte format. Data parsing analyzes each byte to determine its nature and to check whether byte order is respected according to the protocol in use. The parsing function checks the packet length : if the number of payload bytes is different from the length written in the header, the packet is considered as a mistaken frame. Nevertheless, to save time, the parsing function designed does not check the frame check-sum. We suppose that if byte order is respected and packet length is correct, then the frame is valid.

Once verified the packets, data can be analyzed. The GPS module writes frames periodically even if the GPS position data are not valid. Thus after parsing GPS data frames we have to check their validity. Generally, within the GPS data a flag is set to validate (or not) the position information and whether the position is fixed or not. The parsing function checks this flag before storing the useful data. For instance latitude and longitude are considered as useful data. Once the data are stored they are available for high level programs. To ease the use of the driver, I write default handlers which print data on screen, via an UART communication (the printed data are for example the latitude and the longitude in degrees, minutes, seconds).

The figure 5 resumes the GPS data frame processing mechanism.

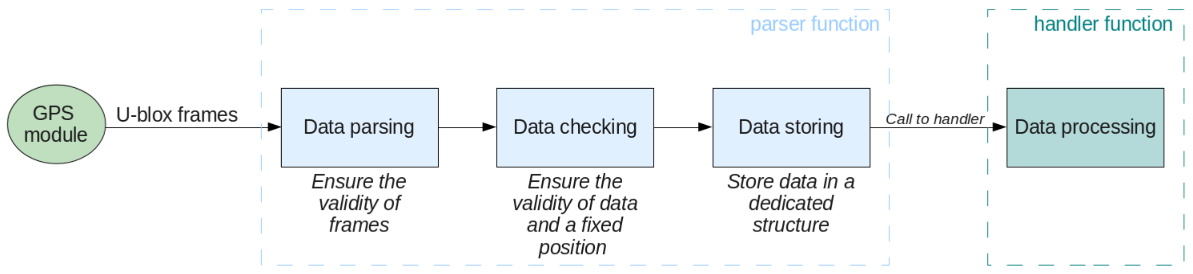


Figure 5: GPS data frame processing mechanism.

To simplify the use of the driver, default settings are set in the GPS initialization routine :

- the protocol is set to : $\mu - blox$ protocol
- the update frame rate is set to : 250ms
- the default handler is set to a handler printing most information available on a GPS data frame (namely : latitude, longitude, x,y,z coordinates and the time)

Of course, users can change these default settings using the appropriate driver configuration functions.

IV.2.2 The application part

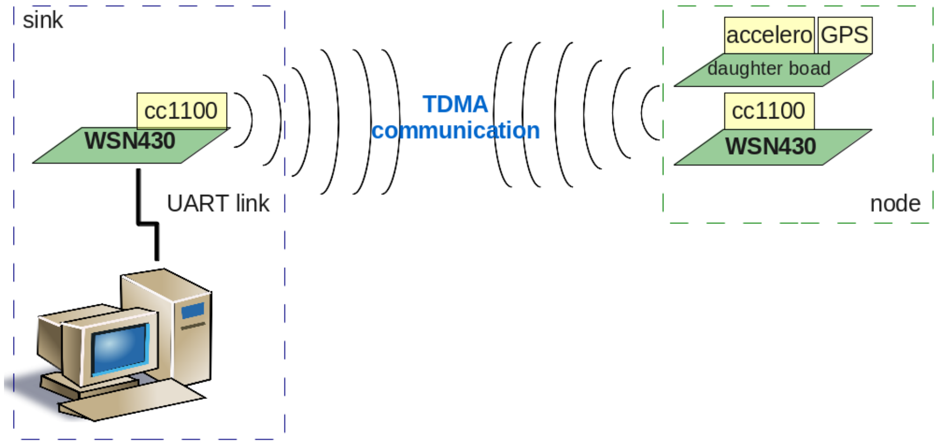


Figure 6: Application part overview.

I work on three different points :

- the implementation of the accelerometer sensor device;
- the discovery and understanding of FreeRTOS features (like the tasks scheduler, ...);
- the use of the TDMA wireless communication way. This step needed most care due to compatibility problems.

Accelerometer sensor

A driver designed for the accelerometer sensor used in the daughter board was already available when I started this internship. Thus, I work on integrating the accelerometer data polling to the GPS data frames retrieving test program. I ensure the compatibility of the two data polling (GPS and accelerometer) using the same i^2c bus.

FreeRTOS

As explained in section III.3 we choose the TDMA communication to exchange data between sensor node and computer sink (see figure 1). This communication protocol is already written, but it needs an operating system called FreeRTOS. This operating system is designed for real-time applications and is suitable for most embedded systems. Therefore, to be able to use the TDMA communication I need to design the application part over this operating system and use its features. I need to learn the main principles of FreeRTOS and how setting up applications using this OS. This required to analyse basic services provided by FreeRTOS such as task creation, to more elaborate ones as task scheduling or task heap [20].

Implementation of TDMA communication

This step is the more complicated I faced as I have to deal with compatibility problems between the GPS and the TDMA drivers.

First of all, as explained in the section III.3, the communication ports of the MSP430 micro-controller are limited. Thus, the GPS driver can not use the port 1 (the UART link) because wireless

[20]The heap is a memory space allocated to a task. The memory space is used by tasks to store local variables allocation and to realize other tasks needing no-persistent memory space.

chip driver needs port 1 as it uses the SPI bus. Therefore, the driver of GPS module needs to be set up as to use i^2c bus.

Secondly, as mentioned in paragraph *i^2c communication* in section IV.2.1, both FreeRTOS and the TDMA driver need timers. On MSP430 micro-controller only two timers are available, the first one is required by the FreeRTOS scheduler and the second one is used by the TDMA driver. Thus, i^2c data polling is no more possible using timer when implementing the application part. We decide to perform the polling via an over-layer instead. Then the driver can be used in any application on any operating system or on bare machine. Nevertheless, the designed over-layer can ease the driver utilization for applications using either FreeRTOS operating system or bare machine (difference is made via a dedicated compilation flag).

The version designed for bare machine uses a micro-controller timer to count elapsed time, and uses the strategy described in paragraph *i^2c communication* in section IV.2.1.

As for FreeRTOS applications, I used features released by the operating system to handle the periodic polling. The GPS initialization and polling are realized by a FreeRTOS task constantly active (run as a background task). This task, after μ-blox GPS module initialization, executes an infinite loop to handle the data polling. To measure precisely the elapsed time between two pollings, I use the delay feature of FreeRTOS. This function put a task to sleep during a predefined constant time. During this sleeping period other tasks, like accelerometer polling or TDMA data sending, can become active and use the processor resource. When the sleeping time of the GPS task run out, this one is awaken and is activated if its priority is higher than other runnable tasks.

At last, I had trouble with the TDMA link itself. TDMA communication needs ongoing exchange of synchronization beacons between nodes and the sink. Therefore, the TDMA is not only active when a data sending is required, but it is running all time in background and sometimes it needs processor resources to preserve the wireless link.

Therefore, when integrating the TDMA communication and the GPS driver, problems relative to i^2c access appear. The i^2c access can not be performed if TDMA beacons come up during this moment.

The easiest, but not working, solution is to mask all interruptions during the reading task. This should ensure that during the reading all processor resources are allocated to the reading and no other task can be active and run. This strategy does not work because the period of time interruption are masked is too long and many synchronization beacons can not be receive and are considered as lost by the TDMA driver. So the wireless link is always broken and lot of data are lost.

Finally, I succeed in handling the i^2c access, without damaging the TDMA link, modifying the tasks priorities. During the reading, the task priority is set to the maximum priority level. Thus, the TDMA reception interruptions can be raised and no beacons are lost. Nevertheless, the TDMA task can not be active and can not process the received data. The i^2c reading is not delayed anymore. Once the reading is fulfilled, the priority is reset to a lower level for the data parsing task, allowing the TDMA task to be active and run if wireless link needs to be preserved.

Now, the TDMA communication works well together with the GPS data frame polling.

V Validations tests and results

V.1 Validations tests

V.1.1 GPS accuracy characterization and driver validation

Regarding the μ-blox GPS sensor, two kinds of validation have been performed : the validation of the driver and the accuracy characterization of data provided by the GPS sensor.

Driver validation
The driver validation is divided in two steps. The first one is to ensure that configuration settings are well taken into account by the GPS module. To check this point, I try different settings parameter values and, I reread the values set in the “registers”. As for the frame sending frequency changes for example, I just control the effective rate modification.

To ensure that the driver can work for a long time, I check that data polling and data parsing can be performed many times without bugs. Like in many embedded systems, there is no choice to check the good working of programs, to run during a long time. Thus, if the GPS drivers run during one hour without bug, we consider the driver to be functional. This period of time was defined by my supervisors in relation to the driver application goal. The drone flying time being limited to 10 minutes long because of batteries), the GPS module would not be run more than an hour without being restarted.

Characterization of GPS measures accuracy
The characterization of the μ-blox GPS module implies to evaluate how accurate the measures provided by the GPS module are. Two type of characterization have been studied : the static and the dynamic characterization.

- The static characterization consists in computing the measure errors for the GPS module sets in a fixed position. The GPS antenna was laid on a characteristic point just front of the INRIA place to perform this characterization. I stored many position measures given by the GPS device and I calculate the standard deviation from the mean measure values. I use the projected values provided by the GPS module which are given in meter. The result is given in figure figure 7. The green line represents the mean value and the purple lines show the standard deviation and the blue line is the data.
  Thanks to these measures, I was able to calculate the static accuracy of the device : the standard deviation is $32cm$ for x axis, $19cm$ for y axis and $53cm$ for z axis. This accuracy seems to be sufficient enough to set up the control algorithms on drones.

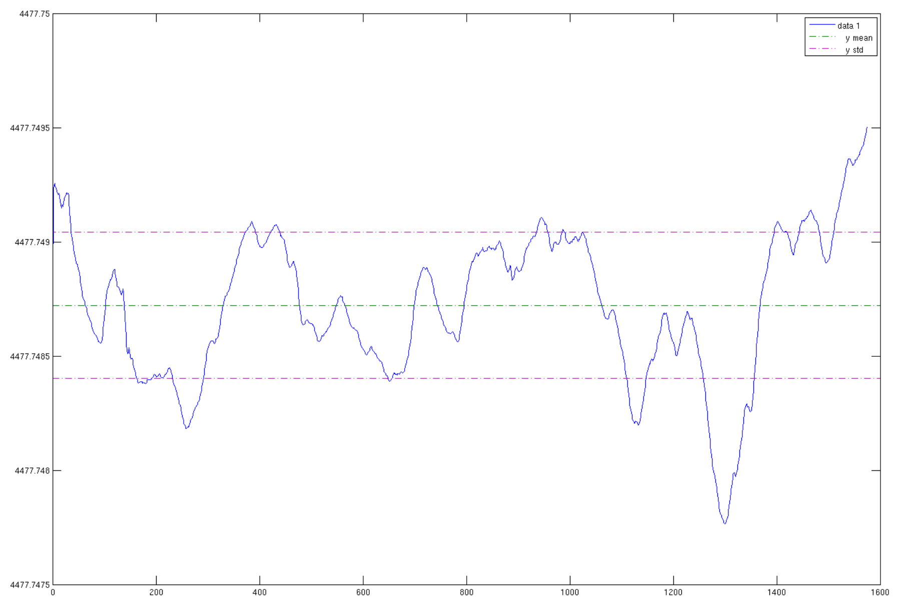


Figure 7: GPS static characterization result for 1500 points on x axis. (the ordinate axis scale is in meter).

- The dynamic characterization. I could not perform totally this characterization due to lack of time and resources. To characterize the dynamic performances of the device a faithful reference is needed. The measures of the reference are compared to the measurements of the μ-blox GPS sensor. A very accurate centimeter scale GPS (from Thales) is available at INRIA and can be used as the faithful reference. But the setting up of this device was complicated and I could not perform the comparison between the two GPS data measures. Therefore, to ensure that dynamic motions can be handled by the GPS sensor I perform some trajectories (on walking or on a mobile car robot) with the sensor, (like circular trajectory for example) and I checked with data plots that the sensor measures correspond to the trajectory.

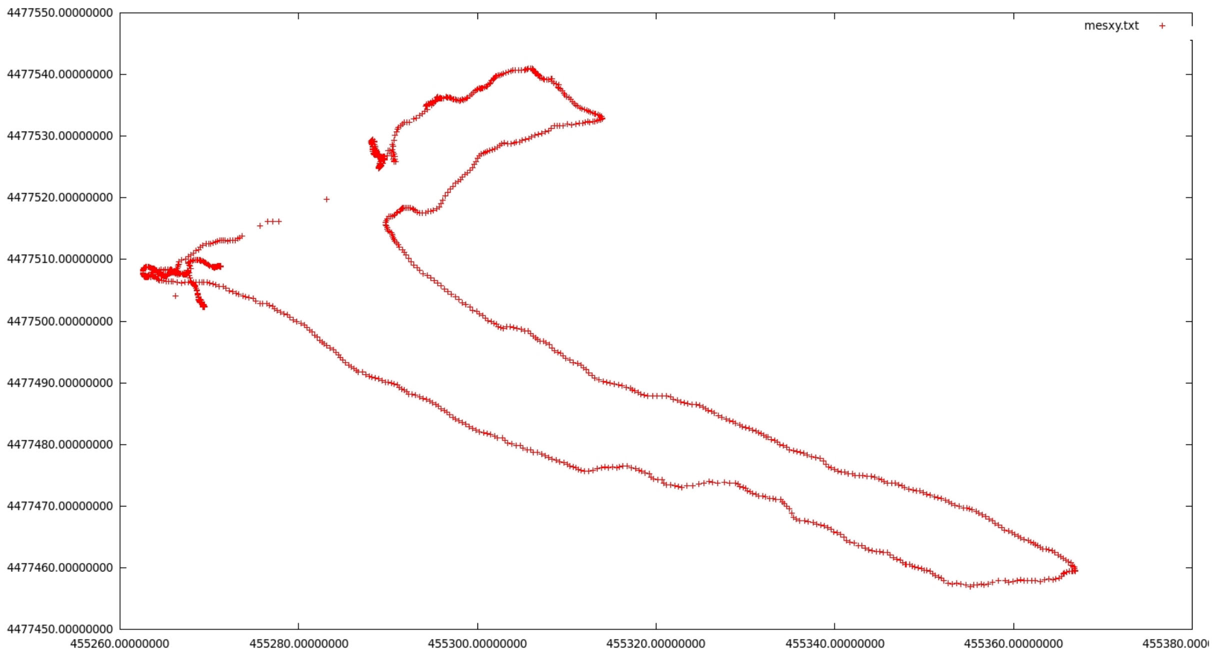


Figure 8: Illustration of dynamic characterization with a trajectory made by a mobile car robot.

V.1.2 Validation of the wireless communication

A characterization of TDMA performance has been realized. First of all, a verification of data transmission time has been performed checking the difference between the emission time and the reception time. We verify that the time spent between two receptions is very close to the time between two emissions. The second point is the characterization of TDMA maximum range. This has to be done because no faithful data about this range was available. To retrieve this information, I carry out the following test several times : I started the TDMA communication and I moved aside the node board and the sink board until frames were not received anymore. Then with the GPS position information I was able to measure a range value. Finally, in an outdoor environment without obstacles, the range for the TDMA communication is around $20m$.

V.1.3 Internship results

Concerning the hardware support, I make sure that the daughter board is functional and I find two hardware bugs :

- the first one is an inversion in serial UART link : the transmission wire and the reception wire are not reversed as it should be.
- the second one is related to the GPS antenna. The ground-plan (copper layer on the board) is not removed around the antenna as it should be. Thus, this ground-plan causes interference on antenna reception and no valid signal from satellite can be received. The copper layer needs to be removed before, to be able to receive satellite signals correctly via this antenna.

As for the software I implement, the performed work covers the two parts specified at the begin of the internship period : the μ-blox GPS driver and the node to sink communication application. A description of the realized software is given in appendix A.6. The figure 9 shows the whole data processing flow :

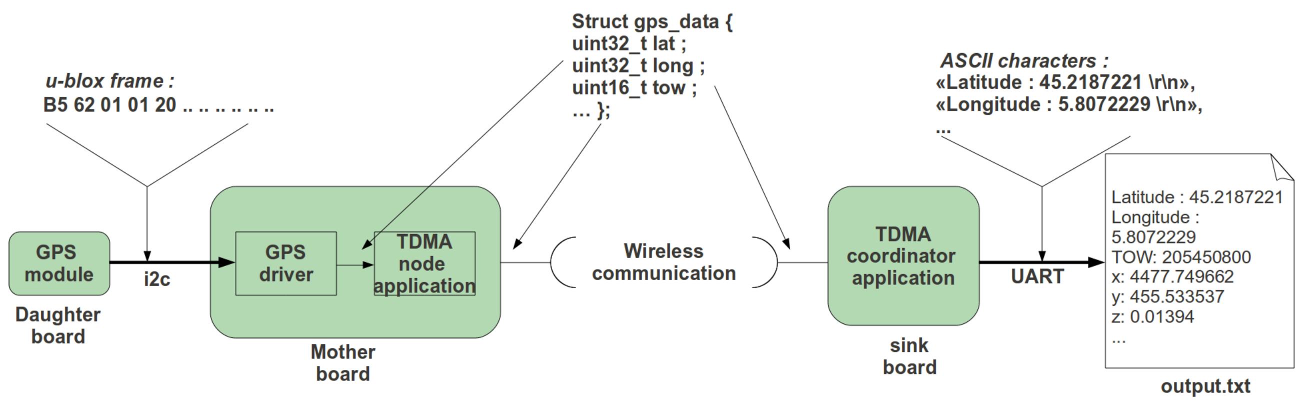


Figure 9: The whole data processing flow to retrieve the GPS data frames.

The figure 9 presents the different steps of the GPS data processing flow from μ-blox GPS module retrieving to data on sink computer. At sink computer the data can be are stored in a file when received from the sink board.

The work I realized uses at best the given resources. The μ-blox GPS driver manages the device features, exploiting two of the three communication bus available (i^2c, and UART) Moreover, most of all configuration settings are easily customizable through the appropriate driver API. The only characteristic the driver does not handle is the energy saving setting. Nevertheless, the μ-blox GPS module do not reply fully to expectations as its maximum frame rate is limited to 4Hz which is not enough to a enable a good drone control.

Concerning the TDMA communication, the data exchange works fine. However, this communication channel implies additional delays in the retrieving of the sensor data at the sink end. As the GPS frames can not be updated often enough to provide position information in " real time" with respect to the drone control, the accelerometer information could be make up for the GPS frames. Accelerometer data could be updated at frequency of 640Hz which is enough for drones motion capture but with TDMA communication latency, these data can not be send to sink end at this rate.

As detailed on appendix A.2, the delay implied by TDMA link depends on the number of slots, i.e. the number of drones flying. Thus the data exchange between a drone and the sink computer can only operate at a period of 45ms which can be slow comparatively to drone motions. To over pass the communication delay, and don't damage the communication, we finally decide to sample accelerometer data each 10ms and to send accelerometer data each 250ms at the same time as the GPS data. Nevertheless, having a sample of acceleration every 10ms an integration of these data can be performed and the lack of GPS data compensated.

But again, a problem persists. The sampling of accelerometer data can not be performed at a rate of 10ms because each 250ms GPS frames must be read on the same i^2c bus. The GPS data frame length (128 octets) being longer than accelerometer frame the reading of the GPS frame takes more than 10ms on i^2c bus (it takes approximately 26ms, see appendix A.1 for more details). So the lack of at least 3 accelerometer frames each 250ms should be taken into account at high level i.e. within the drone control algorithms.

Finally, the system designed is functional and fulfilled the main specifications given by my internship supervisors. At present time, on the sink computer two recovery data mode are enable. Both are based on a written program allowing the reading of the computer serial port [5].

- mode 1 : the drone motions are tracked and written in a file which can be easily plot with a script execution. The reading and the storing of the data, in a file, are performed in a main function.
- mode 2 : the position of the drone are integrated to the control program. Then, these data can be used to control the drone motions as wanted. In this mode, the reading and the storing of the data, in a appropriate data structure, are performed in a thread of the drone control program. Thus, the memory being shared between threads, the drone control thread can get the absolute position data in this structure and take it into account for the next commands sent to the drone.

As for the second mode, some control tests were performed, using a basic proportional corrector. Drone motions are commanded by these equations :

$$\begin{aligned} pitch &= Kp(Xdest - Xinit) \\ roll &= Kp(Ydest - Xinit) \end{aligned}$$

The pitch and the roll are the command values sent to the drone. (Xinit, Yinit) and (Xdest, Ydest) are respectively the coordinates of the drone takeoff point, and the drone destination point. The coordinates are represented in the x/y axis of the GPS module. At beginning, the drone must respected a predefined orientation. The front of the drone must be laid along the x axis. Thus, the x motions correspond to the pitch commands. So we managed to take into account the absolute positions in the drone controlling loop, and to move it to a predefined destination point. The system designed fulfills its main goal because it eases the control of the drone motions.

Currently only GPS data are available in the drone control program data. But, in a near future, an additional software should be designed on the sink part to integrate the GPS position and accelerometer data in the drone control motion program. A plot of a drone trajectory is represented in the the Figure 10. This plot presents the capability of the designed system to track the drone motions despite the relatively slowness of the GPS data updating.

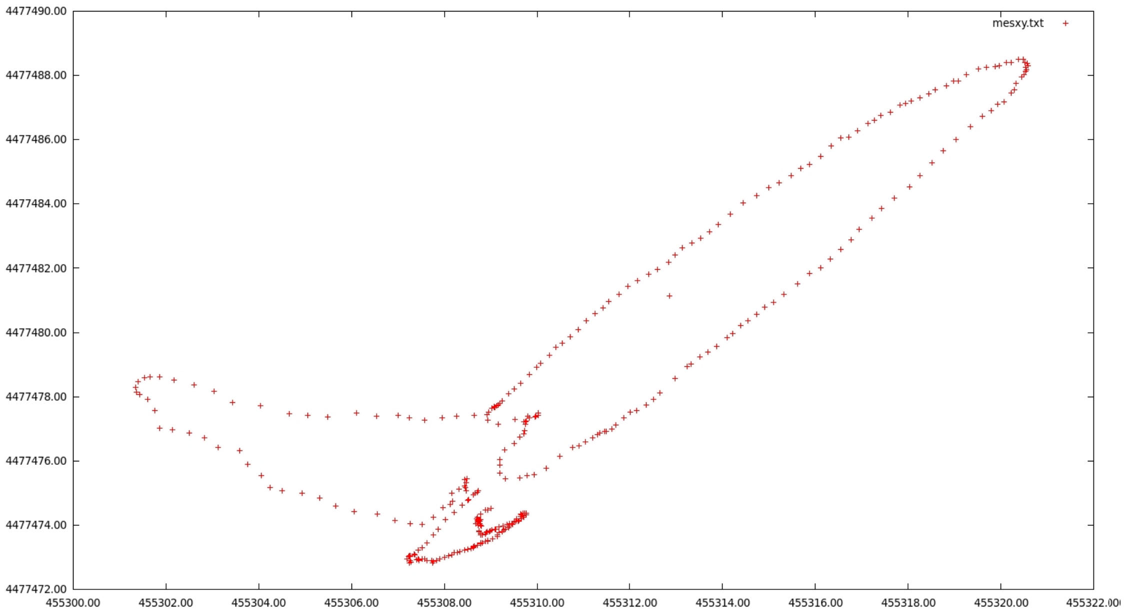


Figure 10: Illustration of a drone trajectory recorded by the designed system.

VI Balance sheet

Regarding the technical aspect of this project, the major difficulty encountered is the setting up of the wireless communication. As shown on the realization schedule in appendix A.5, this step takes me a lot of extra time compared to the scheduled task time (cf appendix A.4). As explained in IV.2.2, to reach an working wireless communication, the GPS driver needs to be re-written to use another communication medium (for the record, the change from UART to i^2c bus was required). This task was the hardest of the project because of problems bound to the i^2c bus were found and needed to be solved before working on wireless communication. Moreover, a lot of time was spend to find the best settings for the TDMA communication allowing a communication without loss.

Another technical problem I face during this internship is relative to the hardware support. At the beginning of the project, routing problems were persistent on the sensors board. And at first I lost much time thinking that if the driver worked wrong it was because of my program. But the problems came from the hardware board itself. Thus, I learn that no bug sources could be ignored, and, in project integrating hardware and software parts, bugs can come from anywhere.

This internship allows me to improve my knowledge about compilation issues. At Phelma school, not enough emphasis is given to compilation mechanisms (like writing makefiles). Most of time makefiles are already written for us. But thanks to help of my supervisors, I learn more about this issue and I work on another compilation tool, useful for cross platform [21] compilation, "cmake".

The code readability is another key point on this project and especially for the driver part. I am used to comment the code I write but I do not really care on the readability of the code. As the driver intent to be used by other people, the way the code is commented or formatted is very important. Particularly for an internship project I will not be able to perform the follow-up of the driver. Thanks to a real confrontation to a professional working environment I understand why coding rules and comments are important.

Another interesting point to this internship is the use of a real-time operating system, as I want to specialize on embedded system. I understand how real operating systems performed some specifics real-time features like the static scheduling for example.

This internship taught me the self-government and the capability to work alone on a software project. The autonomy given by my supervisors constrained me to more strictness in my work. My supervisors let me work as a junior engineer could be work in a society. Namely, they don't check the work provided. This way of proceed constrains to work more carefully. The work done will be used by other people as a faithful program. So the result need to be sure. I think it is a good preview of engineer duties.

[21] In computing, cross-platform, or multi-platform, is an attribute conferred to computer software or computing methods and concepts that are implemented and inter-operate on multiple computer platforms. For example, a cross platform application may run on Microsoft Windows on the x86 architecture, Linux on the x86 architecture and Mac OS X on either the PowerPC or x86 based Apple Macintosh systems.

VII Conclusion

The internship aims the integration of GPS and accelerometer sensor board for drone control motion. This project needs abilities in many areas ranging from electronics skills to operating system knowledge. Apart from the hardware support debugging, I work on designing a GPS driver and setting up an entire communication flow, with wireless data exchanges.

To design the GPS driver, I understand how the GPS module works. Then, I work on the driver implementation. I base part of my work on existing programs driving the micro-controller resources. Then, I set up a TDMA wireless communication using real-time operating system skills (*FreeeRTOS*). The wireless communication enables the recovery of data on a computer in charge to control drone. At last, I implement an accelerometer sensor, basing my work on an existing driver, to improve the motion tracking of the drone.

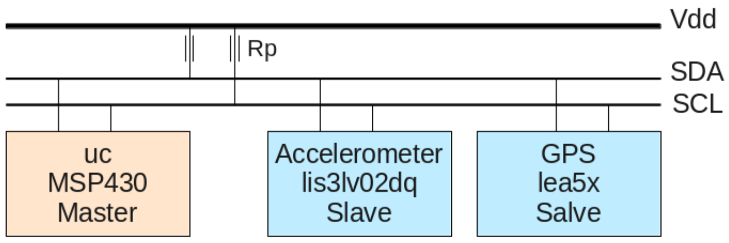
This project globally fulfills expectations of my supervisors. The robots platform needed by a research team can be set up. Thus, the control algorithms could be check on physical support.

To conclude, this 10 week internship was a good preview of the engineer work in a research laboratory. It shows me the kind of work asked to engineers in the low level software development. Now, I will able to make knowingly my career choice.

A Appendix

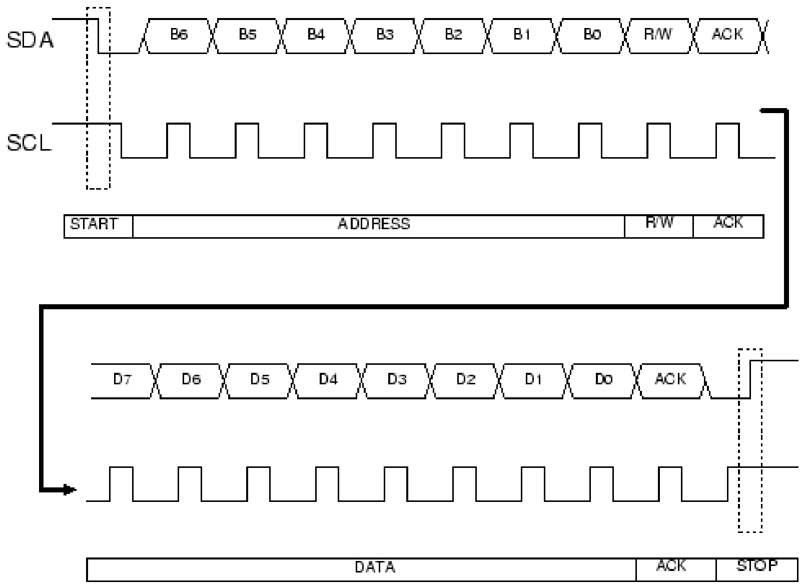
A.1 Inter-Integrated Circuit (i^2c)

The i^2c bus is a serial communication bus used to exchange data between two or more integrated circuits. Initially designed by *Philips*, the $\mathbf{i^2c}$ standard is, nowadays, one of the most used in electronics. Moreover, it revealed to be one of the easiest standard with only a two-wire interface.
The following figure presents how we use the i^2c bus in this project :



The figure A.1 shows the two-wire interface with the clock (SCL) and the data wire (SDA). In the i^2c standard the lines are in open-drain, which means that they need to be pull-up to the supply voltage to work, as in figure A.1 the use of “Rp” resistance.

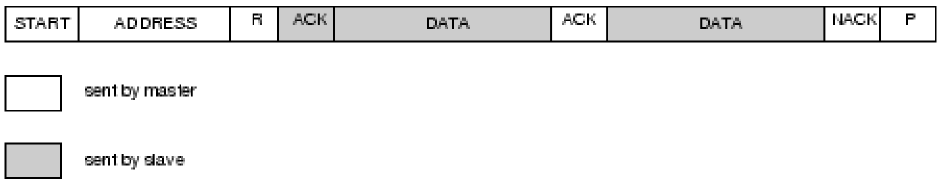
The standard i^2c protocol for data exchange is the presented in figure A.1.



A data sequence begins with a start bit and ends with a stop bit. Then, for each byte sent an acknowledge bit is waited for. Therefore, for each byte, 9 bit are really sent. The standard clock rate are 10kHz, 100kHz or 400kHz. Normally, basic system operates with clock ranging from 0kHz to 100kHz. For the communication with μ-blox GPS module, I set the clock rate to 70kHz being the highest frequency supported by the GPS device.
For the record, the length of data read on GPS device, when a polling is performed, is 128 bytes long. With a frequency set to 70kHz, the polling should last : $(128 \times 9) \times \frac{1}{70000} \approx 16.5ms$
In practice, we get 26ms. This is longer than the theoretical result for two main reasons :

- the acknowledgment bits are not really received instantaneously after the data transmission;
- the processor can execute another task between two bytes reading.

The figure A.1 presents data transfer mechanism in slave to master mode, as implemented for sensor data polling.



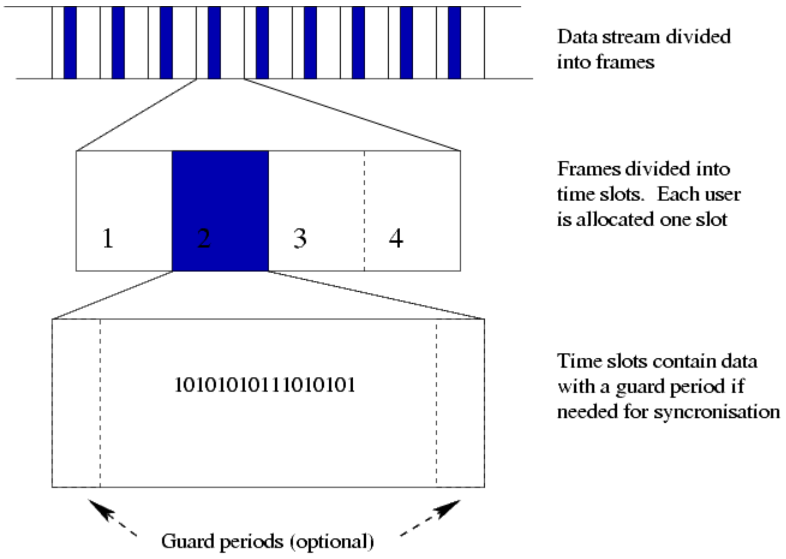
Concerning the reading and the writing of data on the i^{2}c bus, the basic command sequence is the following :

1. Send the START bit (S).
2. Send the slave address (ADDR).
3. Send the Read(R)-1 / Write(W)-0 bit.
4. Wait for/Send an acknowledge bit (A).
5. Send/Receive the data byte (8 bits) (DATA).
6. Expect/Send acknowledge bit (A).
7. Send the STOP bit (P).

Web source : http://www.best-microcontroller-projects.com/i2c-tutorial.html

A.2 Time Division Multiple Access (TDMA)

As explained in glossary, **TDMA** is an acronym for **T**ime **D**ivision **M**ultiple **A**ccess. This is a channel access method for shared medium networks. Several users are able to share the same frequency channel by dividing the signal into different time slots. In our case the frequency used by the radio chip is set to 868MHz. The figure A.2 represents the classical way to divided time into slots for TDMA standard.

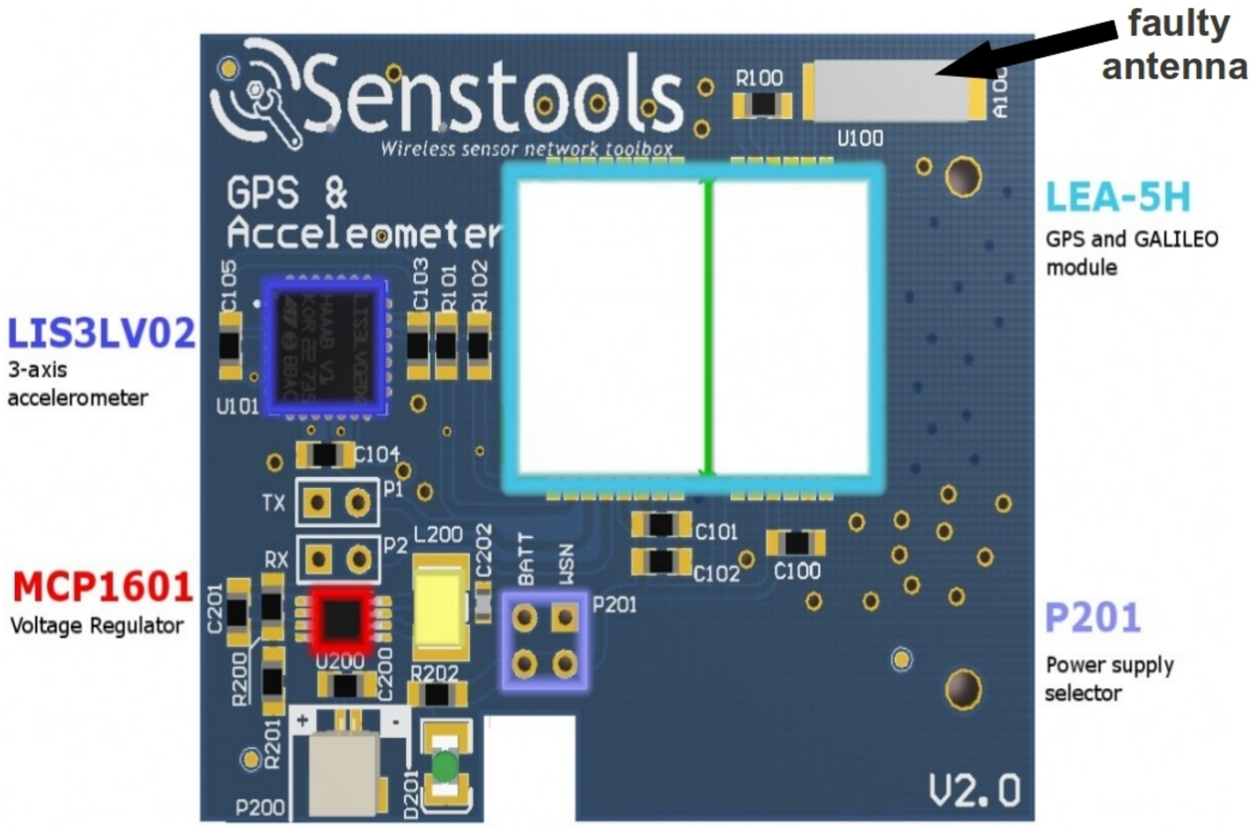

Data stream divided into frames
Frames divided into time slots. Each user is allocated one slot
1
2
3
4
10101010111010101
Time slots contain data with a guard period if needed for syncronisation
Guard periods (optional)

The TDMA driver designed by the "**S**ervice **E**xpérimentation et **D**éveloppement" implements globally the same strategy. However, in addition to node slots, represented above, another slot is added to ease the detection of a new node wanting to connect with the sink. Thus, if there is only a node and a sink, the number of time slot required is two ($= 1node + 1nodedetection$).
In the TDMA driver, the duration of each time slot is the same and can be defined by user. Nevertheless, the time slot duration can not be set to any value. The duration of a slot is related to the length of data to be sent. The default setting for a time slot is set to $15ms$ for 119 useful data bytes to be sent. Another parameter to take into account, when deciding the length of the packet to be sent is that all packets will be added extra start and stop bytes : synchronization bytes and header bytes. So it is important to optimize the number of useful bytes with regards to the sum of sent bytes. The default values of the TDMA driver being computed to get an optimal rate, I keep these values to set up the TDMA communication.

Last important point for the TDMA link : the number of time slots depends on the number of communication nodes. The more nodes, the more time slots. Thus, if there is a lot of drones intending to communicate with the sink, the delay between two time slots dedicated to the same node is increased. And the reactivity of the system can be altered.
Note : Presently, if a node intends to communicate with the sink via TDMA, the delay between two consecutive frames of the same node is $30ms$.

Web sources : http://en.wikipedia.org/wiki/Time_division_multiple_access and http://www.senslab.info

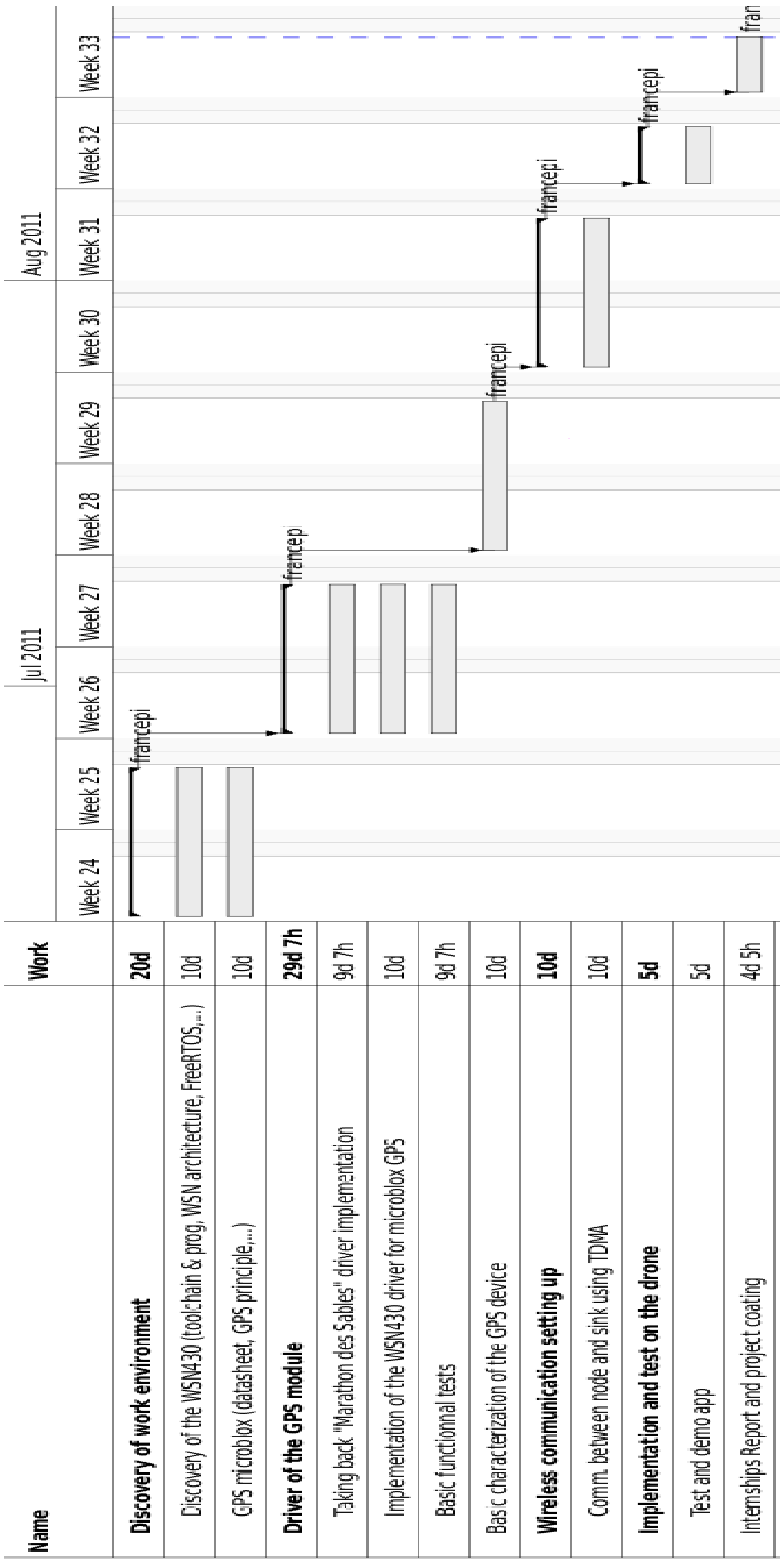
A.3 Daughter board overview



The figure A.3 presents the daughter board integrating GPS and accelerometer sensors.
This board is intended to be connected to motherboard WSN430. The defective antenna is the component located in the upper right corner of the board.

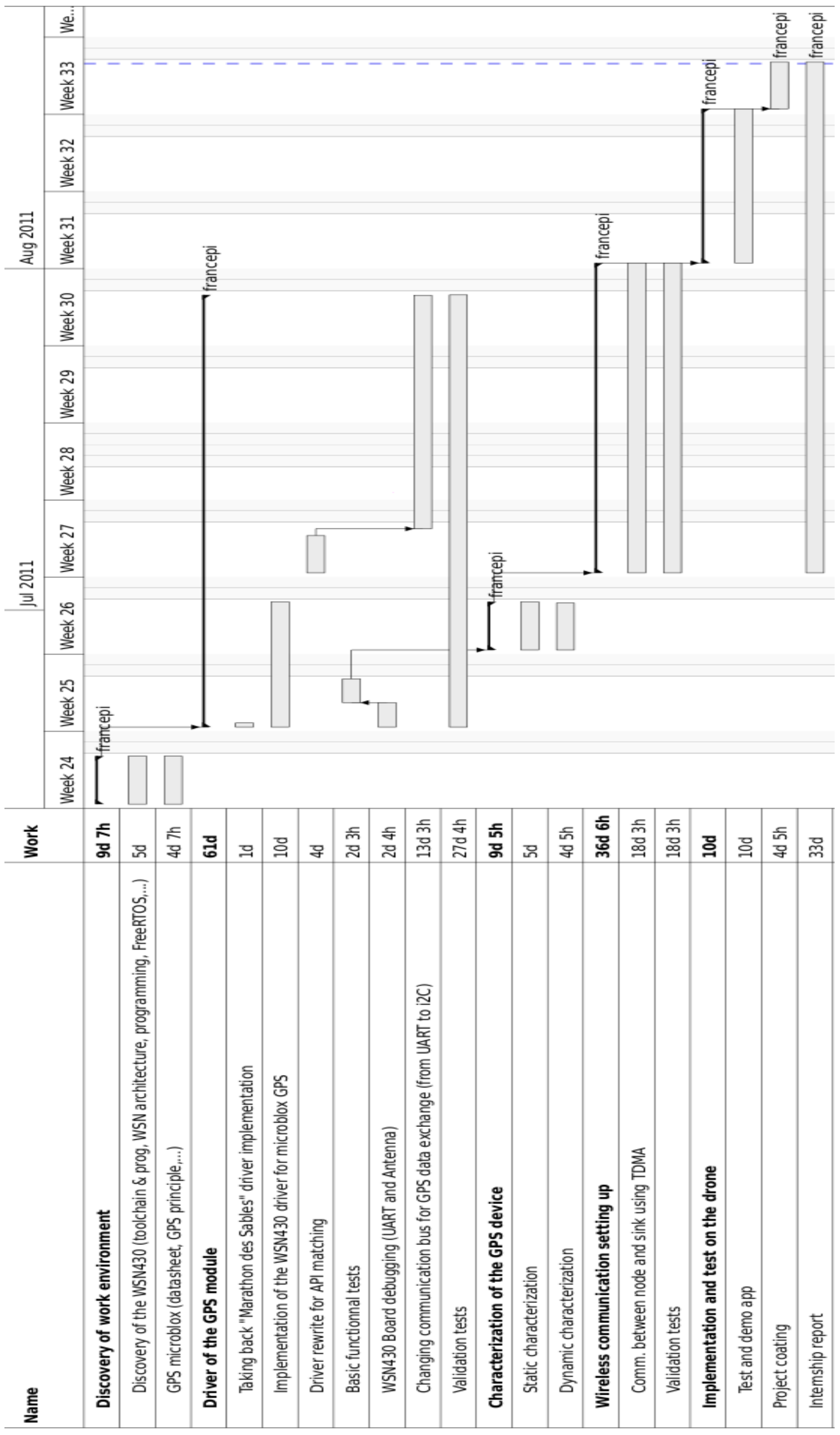
Web sources : http://www.senslab.info

A.4 Forward-looking Gantt diagram



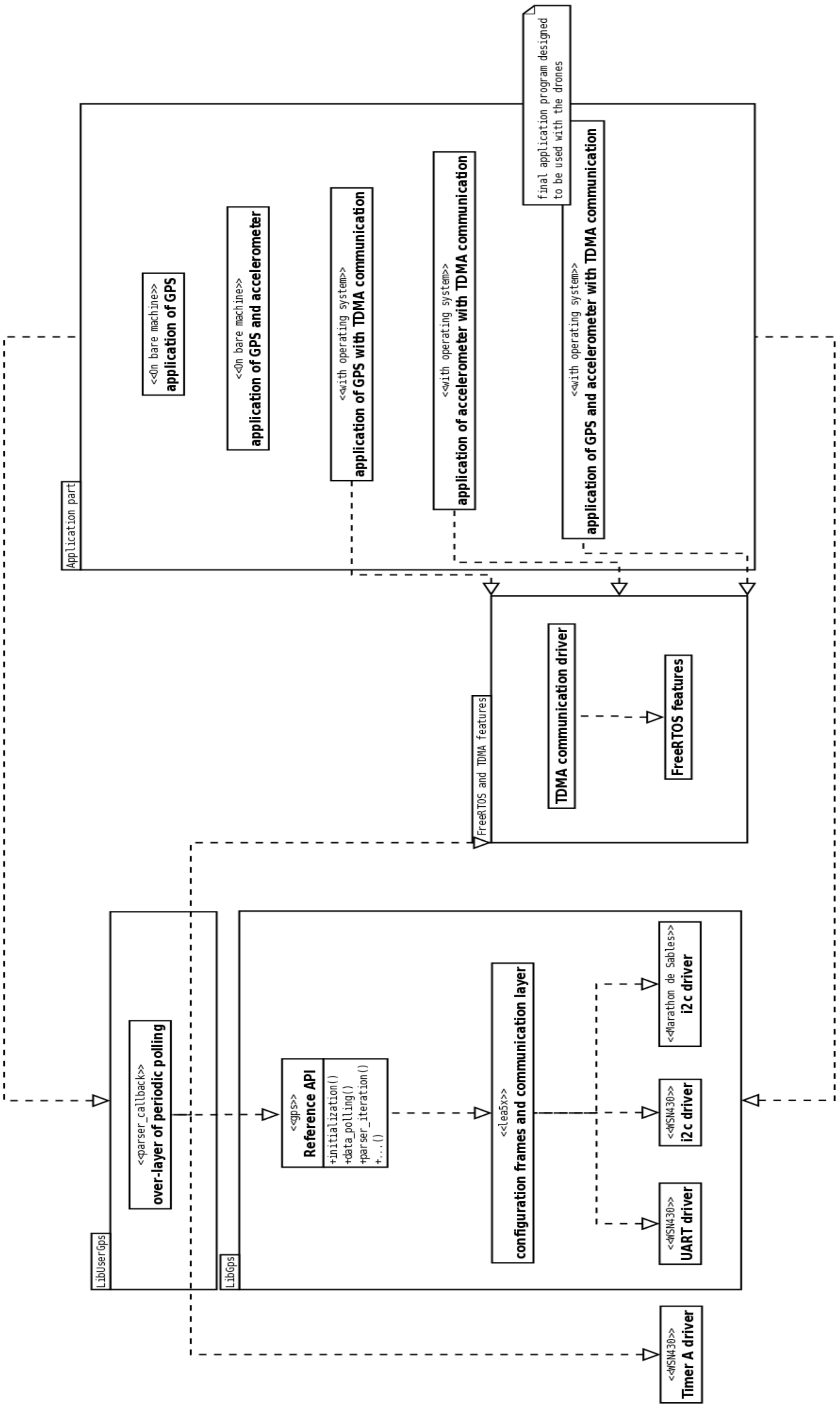
Name	Work
Discovery of work environment	**20d**
Discovery of the WSN430 (toolchain & prog, WSN architecture, FreeRTOS,...)	10d
GPS microblox (datasheet, GPS principle,...)	10d
Driver of the GPS module	**29d 7h**
Taking back "Marathon des Sables" driver implementation	9d 7h
Implementation of the WSN430 driver for microblox GPS	10d
Basic functionnal tests	9d 7h
Basic characterization of the GPS device	10d
Wireless communication setting up	**10d**
Comm. between node and sink using TDMA	10d
Implementation and test on the drone	**5d**
Test and demo app	5d
Internships Report and project coating	4d 5h

A.5 Realization Gantt diagram



Name	Work
Discovery of work environment	**9d 7h**
Discovery of the WSN430 (toolchain & prog, WSN architecture, programming, FreeRTOS,...)	5d
GPS microblox (datasheet, GPS principle,...)	4d 7h
Driver of the GPS module	**61d**
Taking back "Marathon des Sables" driver implementation	1d
Implementation of the WSN430 driver for microblox GPS	10d
Driver rewrite for API matching	4d
Basic functionnal tests	2d 3h
WSN430 Board debugging (UART and Antenna)	2d 4h
Changing communication bus for GPS data exchange (from UART to i2C)	13d 3h
Validation tests	27d 4h
Characterization of the GPS device	**9d 5h**
Static characterization	5d
Dynamic characterization	4d 5h
Wireless communication setting up	**36d 6h**
Comm. between node and sink using TDMA	18d 3h
Validation tests	18d 3h
Implementation and test on the drone	**10d**
Test and demo app	10d
Project coating	4d 5h
Internship report	33d

A.6 Overview of software realizations


Application part
<<On bare machine>>
application of GPS
<<On bare machine>>
application of GPS and accelerometer
<<with operating system>>
application of GPS with TDMA communication
<<with operating system>>
application of accelerometer with TDMA communication
<<with operating system>>
application of GPS and accelerometer with TDMA communication
final application program designed to be used with the drones
FreeRTOS and TDMA features
TDMA communication driver
FreeRTOS features
LibUserGps
<<parser_callback>>
over-layer of periodic polling
LibGps
<<gps>>
Reference API
+initialization()
+data_polling()
+parser_iteration()
+...()
<<lea5x>>
configuration frames and communication layer
<<Marathon de Sables>>
i2c driver
<<WSN430>>
i2c driver
<<WSN430>>
UART driver
<<WSN430>>
Timer A driver

The diagram A.6 presents the class diagram overview of the software implementation performed during this internship.
Arrow represents an implementation. The implementation has been coded in C language.
Only most important program files I write are presented in diagram A.6. Apart the timer A driver, the TDMA and FreeRTOS sources, all other files have either created or updated.
All written source codes are managed by a revision control system (subversion). Thus, they can be used and modified by other people working at INRIA.
The source code comments are compatible with doxygen (a documentation generator). So a documentation is available and eases the reuse of my work.

B Bibliography

1. **μ-blox GPS module :**
   - Web site : *www.u-blox.com/en/lea-5h.html*
   - Data-sheet :
     - LEA-5 μ-blox 5 GPS module: *GPS.G5-MS5-07026-B4*
     - μ-blox 5 receiver description : *GPS.G5-X-07036-G*
2. **mother board and daughter board :**
   - Web site : *http://www.senslab.info*
3. **TDMA :**
   - Web site :
     - *http://en.wikipedia.org/wiki/Time_division_multiple_access*
     - *http://www.senslab.info*
4. **i^{2}c bus :**
   - Web site : *Web source : http://www.best-microcontroller-projects.com/i2c-tutorial.html*
   - Data-sheet : MSP430x1xx Family : *SLAU049F*
5. **unix serial port opening and reading :**
   - Web site : *http://www.easysw.com/~mike/serial/serial.html*
6. **doxygen documentation of MSP430 drivers :**
   - Web site : *http://senstools.gforge.inria.fr/doxygen/main.html*
7. **words definitions :**
   - Web site : *http://en.wikipedia.org*
     - *http://en.wikipedia.org*
     - *http://www.senslab.info*

Glossary

API an **A**pplication **P**rogramming **I**nterface is a set of routines provided to simplify the use of a software or hardware build up by other programmers.

ASCII The **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange (**ASCII**) is a character-encoding scheme based on the ordering of the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes are based on ASCII, though they support many more characters than ASCII does.

CSMA **C**arrier **S**ense **M**ultiple **A**ccess (**CSMA**) is a probabilistic Media Access Control (MAC) protocol in which a node verifies the absence of other traffic before transmitting on a shared transmission medium, such as an electrical bus, or a band of the electromagnetic spectrum. "Carrier Sense" describes the fact that a transmitter uses feedback from a receiver that detects a carrier wave before trying to send. That is, it tries to detect the presence of an encoded signal from another station before attempting to transmit. If a carrier is sensed, the station waits for the transmission in progress to finish before initiating its own transmission. "Multiple Access" describes the fact that multiple stations send and receive on the medium..

driver A driver is a software application allowing to configure and to use easily an hardware device by programmers (or operating systems) through a given API (see the entry concerning API).

GPS **G**lobal **P**ositioning **S**ystem (**GPS**) is a space-based global navigation satellite system that provides location and time information in real-time.

micro-controller a micro-controller is a “small computer” on a single integrated circuit, containing a processor core, memory, and programmable input/output peripherals like analog-digital converters, serial ports communication,....

NMEA NMEA is data specification for communication between marine electronic devices such as echo sounder, sonars, GPS receivers and many other types of instruments. The NMEA standard uses a simple ASCII, serial communications protocol that defines how data is transmitted in a "sentence" from one "talker" to multiple "listeners" at a time. Here is an example of a NMEA GPS frame : $GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M„*76.

Operating System An **O**perating **S**ystem (**OS**) is a set of programs and data in charge of managing the computer hardware resources, such as memory, cpu, task scheduling and providing common services for software application to be run on the computer..

TDMA **T**ime **D**ivision **M**ultiple **A**ccess (**TDMA**) is a channel access method for shared medium networks. It allows several users to share the same frequency channel by dividing the signal into different time slots. see appendix A.2 for more elaborate description.