PROJECTE DE FI DE CARRERA

TÍTOL DEL PFC: Desenvolupament d'una eina Software per al control d'espectròmetres a través de dispositius PDA

TITULACIÓ: Enginyeria de Telecomunicació (segon cicle)

AUTOR: Núria Pujol Vilanova

DIRECTOR: Sergi Pons Freixes

SUPERVISOR: Rubén Quesada López

DATA: 25 de juny de 2008

Títol : Desenvolupament d'una eina Software per al control d'espectròmetres a través de dispositius PDA

Autor: Núria Pujol Vilanova

Director: Rubén Quesada López

Supervisor: Sergi Pons Freixes

Data: 25 de juny de 2008

Resum

Els oceans són un gran indicador de l'estat del planeta i la seva observació pot ajudar a comprendre i a prevenir alguns dels fenòmens d'abast mundial (ja ha quedat demostrada la relació dels oceans amb l'efecte hivernacle i l'escalfament global). Per aquest motiu, i des de fa anys, es realitzen nombrosos estudis arreu del món per tal de monitoritzar els oceans a través de l'adquisició de dades pel seu posterior processat.

Els sensors i dispositius utilitzats en l'adquisició de dades marines han anat evolucionant i oferint noves possibilitats i, en l'actualitat, són els dispositius òptics miniaturitzats els que semblen tenir més possibilitats en aquest camp, ja que permeten la obtenció de dades *in situ*.

Dintre d'aquest marc localitzem l'objectiu principal d'aquest projecte, que consisteix en el disseny i implementació d'un software per controlar el funcionament d'un espectròmetre miniaturitzat a través d'un dispositiu PDA. Degut a que el model de l'espectròmetre utilitzat és un dels sensors òptics capaç d'introduir-se en una sonda oceanogràfica més innovadors del mercat, l'aplicació desenvolupada és un primer pas per a l'obtenció d'una eina moderna per la monitorització del fons marí.

El projecte neix de la col·laboració amb la Unitat de Tecnologia Marina del Centre Mediterrani d'Investigacions Marines i Ambientals (CSIC) dintre de les activitats d'aquest grup en el desenvolupament d'instrumentació de monitorització del fons marí a través d'instrumentació òptica i constituirà una eina *free software* que podrà ser utilitzada lliurement per la comunitat científica.

Title : Spectrometers control software tool governed by PDA devices development

Author: Núria Pujol Vilanova

Director: Rubén Quesada López

Supervisor: Sergi Pons Freixes

Date: 25th June 2008

Overview

Oceans are great estate of the Planet indicators and their observation can help us to understand and prevent some worldwide phenomena (their effect on Global Warming and Climate Change is already proved). For this reasons, many researching studies related with sea data acquisition and processing have been done for many years around de world.

Sea data acquisition sensors and devices have been evolving offering new characteristics and, nowadays, miniaturized optical sensors seems to be the most promising sensors in this area because of their *in situ* data acquiring possibilities.

Considering this context, the project main objective consist in design and develop a miniaturized spectrometer software control tool using a PDA device to govern it. Miniaturize spectrometer model used is one of the most innovative sensors of this characteristics able to be introduced in a oceanographic probe and, for this reasons, the developed application can be considered a first step to getting a modern deep sea monitoring tool.

This project born in collaboration with the 'Unitat de Tecnologia Marina' of 'Centre Mediterrani d'Investigacions Marines i Ambientals' (CSIC) under sea monitoring instrumentation development activities of this research group with the purpose to be a free software tool available to scientific community.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

INTRODUCCIÓ

Els oceans són un gran indicador de l'estat del planeta i la seva monitorització pot ajudar a comprendre i inclús a prevenir alguns fenòmens que es produeixen a escala mundial. Des de fa molt anys, l'estudi dels oceans a través de l'adquisició de dades i el seu processat s'ha convertit en l'objectiu de molts grups d'investigació arreu del món i en molts casos, també han anat desenvolupant aparells cada cop més sofisticats per aquest fi.

Des de fa uns anys, l'interès pel desenvolupament d'eines per monitoritzar el medi marí també és objectiu de grups de recerca de casa nostra, com és el cas de la Unitat de Tecnologia Marina del Centre Mediterrani d'Investigacions Marines i Ambientals. Aquesta Unitat té com a objectiu, entre d'altres, el de crear eines pròpies d'observació del oceans, dins de les seves humils possibilitats, per no haver de dependre exclusivament de desenvolupadors externs.

Aquest projecte neix de la col·laboració, des de fa alguns anys, d'alumnes de la Universitat Politècnica de Catalunya amb el grup de Recerca i Desenvolupament de la Unitat de Tecnologia Marina del CMIMA-CSIC i de l'interès d'aquest per desenvolupar eines pròpies en el camp de la instrumentació òptica i l'adquisició de dades en general. Concretament aquest projecte té com a objectiu el disseny i implementació d'una eina software pel control d'un espectròmetre miniaturitzat a través d'un dispositiu PDA.

L'estructura del document es divideix en quatre capítols dedicats a conceptes previs, a l'entorn de desenvolupament, el software de control i les conclusions, respectivament. El capítol de conceptes previs tracta tots aquells conceptes necessaris per tal de posar en situació al lector i ajudar a entendre millor els objectius del projectes. El capítol dedicat a l'entorn de desenvolupament no només descriu els elements hardware o software sinó també els conceptes teòrics i pràctics que s'han utilitzat en el seu desenvolupament, deixant exclusivament la part de desenvolupament software englobada en el tercer capítol. Finalment, en les conclusions s'exposa el resultat al qual s'ha arribat i les seves possibles millores.

CAPÍTOL 1. CONCEPTES PREVIS

1.1. Els sensors òptics en l'estudi del medi marí

La major part de la superfície terrestre està coberta d'aigua, un 70% aproximadament. Tenint en compte aquesta dada, és lògic pensar que l'estat dels oceans i les seves possibles alteracions estiguin estretament lligades amb els fenòmens que tenen lloc arreu del planeta, com ja ha quedat demostrat en el cas del canvi climàtic i l'efecte hivernacle. Degut a la importància de l'estudi del fons marí, han sigut molts els projectes que s'han desenvolupat i que es segueixen desenvolupant amb l'objectiu de monitoritzar els processos que hi tenen lloc. Molts d'aquests projectes han sigut possibles gràcies a l'avenç tecnològic de les últimes dècades en els camps de la instrumentació, la computació, el processat i les comunicacions que han permès disposar d'eines i dispositius avançats per realitzar aquest tipus de tasques. Pel que fa a la instrumentació òptica utilitzada actualment, la podem classificar en dos grans grups: els sistemes d'adquisició remota o teledetecció i els sistemes d'adquisició *in situ*.

Com el seu propi nom indica, la teledetecció consisteix en l'adquisició de dades d'una superfície sense que la instrumentació hi entri en contacte i amb la possibilitat de fer l'adquisició a molta distància, per tant, molt útil en entorns inaccessibles. La teledetecció òptica permet obtenir informació de la composició de l'entorn estudiat a través de l'adquisició de dades de radiació solar reflexada i/o dispersada. Aquests sistemes poden ser integrats en satèl·lits o en avions i s'utilitza un cas o un altre segons les dimensions de l'àrea a estudiar i la resolució desitjada. Aquestes tècniques però, degut a la naturalesa atenuadora de l'aigua sobre les senyals electromagnètiques (radiació solar), només permeten l'estudi de les capes oceàniques més superficials, el que suposa una gran limitació en l'estudi del fons marí.

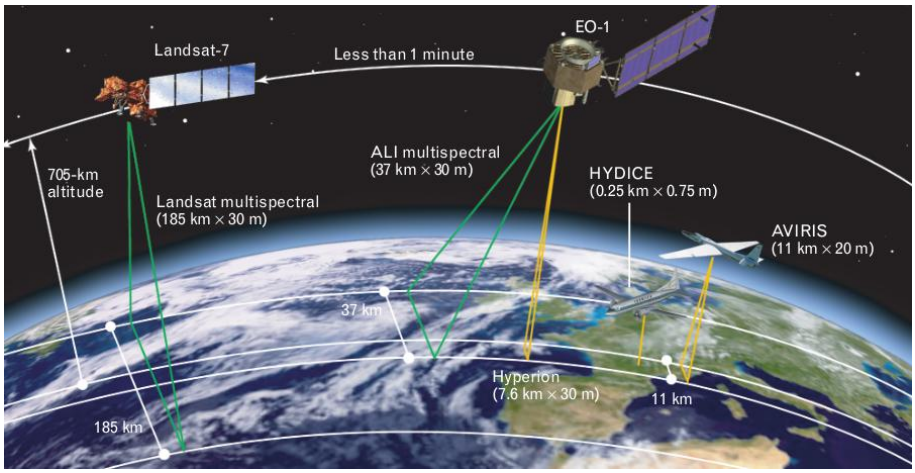

Figura 1.1: Diferents sistemes de teledetecció en relació a la seva resolució.

Una altre grup de tècniques molt utilitzades en la caracterització dels oceans, i que supleixen les limitacions de la teledetecció, són els sistemes de detecció *in situ*. Aquests sistemes permeten caracteritzar la columna d'aigua a través de l'adquisició de mostres en

temps real amb l'acoblament d'instrumentació òptica en vaixells, sondes i vehicles subaquàtics.

Figura 1.2: Diferents sistemes d'adquisició amb instrumentació òptica.

Entre els centres d'investigació que realitzen estudis utilitzant aquest tipus d'instrumentació trobem la Unitat de Tecnologia Marina (UTM), unitat adscrita al Centre Mediterrani d'Investigacions Marines i Ambientals (CMIMA), pertanyent a l'àrea de Recursos Naturals del Consell Superior d'Investigacions Científiques (CSIC). Dintre de les activitats realitzades per aquesta Unitat trobem les de suport logístic i tècnic a les bases polars i vaixells de recerca oceanogràfica espanyols, així com d'altres relacionades amb activitats de recerca, desenvolupament i innovació en el camp de les ciències marines.

Figura 1.3: Estudis realitzats per la UTM amb la utilització de sensors òptics.

Cal remarcar també l'interès institucional en la necessitat de promoure el desenvolupament tecnològic en el camp de la investigació marina a través de la, recentment implantada, nova línia de Desenvolupament de sensors automàtics per a l'observació dels oceans dins el Pla Estratègic de l'àrea de Recursos Naturals del CSIC.

1.1.1. L'espectrometria, els espectròmetres i la signatura espectral

Una de les tècniques òptiques *in situ* utilitzada en els centres d'investigació marina és l'espectrometria. L'espectrometria és un tècnica que permet l'estudi de la composició d'un cos o medi a través de les seves característiques òptiques (adsorció i/o reflexió de les diferents longituds d'ona que componen la llum incident).

En el cas dels estudis marins, aquesta tècnica s'utilitza en la detecció dels diferents compostos presents a l'aigua. Això permet detectar contaminants i concentracions d'organismes perjudicials per l'explotació pesquera (com per exemple la proliferació d'algues tòxiques que afecta a la producció de musclos a la zona del Delta de l'Ebre) d'una manera senzilla i gens invasiva pel medi.

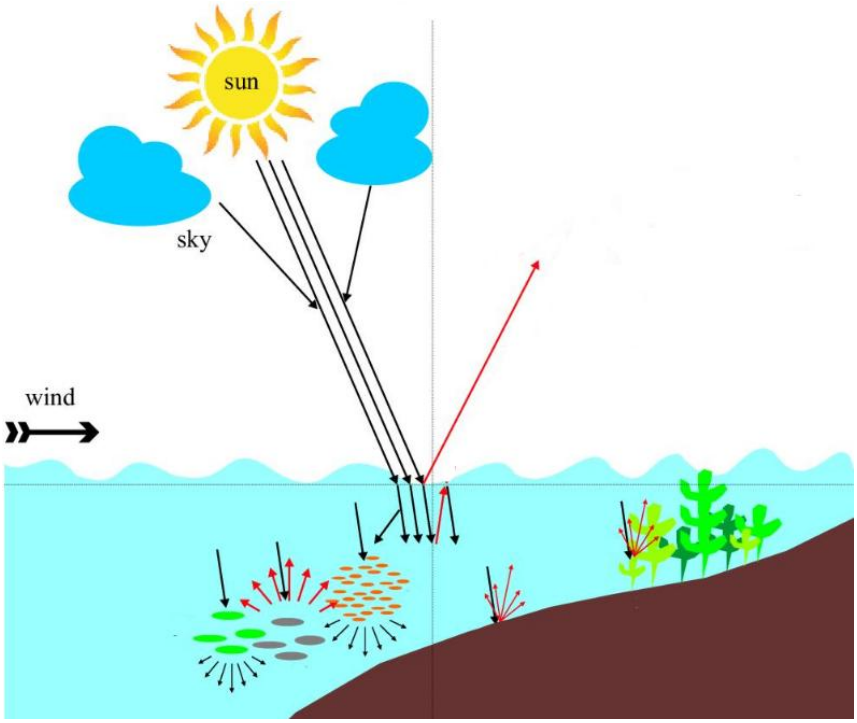

Figura 1.4: Reflexió i refracció de la llum solar en el medi marí.

Els sensors utilitzats en l'espectrometria per tal de quantificar les característiques òptiques del medi d'interès són, naturalment, els espectròmetres. Amb la miniaturització d'aquests dispositius és quan l'espectrometria comença a aplicar-se de manera més estesa i en nous camps, gràcies a les noves possibilitats que implica la seva portabilitat. La possibilitat de comptar amb espectròmetres de petites dimensions fa possible la seva inserció en sondes oceanogràfiques (entre d'altres aparells utilitzats en les introspeccions marines) afegint informació a altres mesures típiques de caracterització *in situ*: pressió, temperatura, salinitat, entre d'altres.

L'estructura bàsica dels espectròmetres miniaturitzats està formada per una entrada òptica on la llum incident queda concentrada, elements òptics que la descomponen en les diferents longituds d'ona i una *array* de fotodíodes (fotodetectors) en que cada fotodíode mesura la intensitat del senyal en un rang determinat de longitud d'ona. A la sortida de cada fotodíode s'obté un voltatge proporcional a la intensitat de senyal captada en cada rang.

El nombre de fotodíodes, les característiques òptiques dels materials utilitzats i les característiques dels elements elèctrics en determinen la sensibilitat, el rang de longituds d'ona mesurables i el temps de resposta (relacionat amb la freqüència mínima de mostreig) i per tant, l'escenari d'aplicació.

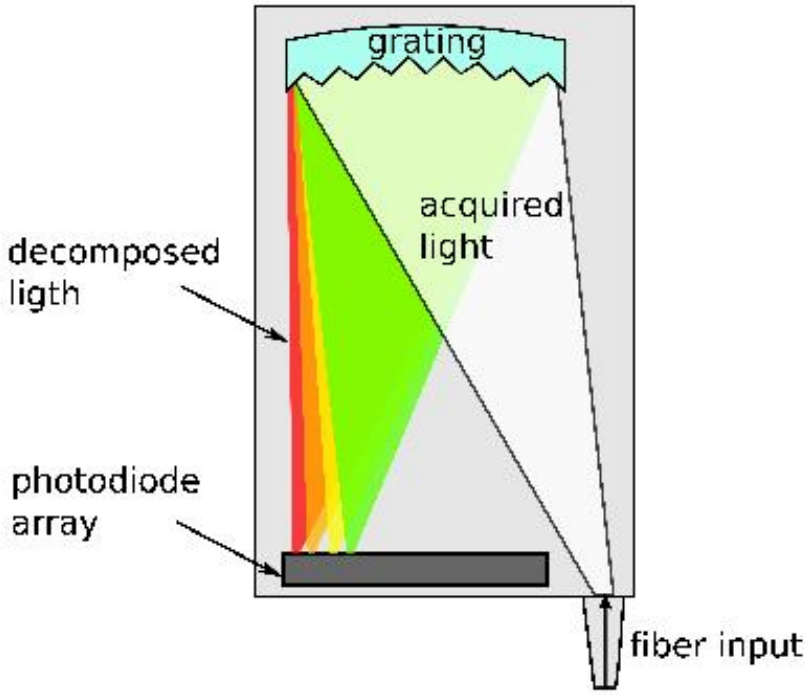

Figura 1.5: Estructura òptica bàsica d'un espectròmetre miniaturitzat.

Basats en aquesta estructura trobem espectròmetres senzills que necessiten d'altres elements per ser controlats (microcontrolador, filtres, A/D, ports de comunicació, etc.) i poder obtenir les dades en format útil, i d'altres en que tota aquesta electrònica ja ve encapsulada i ja estan preparats per la escriptura i/o lectura de dades a través de ports RS232 i/o USB. Tot i la diferència de preu d'aquest últims dispositius (d'un ordre de magnitud, aproximadament), són fàcilment integrables i els més indicats si es volen obtenir resultats ràpids sense necessitat de realitzar desenvolupament electrònic.

Fins ara s'ha parlat sobre els espectròmetres i les seves característiques però no s'ha entrat en detall sobre l'obtenció d'informació útil d'aquests dispositius ni com aquesta és interpretada. La manera més ràpida d'obtenir informació de les dades adquirides pels espectròmetres és representar-les gràficament, obtenint el que s'anomena la signatura espectral.

La signatura espectral és la representació gràfica de la relació de longituds d'ona més o menys absorbides i reflexades pel cos o el medi tenint com a referència la llum incident sobre ell, provinent d'una llum artificial en el cas d'adquisicions realitzades a gran profunditat. La relació d'aquestes longituds d'ona és característica de cada compost i permet obtenir informació sobre les substàncies dissoltes en l'aigua marina de forma quasi immediata i sense necessitat de realitzar un anàlisis químic en un laboratori.

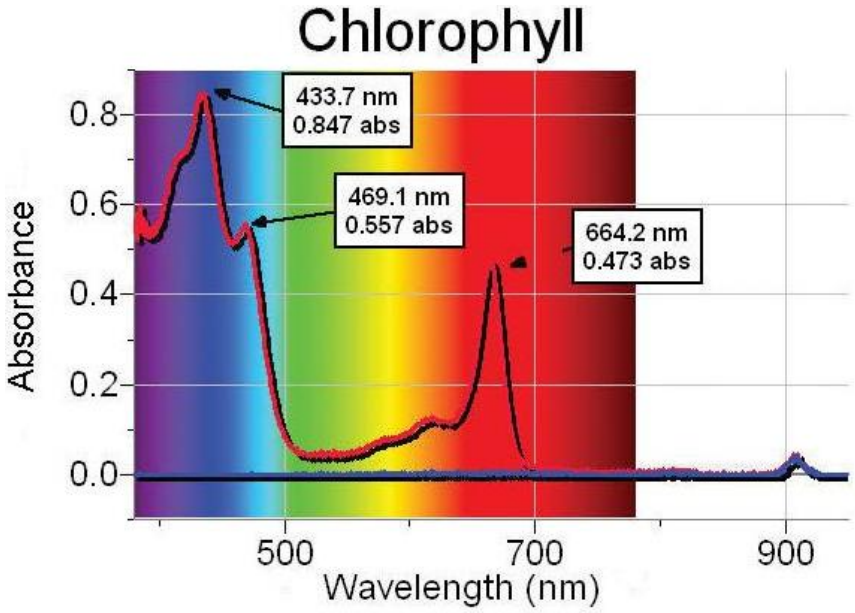

Figura 1.6: Signatura espectral de la clorofil·la.

Els perfiladors són dispositius que utilitzen espectròmetres, entre d'altres sensors, per la caracterització de la columna d'aigua que travessen. En el cas de l'adquisició de dades espectromètriques s'acostumen a utilitzar dos espectròmetres, un captant la llum incident en posició ascendent i l'altre en posició descendent. La senyal d'interès s'obté de la comparació de la senyal captada pels dos sensors, obtenint així un sistema menys sorollós i més sensible.

1.2. Característiques i objectius del projecte

En els últims anys, la UTM ha desenvolupat diferents projectes enfocats al disseny de sistemes d'obtenció de dades espectromètriques utilitzant part de la gran varietat de dispositius disponibles en el mercat. Recentment, ha adquirit diversos espectròmetres miniaturitzats del model USB4000 d'Ocean Optics, un dels models integrats actualment més innovadors per les seves petites dimensions, pes i sensibilitat.

L'objectiu principal del projecte es centra en el disseny i desenvolupament d'un sistema de control del dispositiu USB4000 a través d'un dispositiu PDA. Es pretén desenvolupar una eina software per l'adquisició de dades (adquisició, representació i emmagatzemament de dades), juntament amb el desenvolupament de les llibreries genèriques pròpies del control del dispositiu USB4000 per poder ser reutilitzades en altres aplicacions.

Com ja s'ha comentat, el sistema d'adquisició té la peculiaritat de ser controlat per un dispositiu PDA. Aquesta característica ha sigut un requeriment des del principi, ja que en un futur el sistema desenvolupat es vol introduir en una sonda oceanogràfica, sent necessària una eina apropiada pel control abord d'un vaixell, sota condicions ambientals adverses i a la vegada funcional. Amb aquesta intenció la UTM va adquirir el dispositiu PDA model ReconX de TDS, utilitzat en el projecte. Així doncs, aquest projecte constitueix

una de les primeres proves amb aquest tipus de dispositius des de la seva adquisició.

Amb l'escenari de desenvolupament determinat des del principi, l'objectiu és controlar l'espectròmetre USB4000 a través d'una PDA utilitzant una interfície gràfica dissenyada especialment per aquest dispositiu. La interfície gràfica i el software desenvolupat s'han dissenyat seguint les següents característiques:

- Funcionament senzill, amigable i intuïtiu per poder ser utilitzada per qualsevol personal científic sense coneixements previs del funcionament del dispositiu USB4000.
- Realització de tasques de control d'una manera totalment transparent per l'usuari.
- Modulabilitat del codi per facilitar la seva adaptabilitat a futures modificacions i noves funcionalitats.
- Control visual en temps real de les dades adquirides.
- Emmagatzemament opcional de les dades adquirides en format NetCDF per poder ser tractades per altres aplicacions.

A part de l'objectiu principal, el projecte té altres aspectes d'interès per la UTM en el desenvolupament de futures aplicacions. A continuació s'enumeren aquests objectius específics:

- Configuració i control del dispositiu USB4000 a través del port USB.
- Avantatges i desavantatges de la comunicació USB com a possible alternativa a la comunicació sèrie.
- Estudi sobre l'existència de software lliure i la seva utilització en el desenvolupament d'aplicacions per dispositius PDA.
- Desenvolupament de llibreries específiques pel control de l'espectròmetre USB4000.
- Creació d'un estàndard per l'emmagatzemament de dades espectromètriques en format científic i/o proposar unes primeres pautes.

A més a més de les especificacions i objectius esmentats, cal afegir que el projecte s'ha desenvolupat utilitzant únicament eines de software lliure, seguint la filosofia impulsada per la UTM en els últims anys, i que inclús el propi projecte constitueix per si mateix una eina de software lliure.

Com es comentarà més endavant, a l'hora d'escollir un llenguatge de desenvolupament s'ha optat pel llenguatge Python, un llenguatge en expansió, intuïtiu i estructurat per mòduls. La utilització d'aquest llenguatge ha suposat el seu aprenentatge en paral·lel amb el desenvolupament software.

1.3. Interès científic de l'aplicació

Tot projecte relacionat amb l'estudi del medi marí i la monitorització dels seus processos és ara més que mai de gran interès científic. També la introducció d'espectròmetres en sondes i vehicles oceanogràfics obre noves possibilitats en l'estudi del fons marí (entre d'altres àmbits) i alguns grups d'investigació ja estan desenvolupant aplicacions al respecte arreu del món. Aquest nou interès per les mesures espectromètriques *in situ* queda palès amb la introducció al mercat de nous models d'una forma ininterrompuda ens els últims anys, tot i ser un mercat encara molt reduït i especialitzat.

Aquest tipus d'aplicacions permeten obtenir informació de la composició de l'aigua a diferents profunditats a temps real (encara que després hagi de ser processada) quan, fins no fa gaire, aquest tipus d'informació només podia ser obtinguda a través de l'estudi en laboratoris de l'aigua recollida en les campanyes i per tant, eren necessaris molts més recursos i sobretot temps. Així doncs, els espectròmetres miniaturitzats permeten l'adquisició de dades *in situ* i faciliten que les dades recollides no hagin pogut ser alterades (dades més reals), obtenint resultats d'una manera més ràpida. Per tant, l'aplicació desenvolupada és un primer pas per a l'obtenció d'una eina moderna (utilitza un dels sensors més innovadors del mercat) per la monitorització del fons marí.

El projecte actual constitueix per si sol una eina *free software* de control pel dispositiu USB4000. Avui en dia només existeix una altre eina de control per aquest dispositiu que el fabricant distribueix amb el producte, previ pagament d'una llicència. Aquest software, propietat d'Ocean Optics, està pensat per executar-se en un PC, és poc adaptable a les necessitats de l'usuari i bàsicament està pensat com a eina de laboratori. En el nostre cas, el software desenvolupat podrà ser utilitzat lliurement per la comunitat científica i inclús pot ser el punt de partida d'altres projectes, respectant les condicions de la llicència de distribució.

1.4. Antecedents

Com ja s'ha comentat en apartats anteriors, la UTM amb col·laboració d'estudiants de la Universitat Politècnica de Catalunya, ha realitzat diferents projectes d'investigació amb la finalitat de desenvolupar eines de control de dispositius òptics, emmarcats en projectes finals de carrera de les titulacions tècniques i segon cicle d'Enginyeria de Telecomunicacions (EPSC).

A continuació es fa un recull cronològic dels projectes desenvolupats més rellevants, tenint en compte la seva finalitat i les millores sobre projectes anteriors:

- **Setembre 2003 – Gener 2004:**
  Títol: *'Disseny d'un sistema de mesures espectromètriques'* (TFC)
  Autor/s: Manel Merchán i Sergi Pons

Es tracta del primer d'una sèrie de projectes amb l'objectiu de dissenyar un espectroradiòmetre (sistema de mesures de dades espectromètriques) amb el que es va aconseguir realitzar un primer prototip.

En aquest primer prototip es va utilitzar un espectròmetre senzill i es va dissenyar i desenvolupar tota l'electrònica necessària per tal d'obtenir-ne dades útils. L'espectròmetre era controlat per un microcontrolador PIC18 capaç de mostrejar les dades utilitzant el seu AD intern de 10 bits que posteriorment eren emmagatzemades en una memòria EEPROM externa i/o representades en una interfície gràfica d'un PC (prototip i PC comunicats a través del port sèrie) desenvolupada en Matlab pels propis autors del projecte.

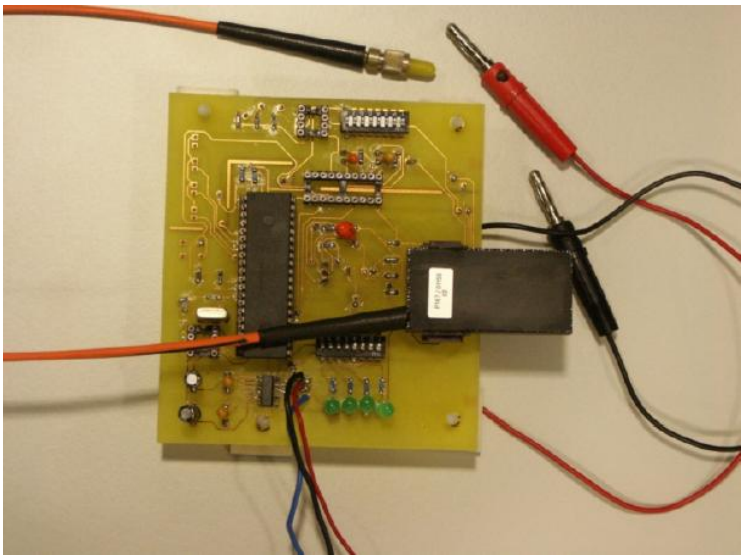

Figura 1.7: Imatge del primer prototip.

- **Febrer 2005 – Setembre 2005:**
  Títol: *'Diseny d'un espectroradiòmetre'* (TFC)
  Autor/s: Maribel Pérez i Núria Pujol

  Aquest projecte, seguint la línia del projecte anterior, tenia com a principal objectiu el de desenvolupar un prototip d'espectroradiòmetre introduint certes millores al projecte anterior.

  En la seva realització es va utilitzar un nou model d'espectròmetre, de control més senzill i de millor resposta freqüencial (S8378-256N SPL de Hamamatsu). El microcontrolador va ser substituït per un dsPIC i es va utilitzar un AD extern de 16 bits per obtenir major resolució en les dades obtingudes. El prototip desenvolupat es comunicava mitjançant el port sèrie amb un PC a través d'una nova interfície gràfica desenvolupada en Matlab, com en el cas anterior.

  Aquest prototip es va dotar de certa intel·ligència per adaptar-se en temps real a les característiques de les dades obtingudes (control automàtic del temps d'integració) i en el disseny de la placa es va tenir en compte l'espai disponible en l'interior d'una sonda oceanogràfica real.

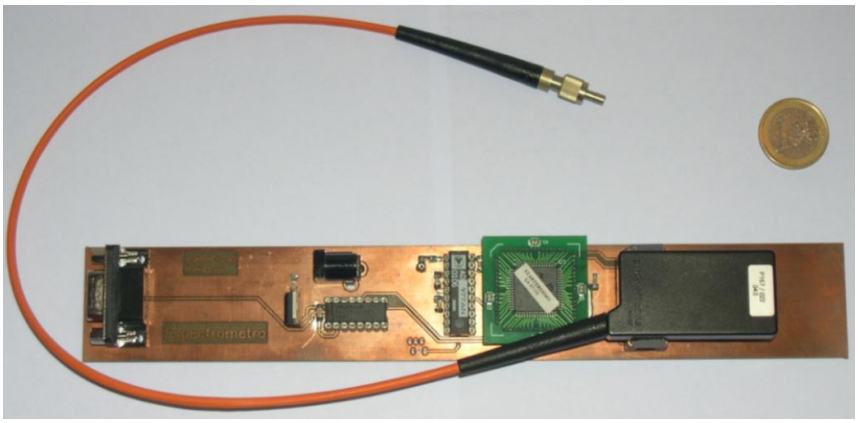

Figura 1.8: Imatge del segon prototip.

- **Setembre 2005 – Febrer 2006**
  Títol: *'Espectroradiòmetre intel·ligent basat en dsPIC'* (PFC)
  Autor/s: Sergi Pons

  En aquest projecte novament es va realitzar un prototip seguint les dimensions disponibles en una sonda real. El canvi més remarcable d'aquesta nova versió va ser el pas de Matlab a Python (multiplataforma) com a llenguatge de programació de la interfície de control i va marcar l'inici del desenvolupament de projectes amb filosofia *free software*.

  Dintre de les novetats respecte projectes anteriors trobem la introducció d'un sensor de pressió, que juntament amb les dades espectromètriques adquirides permetia caracteritzar la columna d'aigua, ja que la pressió serveix per determinar la profunditat a la qual han sigut preses les mostres.

  La intel·ligència desenvolupada pel projecte anterior (control automàtic del temps d'integració) també va ser introduïda en aquest nou prototip amb algunes millores.

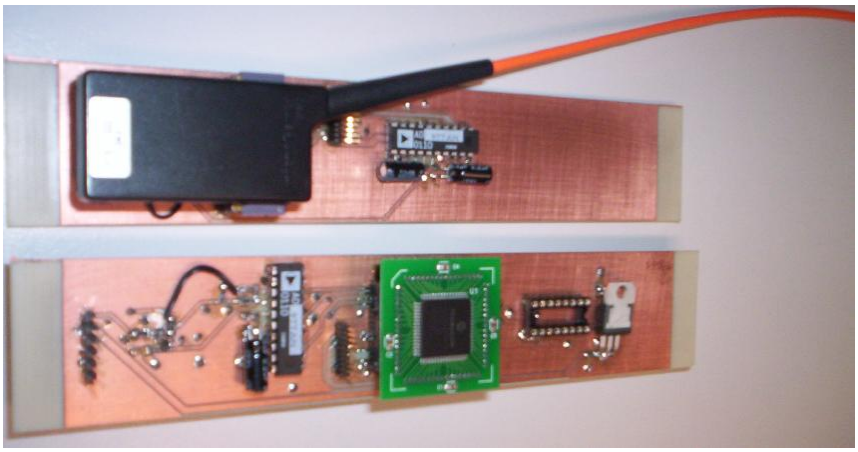

Figura 1.9: Imatge del tercer prototip.

Tot i que s'han realitzat més projectes dins del marc de col·laboració amb la UTM, aquests són els projectes que millor descriuen la línia seguida fins ara i els que han marcat els canvis més significatius.

El projecte actual marca un canvi en la línia de desenvolupament i recerca de la UTM. Aquest cop s'ha deixat de banda la idea de realitzar prototips basats en sensors espectromètrics senzills i s'han centrat tots els esforços en la realització de projectes a mida amb la utilització de dispositius més complexes.

Aquests dispositius són notablement més cars però permeten centrar-se en les necessitats específiques de la Unitat amb la possibilitat d'obtenir resultats tangibles a més curt termini. Com hem vist en el resum cronològic es necessita molt temps per arribar a fer un prototip que mai serà tant robust com els dispositius que es distribueixen de manera comercial.

A mode de resum, dir que els recursos limitats de la UTM han fet centrar-se en el desenvolupament software (en decriment del desenvolupament hardware) i al perfeccionament de noves tècniques de processat, el que suposa una menor inversió, aporta un valor afegit a les aplicacions i amb el que s'obtenen resultats tangibles més ràpidament.

CAPÍTOL 2. ENTORN DE DESENVOLUPAMENT

El capítol descriu l'entorn de desenvolupament del projecte i altres aspectes que s'han cregut necessaris per facilitar la posada en situació del lector. Els diferents apartats descriuen els elements implicats en el projecte (tant hardware com software), les seves característiques, com es pretenen utilitzar i amb quina finalitat.

2.1. Característiques dels dispositius utilitzats

La utilització dels dispositius hardware implicats en el projecte ha sigut un requeriment del projecte des del principi. Així doncs, no s'ha pogut influir en la seva elecció. Tot i així, a continuació es presenta un resum de les característiques dels dispositius utilitzats justificant-ne la seva idoneïtat.

2.1.1. L'espectròmetre miniaturitzat USB4000

L'espectròmetre miniaturitzat USB4000 d'Ocean Optics és l'evolució d'un model anterior, al que s'ha millorat el detector i part de la seva electrònica, per tal de poder obtenir una millor resposta temporal i una major resolució freqüencial del sistema.

Degut a que tota l'electrònica relacionada amb el tractament del senyal es troba encapsulada en el mateix dispositiu, la lectura de dades espectromètriques útils es pot realitzar directament a través del seu port USB o RS232.

Figura 2.1: Vista externa del dispositiu USB4000.

El dispositiu compta amb una entrada de fibra òptica on la llum incident es concentra i es condueix cap al seu interior. Un cop dins de l'espectròmetre, la llum incident es refractada per elements òptics que la separen en diferents longituds d'ona, que es reparteixen per la superfície del detector. En aquest cas, el detector està format per 3648 fotodíodes, on cada un es l'encarregat de captar un determinat rang de longitud d'ona i, en conjunt,

permet obtenir dades de les components de la llum incident dintre del rang de 200 a 1100 nm.

La llum incident captada durant un determinat temps, període anomenat temps d'integració, es convertida per cada fotodíode en una diferència de potencial proporcional a ella. Cada una de les diferències de potencial obtingudes és digitalitzada per un A/D intern de 16 bits i, posteriorment, les dades obtingudes són empaquetades en bytes per ser enviades a través del protocol sèrie o USB. Així doncs, entenem com a adquisició l'acció d'exposar els fotodíodes durant un temps d'integració (configurable) a la llum incident i la recollida de les dades obtingudes per poder ser estudiades i/o emmagatzemades.

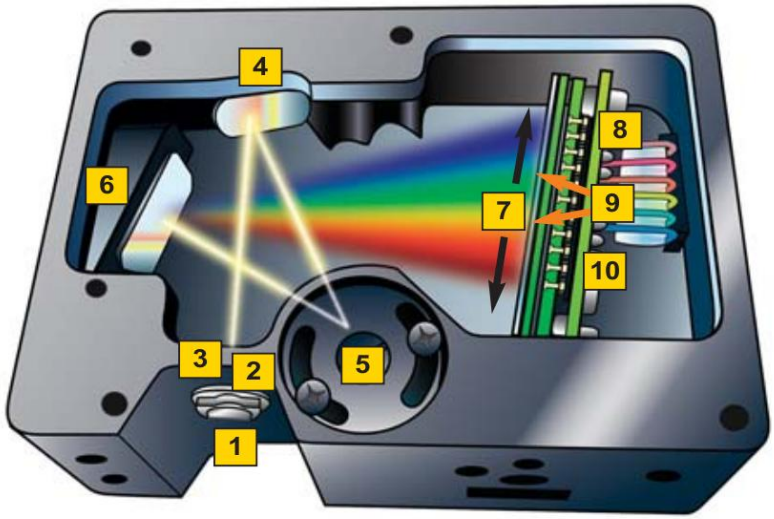

Figura 2.2: Vista interna del dispositiu USB4000.

El procés d'adquisició està internament controlat per tres senyals digitals: SH (*Shutter*), ICG (*Integration Clear Gate*) i M (*Master Clock*), a més a més de l'alimentació i el senyal de referència o massa. El senyal SH defineix la durada del temps d'integració, el senyal ICG indica el temps en que es reseteja el detector entre adquisicions i la senyal M s'utilitza com a temps de referència per realitzar la lectura dels diferents fotodíodes que formen l'array.

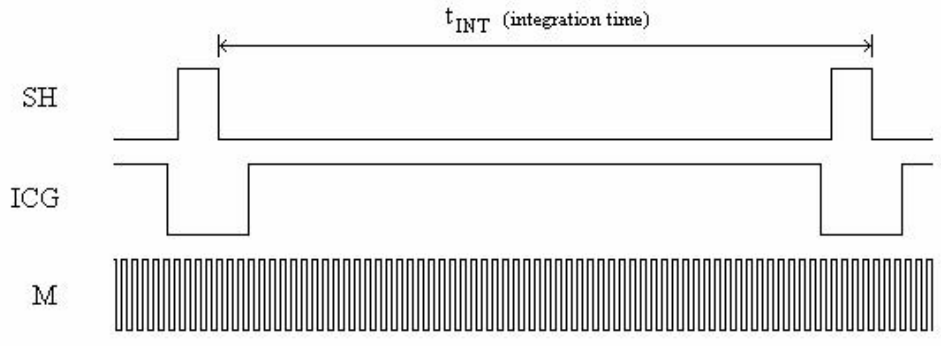

Figura 2.3: Senyals digitals de control del dispositiu USB4000.

Les principals característiques elèctriques i funcionals del dispositiu són:

- Consum: 250 mA. a 5 VDC.
- Dimensions: 89.1 mm x 63.3 mm x 34.4 mm.
- Pes: 190 gr.
- Detector: CCD TCD1304AP Toshiba d'alta sensibilitat.
- Rang de longituds d'ona detectables: de 200 a 1100nm.
- Temps d'integració configurable: de 10us. a 65s.
- Conversor A/D intern de 16 bits.
- Ports: USB 2.0 configurable a 480 Mbps (High Speed) o 12 Mbps (Full Speed) i RS232 (115 Kbaud).
- Coeficients de calibració i correcció configurables (guardats en memòria EEPROM).
- 24 pins per l'activació i connexió amb altres dispositius.

Les característiques que el diferencien de productes similars són el seu baix consum i les seves petites dimensions. Aquestes són característiques importants, ja que aquest dispositiu es vol introduir en una sonda oceanogràfica, on l'alimentació i les dimensions disponibles són autèntiques limitacions tècniques. El temps de resposta del dispositiu i la resolució proporcionada també són millors que en la majoria d'espectròmetres miniaturitzats del mercat.

2.1.2. El dispositiu PDA ReconX

El dispositiu PDA Recon X-Series de TDS està especialment dissenyat per treballar en condicions ambientals extremes. Les seves reduïdes dimensions (16.5 cm. x 9.5 cm. x 4.5 cm.) i el seu pes (490 gr.), la fan també una eina de control manejable. Aquesta sèrie de característiques la converteixen en un dispositiu de control indicat per aplicacions en alta mar.

Moltes de les condicions de treball extremes que el dispositiu és capaç de suportar es troben dins dels estàndards que ha de seguir l'equipament militar nord-americà (estàndard MIL-STD 810F), el que demostra la robustesa del dispositiu.

Entre les característiques relacionades amb aquestes condicions extremes de treball sota les quals és capaç treballar, trobem:

- Protecció vers immersions accidentals.
- Pantalla TFT tàctil dissenyada per evitar enlluernaments provocats per la llum solar.

- Totalment impermeable a la sorra i a la pols.
- Resistent a les vibracions.
- Resistent a condicions d'alta humitat relativa.
- Operativitat en baixa pressió i fluctuacions brusques.
- Rang de temperatura d'operació de -30^{o}C a 60^{o}C.

Figura 2.4: Aparença del dispositiu PDA ReconX de TDS.

A part de les condicions extremes que és capaç de suportar, també són importants les característiques elèctriques i informàtiques del dispositiu, les que realment influiran en el rendiment del software desenvolupat i les que poden suposar algun tipus de limitació.

Les característiques elèctriques i informàtiques més importants són:

- Processador: Intel PXA255 Xscale CPU @ 400 MHz.
- Memòria: 64 MB high-speed SDRAM.
- Emmagatzemament: Memòria Flash no volàtil 256 MB.
- Port d'expansió: Ranura Compact FlashType I i Type II.
- Pantalla: TFT a color de 240x320 píxels (tàctil).
- Bateries: recarregables 3800 mAh (NiMH).
- Ports: RS232 9-pin estàndard i USB (només client).

Com ja era d'imaginar en un dispositiu 'de butxaca', les limitacions principals venen donades per la memòria disponible i la velocitat de processador, el que farà necessària la optimització del codi desenvolupat i sobretot, la instal·lació en el dispositiu de les llibreries estrictament necessàries.

2.2. Estructura de comunicació entre dispositius

2.2.1. Configuració inicial i els seus inconvenients

Des d'un primer moment es va estar treballant en la idea de realitzar la comunicació entre els dos dispositius utilitzant la comunicació USB, ja que tant l'espectròmetre com la PDA disposen d'un port d'aquestes característiques.

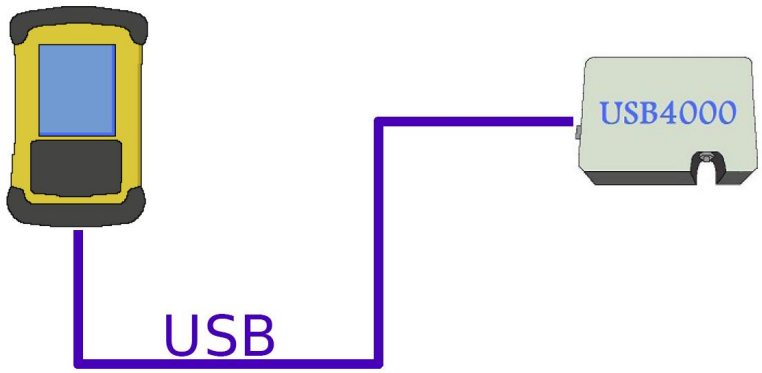

Figura 2.5: Configuració inicial de la comunicació.

Després de diversos intents fallits en la configuració de la PDA per la implementació de l'esmentada comunicació, es va arribar a la conclusió que tot i la PDA disposa d'aquest port no es possible utilitzar-lo tal i com es desitjava. El problema es deu a que el port USB del que disposa aquest model de PDA és únicament client, com posteriorment el fabricant va corroborar, encara que en la majoria de fulls d'especificacions no queda especificat. Els ports USB client només poden ser utilitzats per la comunicació, control i descàrrega de fitxers des d'un PC que exerceix de servidor.

Al descobrir que la idea que es volia dur a terme no es podia realitzar, es van estudiar les possibles solucions que permetien no renunciar a l'ús de la comunicació USB (comunicació més ràpida i segura), obtenint dues úniques solucions:

1. Utilització de la PDA Nomad de TDS:

   Aquest dispositiu compta amb un port USB Host i ja s'han desenvolupat algunes aplicacions al respecte, però aquesta opció queda directament descartada degut al seu elevat cost (entre 2.500 i 3.000$).

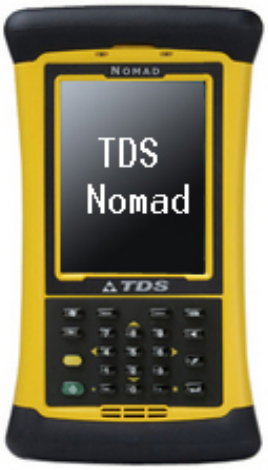

Figura 2.6: Dispositiu PDA Nomad de TDS.

2. Utilització d'un adaptador USB Host Compact Flash:

   Opció més econòmica que l'anterior i inclús el fabricant de la PDA (TDS) disposa dels drivers compatibles amb els adaptadors de la marca Ratoc. Malauradament va ser impossible adquirir la targeta compatible amb el nostre dispositiu (Ratoc USB 1.1 Host Adapter Compact Flash Card CFU1) per estar descatalogada. També es va descartar el mercat de segona mà per ser poc fiable.

Figura 2.7: Adaptador Compact Flash a USB Host (Ratoc).

Un cop esgotades les possibles solucions, es va optar per la modificació de la configuració del sistema i introduir un PC (que en un futur, fora dels objectius d'aquests projecte, es pretén substituït per un PC-104) utilitzant-lo com a pont entre els dos dispositius, que a més a més, permetrà alliberar de càrrega computacional al dispositiu PDA.

2.2.2. Configuració final

Degut als impediments tècnics sorgits en la configuració de la comunicació USB en el dispositiu PDA, ha sigut necessari redissenyar l'escenari de control del dispositiu USB4000.

Aquesta nova configuració segueix la mateixa filosofia que l'anterior però, en aquest cas, s'ha introduït un ordinador, que en un futur pretén ser substituït per un PC-104, a mode de pont en la comunicació entre l'espectròmetre i el dispositiu PDA. Aprofitant aquest canvi de configuració, s'ha redissenyat el software de control per tal d'alliberar de càrrega de treball el dispositiu PDA i que la seva funció passi a ser merament de configuració i control.

Aquesta nova configuració suposa un model més adequat per l'ús final de l'aplicació, ja que permet inserir de forma més senzilla altres sensors (pressió, temperatura,etc.), podent ser igualment controlats per la PDA mentre les dades són tractades i emmagatzemades en un 'ordinador central', encarregat de suportar la major par de la càrrega computacional del sistema.

Figura 2.8: Configuració final del sistema.

En aquesta nova configuració del sistema, i definitiva, la comunicació entre PDA i PC es realitza a través de la comunicació RS-232 i la comunicació entre el PC i el dispositiu USB4000 es realitza a través del port USB, utilitzant el protocol dissenyat pel fabricant de l'espectròmetre per aquest tipus de comunicació.

2.3. Llenguatge de programació i eines software

Un dels requisits a complir pel software desenvolupat, com ja s'ha comentat anteriorment, és el de desenvolupar una eina *free software* oberta a tota la comunitat científica utilitzat únicament programari lliure en el seu desenvolupament.

En aquest apartat s'introdueixen alguns conceptes relacionats amb el software lliure i es descriu el programari i les eines lliures utilitzades en el desenvolupament del projecte.

2.3.1. El software lliure

Anomenen software lliure (*free software* en anglès) el software que pot ser utilitzat, estudiat, modificat sense restriccions, copiat i distribuït a tercers que tenen aquestes mateixes llibertats sobre ell. Per tant, podem dir que un software pot ser considerat *free software* si compleix una sèrie de llibertats:

- La llibertat d'executar el programa per a qualsevol propòsit i ús.
- La llibertat d'estudiar-ne el funcionament i adaptar-lo a les pròpies necessitats.
- La llibertat de distribuir-ne còpies a tercers.
- La llibertat per millorar el programa i fer conèixer aquestes millores al públic en benefici de la comunitat d'usuaris.

Aquestes llibertats impliquen poder tenir accés al codi font del software (codi obert o *open source*), així com als manuals i a tota la informació relacionada, ja que s'entén que també formen part del software.

Alguns dels exemples més rellevants d'eines free software i codi obert en diferents àmbits són els següents:

- Sistemes operatius: Linux (o GNU/Linux), FreeBSD, OpenBSD, NetBSD i GNU/Hurd.
- Llenguatges de programació: GNU C/C++, Perl, Python i Tcl.
- Navegadors Web: Mozilla.
- Aplicacions ofimàtiques: Open Office.
- Aplicacions web i servidors: Apache, Php i MySQL.

Com en qualsevol software propietari, el software lliure també disposa de llicències que plasmen els drets i deures d'usuaris i autors, juntament amb les condicions d'ús del propi software. Inclús la documentació generada i que acompanya al software també pot tenir la seva pròpia llicència. En la majoria dels casos, perquè la llicència pugui ser aplicada s'ha d'incloure el text legal de la llicència amb el programa i codi font. A més a més, s'ha d'adjuntar informació de la llicència al arxius font i a la documentació en forma de capçalera.

Existeixen gran nombre de llicències per la distribució de software lliure que podem classificar en dos grans grups:

- **Llicències permissives:** són llicències que no imposen condicions especials per a la distribució i modificació del software.
- **Llicències robustes (o *copyleft*):** són llicències que imposen condicions a l'hora de redistribuir el software, obligant que les successives redistribucions respectin les condicions de la llicència general.

Una de les llicències per a la distribució de software lliure més utilitzada és la GNU General Public License (GPL), sota la qual es pensa distribuir el software desenvolupat.

La Llicència Pública General, que fou creada per mantenir la llibertat del software i evitar-ne l'apropiació, dóna a l'usuari la llibertat de compartir i/o modificar el software distribuït

sota aquesta llicència i obliga als software derivats d'aquest a distribuir-se amb la mateixa llicència i a mantenir els noms i crèdits dels autors originals.

En conclusió, qualsevol software llicenciat sota una llicència GPL i les seves modificacions ha d'estar disponibles al conjunt de la humanitat.

2.3.2. Característiques del llenguatge Python

El llenguatge Python és un llenguatge de programació cada cop més utilitzat, ja que es tracta d'un llenguatge de codi obert i de ràpid aprenentatge. Aquestes característiques afavoreixen que compti amb una gran comunitat de desenvolupadors i que gran nombre de fonts en línia hi estiguin dedicades (manuals, fòrums de discussió, cursos *on-line*, etc.).

D'una manera més tècnica podem dir que Python és un llenguatge interpretat, orientat a objecte, d'alt nivell i el més important, multiplataforma. Es tracta d'un llenguatge simple i la seva corba d'aprenentatge es prou ràpida, el que es tradueix en una major productivitat des del principi de la seva utilització.

Entre els projectes més important i coneguts desenvolupats en Python trobem: YouTube, algunes aplicacions de Google i el buscador Yahoo!.

2.3.3. Eines software de desenvolupament

En el desenvolupament del software s'han utilitzat diverses eines de software lliure, tant per el propi desenvolupament de l'aplicació com per la configuració dels dispositius.

Abans de començar amb el desenvolupament software pròpiament dit, ha sigut necessari la instal·lació d'un sistema operatiu *free software* tant a l'ordinador de sobretaula com a la PDA, ja que encara avui en dia tots aquests dispositius s'adquireixen amb el sistema operatiu Windows instal·lat per defecte. L'ordinador de sobretaula, on s'ha realitzat la major part del desenvolupament software, compta amb la distribució Ubuntu de Linux (versió 7.10) i en el cas de la PDA, el sistema operatiu del que es disposa és Qtopia versió 2.1.2.

Tot i que una part del software de control dissenyat ha d'executar-se sobre un dispositiu PDA, la totalitat del software ha sigut desenvolupat sobre un ordinador de sobretaula i en algunes proves s'ha emulat la comunicació entre la PDA i l'ordinador central utilitzant dos ordinadors units pel port sèrie, per comoditat. Degut a que el llenguatge Python és multiplataforma per definició, el software desenvolupat en un ordinador es fàcilment adaptable a un dispositiu PDA d'una forma quasi directa.

Pel desenvolupament software tan sols ha sigut necessària la instal·lació, en els ordinadors utilitzats, d'un editor de llenguatge Python i les llibreríes específiques de les funcions cridades pel propi codi. Les llibreries necessàries pel desenvolupament i funcionament s'enumeren a continuació:

- **Editor Python:** DrPython
- **Llibreries Python:**
  - python2.5: llibreries python bàsiques.
  - python-netcdf: llibreries NetCDF per Python.
  - python-numeric: llibreries matemàtiques orientades a treballar amb matrius.
  - python-numpy:llibreries matemàtiques per facilitar les tasques realitzades amb vectors de dades.
  - python-qt4: llibreries pel desenvolupament d'entorn gràfics Qt.
  - python-scientific: llibreries per realitzar càlculs de dades científiques.
  - python-serial: llibreries d'accés al port sèrie.
  - python-pyusb: llibreries d'accés al port USB.

Juntament amb les llibreries enumerades anteriorment, cal afegir les dependències de les mateixes, tot i que en el cas de la instal·lació en un ordinador s'ha utilitzat un gestor de paquets que ja les detecta i les instal·la.

Existeixen diferents editors Python però com que l'aprenentatge d'aquest llenguatge s'ha fet en paral·lel a la realització del projecte, s'ha optat per l'opció més senzilla, l'editor DrPython. Aquest editor s'ha utilitzat també en el disseny de l'entorn gràfic de la PDA, tot i existir eines específiques per aquest fi.

El sistema operatiu Qtopia versió 2.1.2 ha sigut el sistema operatiu instal·lat en la PDA, ja que és la opció *free software* oferida pel propi fabricant del dispositiu. Qtopia és un sistema operatiu per a dispositius mòbils basat en Linux i que actualment gran varietat d'aparells utilitzen: telèfons mòbils, reproductors portàtils multimèdia (PMP), dispositius PDA, etc.

Qtopia és un sistema operatiu *open source* en la seva totalitat i amb una interfície gràfica íntegrament implementada utilitzant llibreries Qt, el que força que les aplicacions desenvolupades sobre aquest tipus de plataforma les hagin també d'utilitzar, ja que altres llibreries gràfiques no són compatibles. Existeixen llibreries Qt sota diferents llenguatges de programació, entre els més importants trobem: C++, Python i Perl.

Les llibreries Qt són un conjunt de llibreries multiplataforma pel desenvolupament d'interfícies gràfiques, que al ser distribuïdes sota una llicència GPL poden ser utilitzades per a la creació d'aplicacions *open source*. És un llenguatge orientat a objecte i el seu principi de funcionament està basat en objectes (elements gràfics), senyals (comunicació entre elements gràfics i funcions) i events (interacció de l'usuari sobre els elements gràfics). Algunes de les aplicacions més populars desenvolupades utilitzant aquestes llibreries gràfiques són: Skype, Google Earth i Adobe Photoshop Elements.

Per tal de poder executar l'aplicació desenvolupada en el dispositiu PDA, a part de contar amb un sistema operatiu, és necessària la instal·lació d'una sèrie de llibreries Python, que no formen part de la versió Qtopia per defecte. La instal·lació de llibreries addicionals en

aquest tipus de dispositiu és una tasca una mica més complicada que en el cas dels ordinadors convencionals, ja que els dispositius PDA no compten amb la mateixa disponibilitat de memòria i els desenvolupadors intenten fer-ne un millor ús.

Les llibreries Python per dispositius mòbils, i també d'altres llenguatges o aplicacions, es troben moltes vegades en format IPKG i són versions reduïdes o fragmentades de versions de les llibreries Python convencionals. La complicació de la instal·lació d'aquests paquets rau en la multitud de paquets a instal·lar i que moltes vegades els mòduls no incorporen la informació necessària (de dependències, per exemple) per a la seva correcta instal·lació.

L'eina recomanada per la instal·lació de les diferents llibreries addicionals per al sistema operatiu Qtopia (distribució Familiar Linux) és l'aplicació Qtopia Desktop. Aquesta aplicació sincronitza la còpia i la instal·lació d'arxius des d'un PC a una PDA a través del port USB. Per tal de realitzar aquesta sincronització hem de configurar les adreces de la connexió USB dels dos dispositius per tal que pertanyin a la mateixa xarxa i puguin comunicar-se. La instal·lació d'aquests arxius a la PDA és susceptible de fallides i en aquests casos és necessari recórrer al protocol SSH de transferència de fitxers i instal·lar els paquets manual i remotament a través del Terminal.

El protocol SSH és un protocol que permet accedir a màquines remotes a través de la xarxa i que a més a més permet copiar dades de forma segura entre el servidor i el client. En el nostre cas aquest protocol ens servirà per accedir a la PDA a través del PC utilitzant el port USB i poder així instal·lar mòduls, modificar arxius,etc.

A continuació, i per finalitzar l'apartat s'enumeren les llibreries Python instal·lades en el dispositiu PDA, juntament amb les llibreries necessàries per la seva instal·lació:

- Llibreries Python:
  - python-disutils: llibreries d'utilitats de la distribució Python.
  - python-math: llibreries d'utilitats matemàtiques.
  - python-numarray: llibreries de processat numèric.
  - python-pyqt3: llibreries d'entorn gràfic.
  - python-pyserial: llibreries per la configuració i control de la comunicació sèrie.
  - python-threading: llibreries pel desenvolupament basat en *threads*.
  - python-sip: requeriment de python-pyqt3
  - python-core: llibreries principals de Python per a dispositius mòbils.
- Requeriments per a la instal·lació de les llibreries Python:
  - libpython: llibreries utilitzades pel llenguatge pytohn.
  - gcc: compilador de llenguatge C.
  - gcc-libs: llibreries complementàries del compilador gcc.
  - binutils: llibreria d'utilitats binaries complementàries del compilador gcc.

2.4. Estàndard d'emmagatzemament de dades

A part de controlar l'adquisició de dades provinents del dispositiu USB4000, en molts casos és interessant que el sistema desenvolupat pugui emmagatzemar les dades obtingudes per tal de poder ser processades posteriorment. Aquest emmagatzemament es pot dur a terme de diverses maneres però la més senzilla i habitual és la de guardar les dades en un fitxer. Com es lògic, per tal de facilitar el posterior tractament de dades per altres programes, és recomanable que s'utilitzi algun format estandarditzat. En el nostre cas, s'ha optat pel format NetCDF, format utilitzat en l'emmagatzemament de dades científiques.

2.4.1. El format NetCDF

El format NetCDF (Network Data Form) és un estàndard obert per l'emmagatzemament i intercanvi de dades científiques en format binari. La principal característica representativa dels fitxers que segueixen aquest estàndard és que són fitxers autodescriptius, el que significa que les seves pròpies capçaleres descriuen les característiques de les dades contingudes (dimensions, magnituds, unitats , etc.). Aquesta característica fa possible que els programes que hagin de llegir i processar les dades contingudes puguin ser molt més flexibles, ja que poden saber l'estructura de dades del fitxer sense necessitat de recorre'l per complet.

Les dades en aquest tipus de fitxers s'emmagatzemen en forma d'arrays i/o matrius d'una, dos o més dimensions, el que en facilita la seva representació gràfica i per la qual cosa aquests tipus de fitxers estan especialment dissenyats. Aquesta estructura de dades queda definida per les següents tipus de paràmetres segons les seves funcionalitats i característiques:

- **Dimensions:** tipus de paràmetres definits per un nom i un nombre enter positiu que defineix la longitud de les dades emmagatzemades, tot i que les dades emmagatzemades poden tenir més d'una dimensió. D'una manera senzilla podem dir que el paràmetre o paràmetres definits com a dimensió corresponen als eixos en una representació gràfica de les dades emmagatzemades. Les dimensions poden ser de qualsevol tipus sempre que serveixin per ordenar les dades (temps, longitud, latitud, latitud, etc.).

- **Variable:** és la unitat bàsica d'emmagatzemament d'informació en un fitxer NetCDF. Les variables estan formades per un nom, el tipus de dades i la seva forma ve definida per una llista de dimensions declarades prèviament. Es la manera de crear l'estructura 'buida' en que les dades seran guardades.

- **Dades:** són realment la informació 'útil' que desitgem emmagatzemar i que corresponen als valors obtinguts en algun tipus d'adquisició. Aquesta informació es guardada seguint l'estructura d'una variable prèviament definida.

- **Atributs globals**: són els atributs que afegeixen informació general al fitxer. Tot i que no hi ha cap atribut global obligatori existeixen convencions sobre els atri-

buts mínims que tot fitxer NetCDF hauria de contenir: *title* (títol descriptiu), *institution* (procedència), *source* (mètode d'obtenció), *history* (historial sobre els estudis o modificacions que s'hagin pogut fer a les dades), *references* (estudis i publicacions relacionades) i *comment* (comentaris tècnics sobre les dades i/o la seva adquisició).

Aquest tipus de format es molt comú en l'emmagatzemament de dades meteorològiques o en dades en que poden descriure's en funció del la longitud, la latitud i l'altitud en que foren adquirides. En el cas particular de treballar amb dades espectromètriques, de les quals actualment no podem disposar de les dades de posició d'una manera automatitzada, provoca que aquestes dades es guardin en format NetCDF però amb altres magnituds de referència.

En la creació i configuració dels arxius NetCDF d'emmagatzemament de les dades espectromètriques adquirides s'ha fet ús de la llibreria python-netCDF, llibreria Python específica per treballar amb aquest tipus de format.

CAPÍTOL 3. SOFTWARE DE CONTROL DEL DISPOSITIU USB4000

3.1. Descripció general de l'aplicació

Com ja s'ha anat repetint al llarg de tot el document, l'aplicació dissenyada és una eina de control del dispositiu USB4000 per l'adquisició de dades espectromètriques. L'escenari de l'aplicació el formen tres dispositius: una PDA (ReconX), un ordinador convencional i un espectròmetre USB400. L'existència d'un ordinador com a pont de comunicació entre el dispositiu PDA i l'espectròmetre obliga a dividir el software de control en dos blocs (una part del codi de control s'executarà en la PDA i l'altra en l'ordinador) i que aquests dos blocs, a més a més, s'hagin de comunicar i coordinar entre ells.

El software de control implementat està format per diverses llibreries que governen els diferents dispositius i controlen la comunicació entre ells. El dispositiu PDA està governat per la llibreria *S4000AppPDA.py*, mentre que l'ordinador central està governat per la llibreria *S4000AppPC.py*, que a la vegada utilitza llibreries específiques desenvolupades per l'aplicació. Aquestes llibreries específiques implementen control del dispositiu USB4000 (*OOS4000v2.py*), l'emmagatzemament de dades en format NetCDF (*SaveNetCDF.py*) i el control d'errors (*CRC.py*). A més a més, també s'han utilitzat alguns mòduls de llibreries Python estàndards per implementar la comunicació sèrie i USB, entre d'altres funcions.

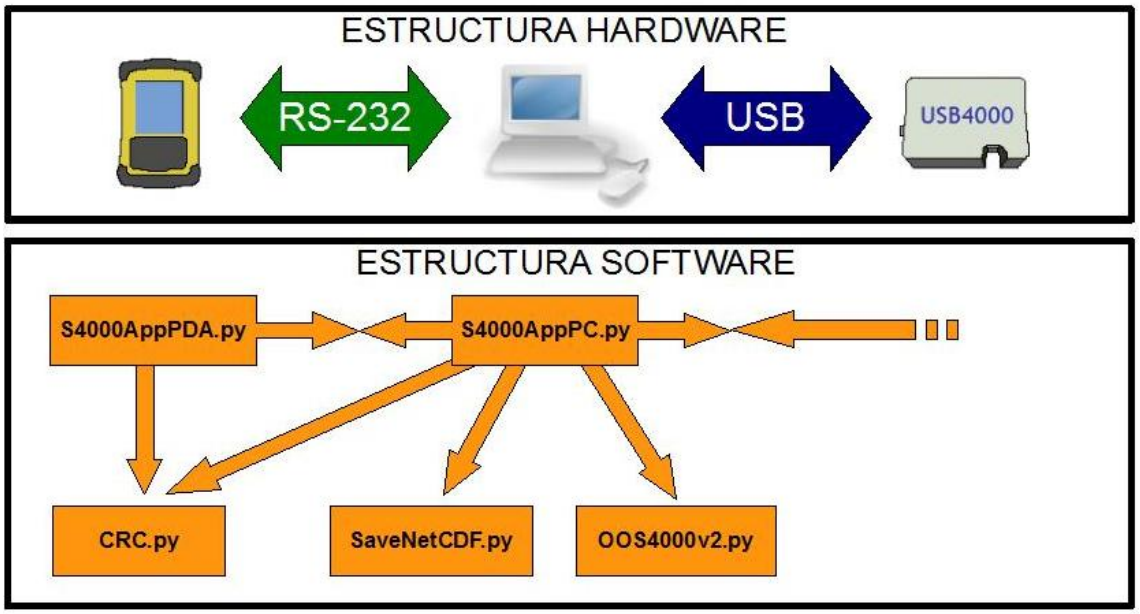

Figura 3.1: Esquema software del sistema en relació a l'esquema hardware.

Per tal de coordinar tot el procés d'adquisició de dades, el sistema de fitxers desenvolupat realitza un sèrie d'accions que podem agrupar en:

- **Configuració:** introducció per part de l'usuari de paràmetres referents a l'adquisició (número d'adquisicions a realitzar, espaiat temporal entre mostres, temps d'integració) i posterior emmagatzemament d'aquestes per poder ser utilitzades en l'execució i les peticions al dispositiu USB4000. També és important algun tipus de control sobre la validesa de les dades introduïdes.
- **Adquisició:** pas de paràmetres de configuració (en forma i format adequat) del dispositiu PDA a l'ordinador central i al dispositiu USB4000, i recollida de dades espectromètriques provinents del dispositiu USB4000 pel seu emmagatzemament en memòria temporal.
- **Representació:** visualització gràfica de les dades obtingudes en temps real per poder avaluar la seva coherència i detectar visualment el correcte funcionament del sistema i els punts de més interès.
- **Emmagatzemament:** arxivament en de les dades adquirides en format NetCDF seguint els paràmetres de configuració.

L'aplicació s'ha dissenyat per poder realitzar processos d'adquisició simples (una sola adquisició), continus finits (nombre configurat d'adquisicions) o continus indefinits (realitza adquisicions fins que l'usuari decideix aturar el procés), així doncs, les diferents tasques no sempre es van alternant en l'ordre anteriorment descrit i per cada adquisició no es realitzen necessàriament totes les tasques.

De les tasques anteriorment comentades el dispositiu PDA realitza les de càrrega computacional més senzilles (configuració i emmagatzemament) mentre que l'ordinador central és el que realment controla el funcionament del sistema.

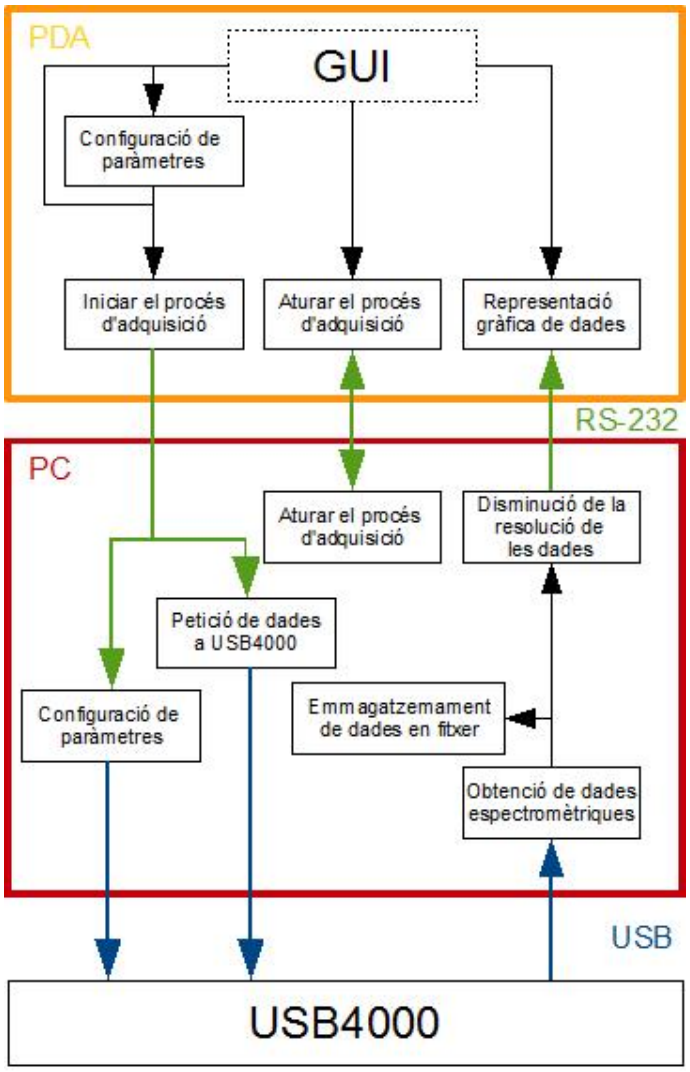

Figura 3.2: Esquema general de les tasques de l'aplicació.

3.2. Característiques i funcionalitats

Aquest apartat descriu en profunditat les funcionalitats del codi desenvolupat fins arribar al punt de comentar algunes funcions i adjuntar algunes línies de codi a mode d'exemple quan s'ha cregut necessari.

En comptes de descriure cada fitxer línia a línia s'ha cregut més útil descriure el funcionament global l'aplicació a partir de la descripció de les característiques i funcionalitats del sistema per comprendre millor la interacció entre dispositius.

3.2.1. Interfície gràfica (GUI)

Una de les funcions de la interfície gràfica és la de facilitar a l'usuari la configuració de paràmetres imprescindibles per ajustar el procés d'adquisició a les seves necessitats. La

introducció d'aquests paràmetres ha de ser senzilla i el més intuïtiva possible, per la qual cosa s'ha dissenyat una interfície gràfica simple tenint en compte els següents principis:

- Integració de la finestra principal a les funcionalitats del sistema operatiu de la PDA, com per exemple el teclat virtual.
- Aprofitament al màxim de les possibilitats tàctils del dispositiu.
- Colors, text i gràfics adaptats a la lluminositat ambiental.
- Estructura lògica i intuïtiva.
- Anglès com a idioma vinculant amb l'aplicació.

La interfície gràfica de l'aplicació està estructurada en un petit espai gràfic per a la representació de les dades espectromètriques obtingudes i dues pestanyes que contenen caselles de configuració de paràmetres i botons d'accionament. Una de les pestanyes conté tots els elements que controlen l'adquisició pròpiament dita (botons d'accionament i paràmetres de configuració de l'adquisició) i l'altra conté els paràmetres relacionats amb el l'emmagatzemament de dades en format NetCDF (paràmetres de configuració d'arxius). Aquesta estructura és fàcilment ampliable a noves funcionalitats simplement ampliant el nombre de pestanyes.

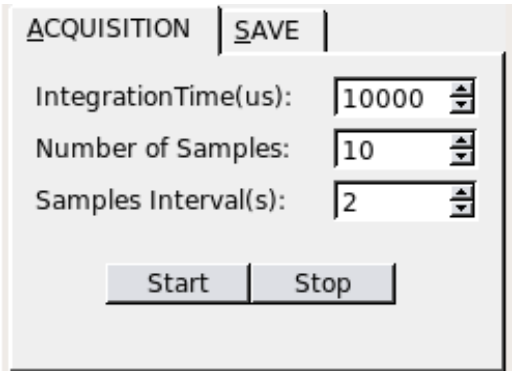

Figura 3.3: Pestanya de configuració dels paràmetres d'adquisició.

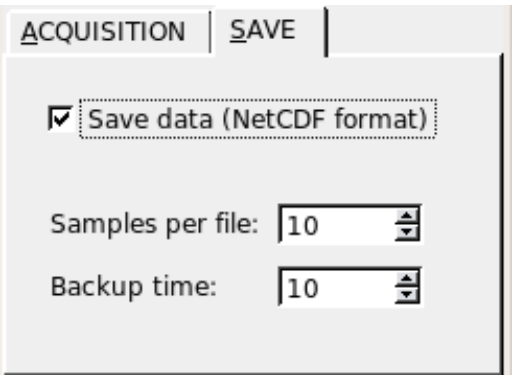

Figura 3.4: Pestanya de configuració dels paràmetres d'emmagatzemament.

Per poder assegurar el correcte funcionament de l'aplicació s'ha hagut d'evitar a través del software la introducció de valors no vàlids per part de l'usuari i també s'han inhabilitat alguns elements durant el procés d'adquisició per evitar errors que poguessin provocar un mal funcionament del sistema.

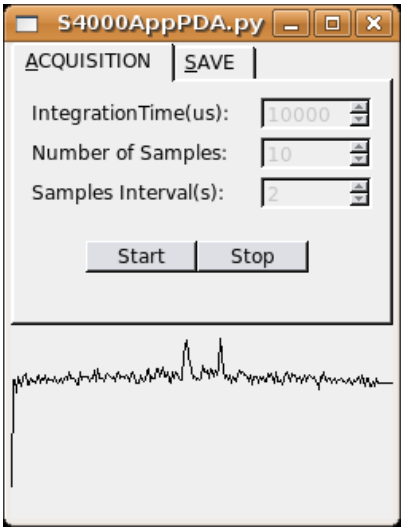

Figura 3.5: Exemple d'inabilitament d'elements de la GUI durant l'adquisició.

3.2.2. Configuració de paràmetres

El primer pas per realitzar la lectura de dades espectromètriques és la introducció dels paràmetres de configuració necessaris, a no ser que es vulguin utilitzar els paràmetres que apareixen per defecte en la interfície gràfica. Els paràmetres configurables de cada pestanya i el seu significat queden descrits a continuació.

Paràmetres configurables de la pestanya d'adquisició (*'Adquisition'* Tab):

- **Temps d'integració** (*'Integration Time'*)**:** aquest paràmetre correspon al temps en microsegons que els fotodíodes estan exposat a la llum incident durant l'adquisició. Aquest valor és directament proporcional a l'amplitud de les dades espectromètriques obtingudes i per tant, si cada cop es realitzen mesures a més profunditat s'ha d'anar augmentant aquest valor si es vol aprofitar tot el marge dinàmic del conversor A/D.

- **Nombre de mostres** (*'Number of Samples'*)**:** aquest paràmetre correspon al nombre de mostres que es volen realitzar consecutivament. La introducció del valor '0' en aquest camp configura l'adquisició com a continua indefinida i per tant, en aquest cas el procés d'adquisició (en el que es realitzen diverses adquisicions) no s'atura automàticament sinó quan l'usuari decideix aturat aquest procés a través del botó 'Stop' de l'aplicació.

- **Espaiat temporal entre mostres** (*'Samples interval'*)**:** aquest paràmetre configura el temps d'espera entre l'adquisició d'una mostra i la de la següent en segons.

Aquest paràmetre, juntament amb el nombre de mostres, determina la durada del procés d'adquisició. Aquest temps serà, aproximadament, el producte entre el nombre de mostres i l'espaiat temporal.

El valor de l'espaiat temporal no pot ser menor que el temps d'integració (limitat pel propi software), ja que tot i que el programa principal està desenvolupat a partir de threads, el dispositiu USB4000 no pot realitzar diverses adquisicions en paral·lel. Fins que no ha acabat d'una mostra, de durada aproximadament igual al temps d'integració configurat, no pot començar el procés d'adquisició de la següent.

El valor idoni de l'espaiat temporal ve determinat, lògicament, per l'entorn de treball i l'objectiu de l'adquisició. En el cas de voler caracteritzar una columna d'aigua es desitjable que l'espaiat temporal sigui un valor petit, ja que l'objectiu de l'adquisició és determinar la composició de la columna i determinar els diferents estrats. Quan l'objectiu és caracteritzar un punt més o menys fixe, aquest espaiat pot ser un valor major, ja que l'interès de l'adquisició es centra en la detecció de canvis significatius. En cap dels dos casos és necessari un espaiat temporal de menys d'un segon, el mínim configurable a través de la interfície gràfica.

Paràmetres configurables de la pestanya d'emmagatzemament de fitxers (*'Save File'* Tab):

- **Mostres contingudes en cada fitxer** (*'Samples per File'*)**:** aquest paràmetre determina el nombre màxim de mostres que ha de contenir cada fitxer de dades i per tant, determina la mida màxima dels fitxers.

  En molts casos els aparells d'adquisició de dades oceanogràfiques estan molts mesos adquirint mostres en alta mar, això suposa un gran de volum de dades a emmagatzemar. Per tant, és aconsellable repartir aquestes dades en fitxers de mida raonable per poder ser tractats fàcilment i evitar que la corrupció d'un fitxer suposi la pèrdua de les dades de diversos mesos.

  El nombre de mostres per fitxer s'ha limitat a 30.000 tot i que per aquest nombre de mostres s'obtenen arxius exageradament grans i el més normal serà configurar aquest paràmetre a unes 200 mostres.

- **Nombre de mostres de 'backup'** (*'Backup Time'*)**:** aquest paràmetre determina cada quantes mostres s'actualitza l'arxiu de dades. Encara que les mostres adquirides es guardin en diferents arxius, cada arxiu seguirà contenint moltes mostres i per això s'ha cregut necessari introduir aquest segon paràmetre de configuració. Amb la configuració d'aquest valor s'aconsegueix que les dades es vagin copiant a l'arxiu en blocs petits de mostres, fent que davant una fallada del sistema, per exemple, es perdin el menor nombre de mostres possible.

Malauradament cap paràmetre resguarda al dispositiu de fallades en el funcionament i que des d'un determinat moment es deixin d'obtenir mostres.

3.2.3. Comunicació entre PDA i PC

La comunicació entre PDA i PC es realitza a través del port sèrie, això implica que tota la informació a intercanviar entre els dispositius ha de tenir format de byte o millor dit, de caràcter. Com ja hem comentat anteriorment, tots els paràmetres de configuració són valors que per defecte tenen format enter (Python considera format enter nombres codificables fins amb 32 bits) i que per tant, no poden ser enviats directament pel port sèrie.

Python disposa de la llibreria *struct* que conté funcionalitats que permeten empaquetar estructures de dades de qualsevol format a format string i posteriorment recuperar-les en el format original. Així doncs, a partir d'aquesta llibreria s'ha construït un simple 'protocol' de comunicació entre aquests dos dispositius.

En el disseny del protocol de comunicació s'ha tingut en compte que existeixen tasques diferenciades que impliquen la comunicació entre aquests dispositius en un sentit o un altre i per tant, per cada tipus de tasca s'ha dissenyat un tipus de paquet. Els paquets segueixen una mateixa estructura però es diferencien per la seva capçalera i longitud. L'estructura del paquet està formada per un caràcter indicador de principi de paquet, dos caràcters que en determinen el tipus i la informació útil del paquet.

A continuació es descriuen els tipus de paquets utilitzats:

- **Paquet de configuració:** transporta paràmetre de configuració des del dispositiu PDA fins a l'ordinador central, ja que és aquest dispositiu el que realment controla el procés d'adquisició.
- **Paquet de dades:** transporta les dades espectromètriques provinents del dispositiu USB4000 des del PC central fins a la PDA per poder ser representades gràficament.
- **Paquet ACK:** confirma a l'ordinador 'central' l'arribada de paquets de dades espectromètriques a través de retornar l'identificador amb el que l'ordinador central ha enviat el paquet. Aquest tipus de paquets també s'utilitzen per comprovar i controlar el correcte funcionament de la comunicació. Per les característiques actuals de la comunicació no serien estrictament necessaris aquest tipus de comprovacions ja que la comunicació es realitza per cable i sense possibles problemes de col·lisions però està pensat per la realització de futures proves utilitzant una comunicació inalàmbrica entre PDA i PC.
- **Paquet de finalització:** indica la finalització del procés d'adquisició als dos dispositius. Aquest paquet es enviat del PC a la PDA quan s'ha arribat el nombre de mostres configurades o de la PDA al PC (i posteriorment retornat) quan l'usuari decideix aturar el procés d'adquisició.

Fins el moment s'ha descrit el protocol de comunicació però no s'ha entrat en detalls de com s'implementa la comunicació sèrie amb el llenguatge Python a través de la utilització de la seva llibreria *pyserial*. El primer pas necessari per la utilització del port sèrie és la seva configuració. Aquesta configuració es realitza a través de la inicialització d'un objecte Serial amb la configuració dels seus paràmetres:

```
ser=Serial('/dev/ttyS0',57600)
```

El primer paràmetre de la crida correspon a una referència del port sèrie al qual volem accedir, la sintaxis utilitzada en aquest cas és la mateixa que utilitza qualsevol sistema operatiu i el segon paràmetre simplement configura la velocitat de transmissió. Cal remarcar que el valor de velocitat de transmissió ha de ser un valor dels estipulats per aquest tipus de transmissió (no pot ser qualsevol valor) i s'ha d'haver configurat amb el mateix valor els dos dispositius que es comuniquen.

Així es pot donar per configurada la comunicació, obtenint una variable a la que podem accedir per llegir/escriure a través del port sèrie. En el nostre cas, la configuració del port sèrie es realitza a l'iniciar el programa, a la vegada que es declaren i s'inicialitzen algunes variables.

A continuació i a mode d'exemple s'adjunten les línies de codi necessàries per empaquetar les dades corresponents a un paquet de configuració i el seu enviament a través del port sèrie:

```
tosend=pack('c2sHiHBB','*','AD',nsamples,itime,interval,SamplesxFile,ToSave)
ser.write(tosend)
```

El primer paràmetre del mètode *pack* correspon al format de les dades empaquetades. Per exemple, 'c' correspon a un caràcter, '2s' a una string de 2 caràcters, 'i' a un enter de 32 bits o 'B' a un enter de 4 bits. Sabent aquesta cadena descriptiva del format de les dades a empaquetar podem recuperar les dades en el format original a través del mètode *unpack*.

3.2.4. Comunicació entre PC i USB4000

En la implementació de la comunicació entre PC i USB4000 s'ha utilitzat el mòdul pyUSB de Python, específic per implementar la comunicació USB entre dispositius. El primer pas per poder comunicar dos dispositius a través de la comunicació USB és l'establiment de la connexió.

Tot dispositiu USB disposa d'un gran nombre d'atributs relacionats amb la seva descripció, configuració, comunicació, etc. Dintre dels atributs anomenats descriptius trobem l'atribut *idVendor* (identificador únic de fabricant), que en el nostre cas utilitzarem per identificar el dispositiu USB4000 connectat al PC i poder així obrir la comunicació amb ell. L'identificador dels dispositius USB4000 conté el valor hexadecimal 0x2457, tal i com indica el seu *datasheet*.

Com a curiositat, comentar que a través de la comanda lsusb -v de Linux obtenim un llistat de tots els atributs dels diferents dispositius USB connectats al nostre PC, així com els dels ports USB disponibles.

Per poder accedir a un determinat dispositiu connectat a qualsevol dels ports USB d'un PC i establir una connexió és necessari tenir en compte l'estructura de les classes y funcions definides segons el mòdul pyUSB.

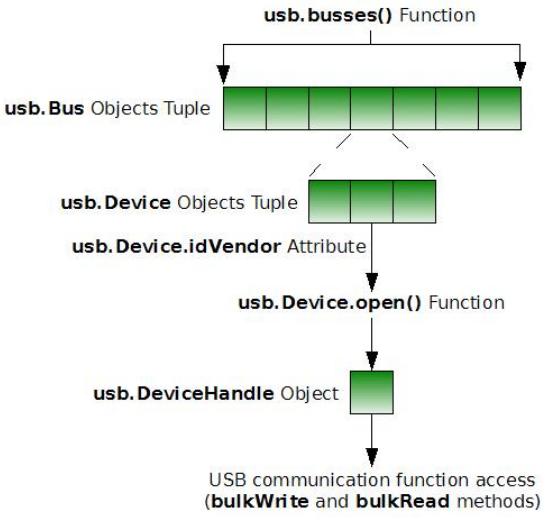

Figura 3.6: Esquema de classes definida per la llibreria *pyUSB*.

Per poder establir la connexió amb el dispositiu USB4000 s'han de recórrer tots els objectes tipus *Bus* (corresponen als ports USB dels que es disposen) i per cada un recórrer tots els objectes tipus *Device* (dispositius USB connectats en el moment de la cerca) en busca d'un dispositiu amb l'atribut *idVendor* buscat. Un cop trobat el dispositiu (un objecte *Device* segons la llibreria pyUSB) apliquem el seu propi mètode *open()* obtenint així un objecte de tipus *DeviceHandle*, la única classe que realment disposa del mètodes de lectura/escriptura sobre la comunicació USB. Amb la crida del mètode *open()* i l'assignació de l'objecte *DeviceHandle* a una variable per poder accedir-hi podem donar per establerta la connexió amb el dispositiu.

Per poder realitzar qualsevol de les accions permeses sobre el dispositiu (configurar el temps d'integració, lectura de dades espectromètriques, etc.) és precís utilitzar els mètodes *bulkRead* i *bulkWrite* de la classe *DeviceHandle* i seguir els paràmetres de configuració marcats pel fabricant. Primer de tot és necessari enviar la comanda corresponent a l'acció que es vol realitzar (en el *datasheet* del dispositiu apareixen en la taula *USB Command Summary*) seguint el següent format:

DeviceHandle.bulkWrite(endpoint, comanda)

El mètode *bulkWrite* requereix dos paràmetres d'entrada. El primer paràmetre correspon a la direcció de l'*endpoint* a utilitzar per l'escriptura de comandes i el paràmetre comanda corresponent al codi hexadecimal de l'acció a realitzar un cop convertida a format *char*. Per exemple, en el cas de voler canviar el temps d'integració aquests dos valors correspondrien a 0x01 i 0x02 (prèviament passat a format char) respectivament.

Anàlogament, per llegir qualsevol informació provinent de l'espectròmetre s'ha d'utilitzar el mètode *bulkRead*, precedit de la utilització del mètode *bulkWrite* amb la comanda d'alguna acció de lectura (0x09 pel cas de lectura de dades espectrals). La crida de mètode

bulkRead, molt similar al mètode *bulkWrite*, segueix el següent format:

lectura=DeviceHandle.bulkWrite (endpoint, bytes)

Com podem veure, el mètode *bulkRead* també necessita dos paràmetres d'entrada, corresponents a la direcció de l'endpoint i al nombre de bytes que esperem llegir, respectivament.

Els *endpoints* als que hem anat fent referència queden definits en el *datasheet* del fabricant i tenen relació amb el protocol de comunicació USB. Tota connexió USB està formada per connexions unidireccionals (s'aconsegueix la bidireccionalitat ja que algunes connexions són d'entrada i altres de sortida) i cada *endpoint* fa referència a una d'aquestes connexions, associada cada una d'elles a una o diverses funcions del dispositiu.

La següent taula mostra informació detallada sobre els endpoints associats al dispositiu USB4000 i les funcions relacionades:

ENPOINT	**FUNCIÓ**
0x01 (OUT)	Enviar instruccions al dispositiu
0x82 (IN)	Adquisició de dades espectromètriques (píxels 1024-3840 i paquets sync)
0x86 (IN)	Adquisició de dades espectromètriques (píxels 0-1023)
0x81 (IN)	Adquisició de la resta de dades (configuració, etc.)

Taula 3.1: Funcions del USB4000 i endpoints associats.

Cal tenir en compte que qualsevol lectura s'han de llegir tots els seus bits (encara que no els utilitzem), del contrari aquests bits romanen en el *buffer* i provoquen que el dispositiu no accepti cap altre petició de lectura i/o escriptura.

Pel control de la comunicació USB entre el PC i el dispositiu USB4000 s'ha implementat una llibreria genèrica amb les principals funcionalitats relacionades amb l'adquisició de dades del dispositiu USB4000 anomenada *OOS4000v2.py*. S'ha intentat realitzar una llibreria de funcions el més genèrica possible per poder ser utilitzada per altres aplicacions relacionades amb el dispositiu USB4000. Els mètodes d'aquesta llibreria queden descrits a continuació:

- ***OpenDevice:*** retorna un objecte de tipus HandleDevice (comunicació USB) relacionada amb el dispositiu USB amb el mateix *idVendor* que li hem passat com a paràmetre d'entrada.
- ***ResetDevice:*** Reseteja la comunicació del dispositiu USB que li hem passat com a paràmetre d'entrada. No obtenim cap paràmetre a la sortida.
- ***SetIntTime:*** Configura el nou temps d'integració al dispositiu USB indicat. No obtenim cap paràmetre de sortida.
- ***GetSample:*** Retorna una única adquisició de dades espectromètriques en forma de vector de 3840 valors.
- ***MaxSample:*** Retorna el valor màxim d'un vector.

- ***GetReducedSample:*** A partir d'un vector corresponent als valors d'una adquisició completa s'obté una versió 'reduïda' per poder ser representada en la pantalla de la PDA.

3.2.5. Adquisició de dades

Un cop explicats els detalls de la implementació de la comunicació entre dispositius podem tractar l'apartat importat del funcionament del software de control, l'adquisició.

Tant bon punt els paràmetres de la interfície gràfica s'han configurat es pot posar en funcionament l'adquisició de dades, simplement prement el botó 'Start' de la pestanya d'adquisició. Un cop posat en funcionament l'adquisició tots els paràmetres de configuració son enviats a l'ordinador central a través del port sèrie, indicant d'aquesta manera que es vol iniciar el procés d'adquisició de dades. Aquests paràmetres s'utilitzen per inicialitzar algunes variables necessàries durant el procés d'adquisició, ja que és 'l'ordinador central' el que realment controla tot el procés.

A partir d'aquest moment, amb totes les variables necessàries configurades, l'ordinador central pot començar a fer peticions al dispositiu USB4000 a través del port USB. En aquest procés primer es necessari configurar el valor del temps d'integració i posteriorment realitzar l'adquisició de dades pròpiament dita.

Per poder modificar el temps d'integració hem de seguir el format del mètode *bulkWrite*, comentat anteriorment, tenint en compte que el valor 0x01 correspon a l'*endpoint* d'enviament de comandes i *buftx* correspon al temps d'integració en el format especificat pel fabricant:

DeviceHandle.bulkWrite(0x01,buftx)

En aquesta versió del software moltes adquisicions consecutives tenen el mateix temps d'integració, tot i així abans de fer qualsevol nova adquisició sobreescriurem aquest paràmetre sense potser ser realment necessari, ja que és un valor que el dispositiu USB4000 guarda en memòria. Tot i així, resulta més eficaç i ràpid que fer les pertinents comprovacions. Aquesta estructura s'ha cregut la més adequada tenint en compte futures millores del programa on, per exemple, serà possible que el temps d'integració s'adapti a les condicions de lluminositat ambiental per si sòl.

Un cop realitzada aquesta configuració es pot dur a terme l'adquisició de dades tenint en compte que les dades espectromètriques que esperem rebre segueixen el format especificat pel fabricant.

Cal tenir en compte que no totes les dades adquirides corresponen a informació vàlida, però totes han de ser llegides pel bon funcionament del dispositiu. La informació a desestimar correspon a l'últim paquet, que conté informació de sincronització.

Com en el cas del port sèrie, pel port USB tampoc es poden transmetre dades sinó és en format de byte. Com s'ha comentat anteriorment, el conversor A/D intern del dispositiu USB4000 és de 16 bits, així que a la seva sortida obtenim valors codificables amb 16

# PAQUET	ENDPOINT LECTURA	# BYTES	PÍXELS
0	EP6In (0x86)	512	0-255
1	EP6In (0x86)	512	256-511
2	EP6In (0x86)	512	512-767
3	EP6In (0x86)	512	768-1023
4	EP2In (0x82)	512	1024-1279
5	EP2In (0x82)	512	1280-1535
...	EP2In (0x82)	512	...
14	EP2In (0x82)	512	3584-3840
15	EP2In (0x82)	1	Sync Packet

Taula 3.2: Format de les dades espectromètriques.

bits (2 bytes). Això significa que cada un dels 3840 valors corresponents a la intensitat lumínica captada per cada fotodíode es transmeten en forma de 2 bytes i que un cop les dades són llegides per l'ordinador central han de ser reestructurades altre cop.

Arribats a aquest punt, si així l'usuari ho ha configurat, es guarden en format NetCDF. Realment les dades es guarden en fitxer cada un cert nombre de mostre però sempre són guardades en una variable temporal per ser emmagatzemades en fitxer segons els paràmetres configurats per l'usuari.

Després del seu emmagatzemament, les dades espectromètriques són transmeses al dispositiu PDA un cop l'ordinador central ha passat les dades a un format representable per la pantalla de la PDA (es necessari reduir el nombre de valors a representar).

El sistema segueix adquirint dades del dispositiu USB4000 i representant-les en la PDA i emmagatzemant-les en arxius NetCDF seguint la configuració de l'usuari fins que no rep les ordres d'aturar aquest procés. L'aturada del procés pot venir provocada per haver adquirit el nombre de mostres configurat o que l'usuari hagi decidit aturar el procés prement el botó 'Stop' de la PDA.

Tot el procés de comunicació entre el PC i la PDA per a la representació de dades espectromètriques està recolzat per un sistema de control d'errors, a través de l'enviament de paquets ACK per la confirmació de l'arribada de paquets i el camp de control CRC per assegurar-ne la integritat. A més a més, existeixen sistemes de retransmissió de paquets fins al punt de poder aturar el sistema si es creu que existeixen masses errors de transmissió.

Aquest sistema de control d'erros no va tant encaminat a salvaguardar les dades representades sinó a detectar errors en la comunicació, tot i que pel tipus de comunicació actual no és usual obtenir errors.

Un cop aturat el procés d'adquisició podem recuperar les dades emmagatzemades en format NetCDF accedint als sistema de fitxer del PC central on s'han anat guardant. Les dades en aquest tipus de fitxers no són accessibles directament ja que la informació és guardada en format binari. Per poder-hi accedir es precís utilitzar algun programa pel tractament d'aquest tipus de dades o la creació de rutines amb aquesta finalitat, en Python per exemple. L'actual software no té aquesta funció implementada.

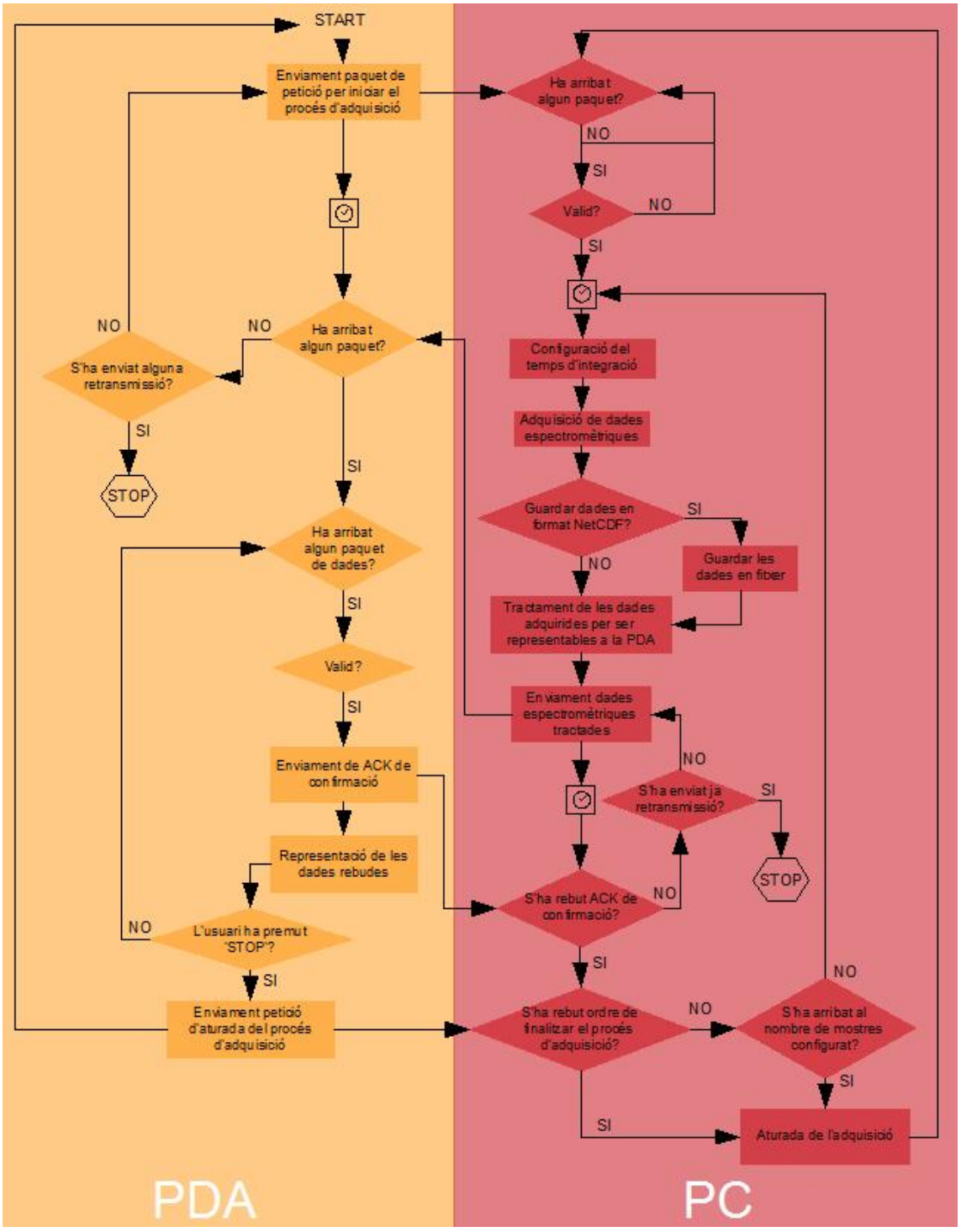

Figura 3.7: Diagrama de flux del procés d'adquisició.

3.2.6. Representació gràfica

La representació gràfica de les dades obtingudes de l'espectròmetre USB4000 ens permet controlar visualment el correcte funcionament del dispositiu. Degut a les dimensions de la pantalla de la PDA, es disposa d'una espai de 240x120 píxels per la representació d'aquestes dades, que provoca que les dades obtingudes hagin d'exprerimentar algun tipus de compressió.

La única finalitat de la representació gràfica és el control del funcionament del dispositiu, així que s'ha aplicat la compressió més senzilla possible. Si dels 3840 valors únicament se'n representa un de cada 16 obtenim una lectura de 240 valors, però d'amplitud no representable, encara.

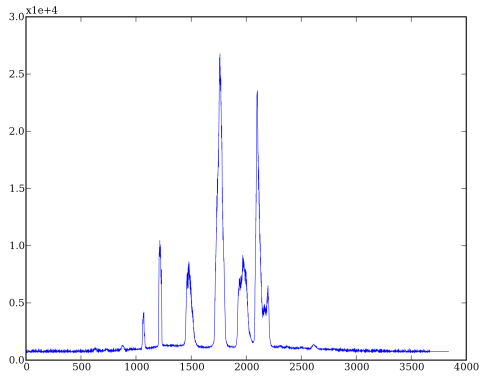

Figura 3.8: Representació de les dades obtingudes.

Per adaptar aquesta l'amplitud de les dades obtingudes a un valor màxim de 120 assignem aquest valor al valor màxim de la mostra i donem un valor a la resta de mostres en comparació amb aquest valor, obtenint així una representació gràfica que sempre ocuparà tota la pantalla disponible.

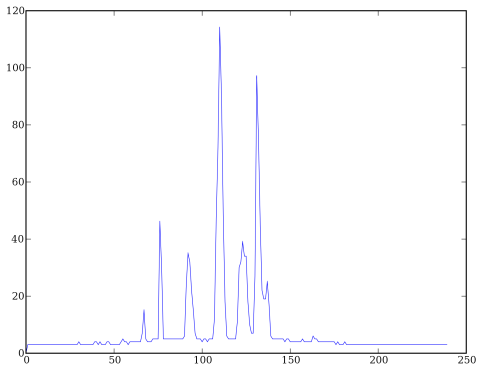

Figura 3.9: Representació de les dades simulant la resolució de la pantalla de la PDA.

El fet que en cada representació canvii el fons d'escala provoca que es perdi la referencia de la intensitat lumínica rebuda, tot i que una bona referència pot ser fixar-se en el nivell d'*offset* provocat pel soroll superposat al senyal.

Tot i la reducció de resolució que suposa aquesta adaptació de les dades a la pantalla de la PDA, es poden seguir reconeixent visualment les característiques de les dades representades. Aquesta representació en la PDA és simplement de control del i les dades que realment són emmagatzemades, pel seu posterior tractament, són la versió completa de les dades adquirides.

3.2.7. Emmagatzemament de les dades

L'emmagatzemament de dades no és una funcionalitat per defecte de l'aplicació, sinó seleccionable. Si es desitja que les dades adquirides es guardin (actualment només existeix la possibilitat que les dades siguin guardades en format netCDF) només s'ha de seleccionar aquesta opció en la interfície gràfica de la PDA i introduir els valors de configuració.

Els paràmetres configurables per l'usuari referent a l'emmagatzemament de dades són '*Samples per File*' i '*Backup Time*', corresponents al nombre màxim de mostres que pot contenir un fitxer i cada quantes mostres es vol fer el bolcat de les dades en el fitxer, respectivament. Podem entendre el nombre de mostre de *backup* com el nombre màxim de mostres que s'està disposat a perdre si en un moment donat la comunicació falla, el sistema es queda bloquejat, etc.

Les tasques relacionades amb la creació d'arxius NetCDF són realitzades per l'ordinador central que les guarda en memòria. Les funcionalitats relacionades amb l'emmagatzemament de dades estan descrites en un fitxer específic desenvolupat a mida per l'aplicació i, a diferència de molts altres programes d'emmagatzemament de dades en format NetCDF, permeten guardar les dades a mesura que es van adquirint i així fer un millor ús de la memòria utilitzada. Aquesta llibreria específica, *SaveNetCDF.py.* conté una única classe amb els seus mètodes per la generació i gestió d'arxius.

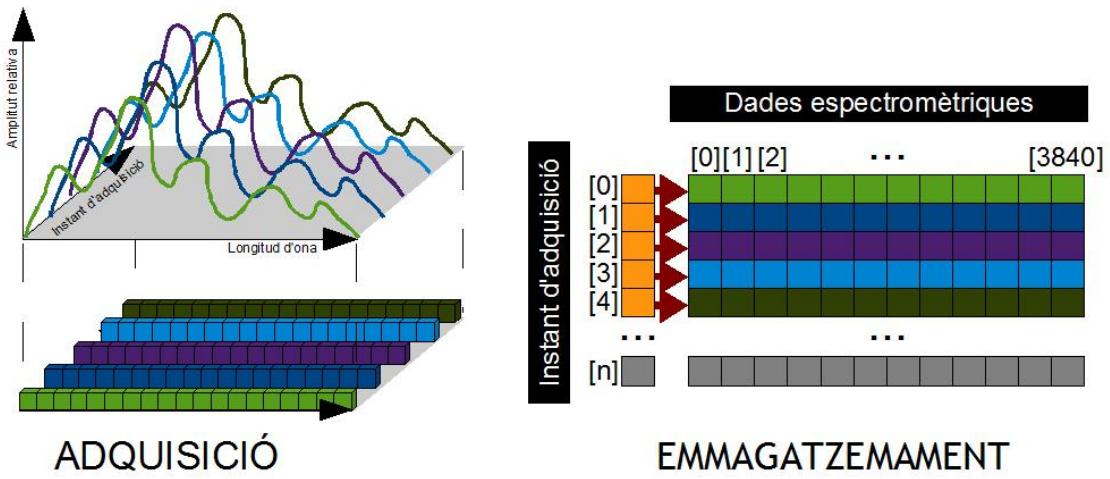

Figura 3.10: Estructura de les dades emmagatzemades en format NetCDF.

Les dades emmagatzemades es divideixen bàsicament en metadades i dades adquirides. Les metadades són paràmetres descriptius que s'han configurat per defecte (l'usuari no pot modificar) i contenen informació sobre el dispositiu d'adquisició de dades, el centre

al qual pertanyen, etc. Les dades equivalents a una mostra són emmagatzemades en vectors de 3840 valors que equivalen a una mostra i el conjunt de mostres es troben relacionades amb un vector amb els respectius instants d'adquisició de les dades. La pròpia estructura del NetCDF relaciona cada mostra amb el seu instant d'adquisició.

L'instant d'adquisició correspon al valor del temps POSIX corresponent a l'inici de l'adquisició obtingut a través de la llibreria *time* de Python per l'obtenció de variables referents al temps en diferents formats. El temps POSIX és un sistema per la descripció d'instants de temps i que utilitza el nombre de segons transcorreguts des d'un instant de temps concret (l'1 de Gener de 1970 es considera el temps POSIX zero) com a referència temporal. Aquest temps facilitarà en un futur la ordenació cronològica de les mostres obtingudes, la cerca d'una mostra concreta o la selecció d'un determinat grup de mostres.

En el format del nom dels fitxers s'ha seguit amb la idea de facilitar la localització i reordenació de les mostres obtingudes. El propi nom del fitxer conté la data i l'instant d'adquisició de la primera mostra continguda en el fitxer. El nom del fitxer segueix el format 'SPYYYYMMDDThhmmss.nc'.

A continuació es detalla el significat del format del fitxer de dades:

- 'SP' : indica que el fitxer conté dades espectromètriques
- 'YYYY' : indica l'any d'adquisició de la primera mostra del fitxer.
- 'MM' : indica el mes d'adquisició de la primera mostra del fitxer.
- 'DD' : indica el dia d'adquisició de la primera mostra del fitxer.
- 'T' : separador entre els dígits referents a data i els referents a temps.
- 'hh': indica l'hora d'adquisició de la primera mostra del fitxer.
- 'mm': indica el minut d'adquisició de la primera mostra del fitxer.
- 'ss': indica el segon d'adquisició de la primera mostra del fitxer.
- '.nc': format de fitxer dels arxius NetCDF.

Així ,per exemple, una mostra adquirida l'1 de Juny del 2008 a les 11:40:18, sent aquesta la primera mostra del fitxer, és guardaria en el fitxer amb nom 'SP20080601T114018.nc'. A part de poder localitzar i reordenar les dades adquirides aquest format també té l'avantatge que les dades queden ordenades cronològicament en el directori, de més actuals a menys.

Tot i els avantatges comentats sobre la capacitat autodescriptiva dels arxius NetCDF, aquests presenten un clar desavantatge ja que la memòria utilitzada per aquest tipus de fitxers en relació amb el nombre de mostres emmagatzemades és molt elevada.

En aquesta versió de l'aplicació no s'ha pres cap mesura per tal de solucionar aquest desaprofitament de la memòria però es pretén utilitzar algun sistema de compressió en futures versions.

NOMBRE DE MOSTRES	MIDA DEL FITXER NETCDF (MB)
100	1,5
200	2,9
500	7,3
1000	14,7
5000	73,3

Taula 3.3: Mida del fitxer NetCDF resultant en funció del nombre de mostres.

3.2.8. Control d'execució mitjançant Threads

Encara que no s'ha comentat fins ara, pràcticament tota l'aplicació queda descrita i controlada per objectes de tipus *thread*. La teoria sobre l'aplicació de *threads* en el desenvolupament software és que permeten l'execució de diversos processos o diferents fils d'execució en paral·lel, encara que a la realitat es tracta més d'una compartició temporal del temps de processador per part dels diferents processos i no que el processador els executi a la vegada.

En el nostre cas, no s'ha aprofitat al màxim les possibilitats que ofereix treballar amb aquesta classe d'objectes, ja que poques tasques del software es realitzen el paral·lel, però l'estructura actual del software queda preparada per la inserció d'altres sensors i/o funcionalitats i que es puguin realitzar en paral·lel.

Els *threads*, a part de les seves característiques pròpies, tenen altres objectes relacionats, que en el nostre cas seràn els que realment ens interessaran, com ara són els *Events* i els *Timers*.

Els objectes de tipus *Event* constitueixen el sistema més simple de comunicació entre *threads* i permeten disparar l'execució d'una determinada funció, ja que a partir de *flags* interns que s'activen o es desactiven donada una determinada condició poden donar pas a l'activació d'un altre *thread*, per exemple. La idea de funcionament dels *Events* respecte als *Threads* és similar al funcionament d'una màquina d'estats on les entrades que fan variar d'estat són els *flags* i l'estat en si equival a l'execució d'un *thread*.

Els objectes de tipus *Timer* tenen les característiques dels objectes de tipus *Threads*, de fet és tracten de threads amb característiques especials. Els objectes Timer s'utilitzen per cridar a una determinada funció un cop ha transcorregut el temps configurat. Utilitzants aquest tipus d'objectes és com es realitza l'adquisició de dades i es configura l'espaiat temporal entre mostres.

Els threads i els events s'han utilitzat en el control de la comunicació sèrie per poder mantenir el port sèrie a l'espera de l'arribada de paquets correctes que disparen la l'execució d'una determinada funcionalitat, com en el cas de les classes *WaitingHeader* o *WaitingPacket*.

CAPÍTOL 4. CONCLUSIONS

El software desenvolupat constitueix una eina senzilla de control del dispositiu USB4000 utilitzant un dispositiu PDA com a element de control, tal com es va marcar en els objectius, sent l'únic projecte d'aquestes característiques del que es té constància.

Com ja s'ha comentat en diversos apartats, tot el desenvolupament s'ha realitzat utilitzant eines de software lliure seguint les directives de la UTM per desenvolupar aplicacions d'interès científic a través d'eines de lliure distribució. Aquestes eines tenen l'avantatge de no estar lligades a cap tipus de llicència de pagament, a diferència d'altres d'eines encara utilitzades per la UTM com Matlab. Degut a que l'aplicació es pensa distribuir sota llicència GPL (codi obert) permet que aquesta quedi oberta a futures modificacions. Aquestes futures modificacions, per tant, poden ser no tant sols dins de l'àmbit de la UTM sinó també a través de col·laboracions externes.

El màxim interès de l'aplicació desenvolupada no es centra en la pròpia aplicació sinó en la nova línia de desenvolupament que inicia, ja que és la primera experiència de la UTM en el desenvolupament de software sobre plataformes PDA i la utilització d'aquest model d'espectròmetre en la realització d'una eina pròpia. Aquesta eina suposa una bon punt de partida en la creació d'una alternativa al software propietari i, a més a més, posa de manifest les possibilitats actuals en el camp del desenvolupament amb software lliure en dispositius mòbils, tema no extensament documentat per ser encara minoritari.

Un altre punt d'interès de l'aplicació és el fet d'haver aconseguit realitzar la comunicació amb el dispositiu USB4000 a través del port USB per tal de poder realitzar la seva configuració i la posterior obtenció de dades espectromètriques per poder ser representades i/o emmagatzemades. Aquest no ha sigut un punt senzill en el desenvolupament, com s'ha comentat anteriorment, no s'ha pogut tenir accés al codi font de l'aplicació del fabricant al tractar-se de software propietari i el datasheet del dispositiu és un document poc explicatiu per si sòl, ja sigui per tractar-se d'un producte molt nou o pel poc interès del propi fabricant. Amb l'ajut de les llibreries python especialitzades en la comunicació USB i el coneixement adquirit sobre la comunicació amb el dispositiu USB4000 s'ha desenvolupat un llibreria genèrica amb les principals funcions que es poden dur a terme sobre aquest dispositiu.

L'aplicació es tracta també del primer projecte de desenvolupament de la UTM que utilitza la comunicació USB amb la intenció d'acabar desbancant el port sèrie per la comunicació entre dispositius, ja que la comunicació sèrie cada cop es troba més en desús i molts ordinadors portàtils, per exemple, ja no duen aquest tipus de port sortint de fàbrica. La comunicació USB permet la connexió de més d'un dispositiu per port, no és tant sensible a desconnexions i a més a més permet velocitats de transmissió de dades més altes a través d'una comunicació tant o més robusta que en el cas del protocol RS-232.

El funcionament de l'aplicació s'adapta a les característiques de l'adquisició de dades espectromètriques destinades a la caracterització de la columna d'aigua a través de la realització d'adquisicions, d'una forma més o menys indefinida separades per un espaït de temps constant configurable, cosa que el software del fabricant de l'espectròmetre no permet. En el desenvolupament del software s'ha apostat per la utilització de threads el que

implica una clara modulabilitat del codi, facilitant la inserció de nous dispositiu en l'aplicació i, entre altres aspectes, permet la representació gairebé en temps real de les dades adquirides. Tot i així, podem dir que el control d'errors i retransmissions dissenyat en la transmissió de dades entre l'ordenador central i el dispositiu PDA està sobredimensionat per la comunicació actual entre dispositius (comunicació per cable) però s'ha deixat configurat per a la realització de futures proves utilitzant comunicacions sense fils (Bluetooth).

Finalment comentar un aspecte innovador relacionat amb el desenvolupament software relacionat amb l'emmagatzemament de dades en format NetCDF. Normalment aquest tipus d'estàndard s'utilitza en l'emmagatzemament de dades en format científic estàndard un cop es compta amb la totalitat de dades a guardar. En canvi, el software desenvolupat emmagatzema les dades espectromètriques d'una forma dinàmica, a mesura que aquestes són adquirides, el que evita la pèrdua de la totalitat de les dades si es produís algun error de funcionament de l'aplicació.

Tot i els resultats satisfactoris obtinguts, la versió actual és una primera aproximació a un software d'adquisició realment aplicable i per tant, és susceptible de millores, però també un bon punt de partida per altres projectes i aplicacions. S'ha posat especial interès al respecte documentant el codi desenvolupat dins del propi codi per facilitar-ne la comprensió i les futures modificacions.

Pel que fa a l'impacte mediambiental de l'aplicació, remarcar que l'aplicació no produeix cap efecte advers en el medi ni s'han generat cap mena de residus en el seu desenvolupament, ja que tot el desenvolupament a sigut a nivell software. A més a més de tractar-se d'un sistema de mesures gens invasiva pel medi, es tracta d'una eina que pot ser utilitzada per detectar i prevenir possibles desastres ecològics a través de la detecció de contaminants, proliferacions anormals de microorganismes i de l'estudi de la salud dels oceans en general.

4.1. Futures millores

Com a tota primera versió d'un projecte, i més sent part d'un projecte de I+D, és lògic pensar que el desenvolupament d'aquesta eina no és un procés finalitzat i per tant, queda oberta a millores.

Dintre de les possibles millores més directes trobem les que s'han tingut en ment durant el desenvolupament del projecte però que no s'han pogut arribar a realitzar per falta de temps. Com ja s'ha comentat, el codi s'ha configurat per la realització de futures proves amb la utilització de tecnologia Bluetooth entre la PDA i l'ordinador central. Fruit d'aquestes proves es pretén obtenir informació sobre si aquest tipus de comunicació és viable en la nostra aplicació i quines serien les modificacions necessàries en el sistema de control d'errors de transmissió per fer aquesta comunicació prou robusta.

Una opció interessant a incloure en el sistema d'adquisició és la possibilitat de realitzar adquisicions utilitzant un control automàtic del temps d'adquisició de manera que el propi dispositiu ajustés aquest valor per obtenir el major aprofitament del marge dinàmic. Una versió més desenvolupada també podria ser capaç d'ajustar l'espaiat temporal entre mostres al detectar major variació en les dades obtingudes.

Un altre aspecte millorable és l'espai en memòria necessari per emmagatzemar les dades en format NetCDF. Com ja hem comprovat, els arxius presenten el gran avantatge de ser arxius autodescriptius però a canvi de veure incrementat notablement l'espai necessari per l'emmagatzemament de les dades. Una de les solucions plantejades és la utilització d'algun tipus de compressió, ja sigui del propi arxiu NetCDF o de les dades adquirides abans de ser emmagatzemades.

El llenguatge Python ofereix la possibilitat d'utilitzar l'estructura de control *try* que permet 'pescar' els errors ocorreguts durant l'execució del programa i realitzar una determinada acció per cada tipus d'error ocorregut. Això permetria un funcionament més robust de l'aplicació davant d'erros de sistema o com a mínim poder avisar a l'usuari dels problemes existents i presentar l'opció de reiniciar el sistema, però s'han d'acabar d'estudiar les possibilitat i els inconvenients d'aquest tipus d'estructures.

També són interessants millores en el funcionament i configuració del sistema però aquestes són realitzables a més llarg termini, relacionades amb noves funcionalitats i evolucions més sèries del sistema.

Les sondes oceanogràfiques acostumen a contenir més d'un tipus de sensor per intentar caracteritzar al màxim la capa d'aigua que travessen i així aprofitar millor les costoses campanyes a alta mar. En el nostre cas seria interessant introduir sensors de pressió i algun tipus d'indicador de posicionament per arribar a aconseguir localitzar les dades adquirides en funció de longitud, latitud i profunditat.

Una limitació actual del sistema són la durada de les bateries del dispositiu PDA. El sistema s'ha dissenyat perquè pugui adquirir dades durant molt de temps però les bateries de la PDA pel control de l'adquisició no poden durar tant de temps. És veritat que aquest dispositiu es pot connectar al corrent però llavors deixaria de ser una eina funcional. Una possible solució passa configurar l'aplicació perquè la PDA pugui realitzar peticions asíncrones a l'ordinador central per tal de configurar paràmetres, consultar l'estat de l'adquisició actual, etc.,mentre la resta de temps es troba en *stand by* aprofitant millor la vida de les bateries.

Aquests són alguns del exemples de possibles millores en les que s'està o es pretén treballar encara que algunes poden tenir resultats a molt llarg termini. Com ja se sap, el procés d'innovació és lent, complex i sense final clar.

BIBLIOGRAFIA

[1] *USB4000 Data Sheet*.[PDF]
Accessible a l'adreça:
http://www.oceanoptics.com/Technical/engineering/USB4000%20OEM%20Data%20Sheet.pdf

[2] van Rossum, Guido.
Python Reference Manual (Release 2.5).[PDF]
(19th September, 2006)
Accessible a l'adreça:
http://www.dil.univ-mrs.fr/ garreta/PythonBBSG/docs/ref.pdf

[3] Matloff, Norman ; Hsu,Francis
Tutorial on Threads Programming with Python.[PDF]
(April 11, 2007)
Accessible a l'adreça:
http://heather.cs.ucdavis.edu/ matloff/Python/PyThreads.pdf

[4] *Python Programming Language – Official Website*.[WEB]
Accessible a l'adreça:
http://www.python.org/

[5] van Rossum, Guido.
Python Tutorial.[WEB]
(21st February, 2008)
Accessible a l'adreça:
http://docs.python.org/tut/tut.html

[6] *Qt Reference Documentation (Open Source Edition)*.[WEB]
Accessible a l'adreça:
http://doc.trolltech.com/3.3/

[7] *Qtopia Desktop*.[WEB]
Accessible a l'adreça:
http://doc.trolltech.com/qtopia2.1/html/qtopiadesktop-overview.html

[8] *Tripod Data System (Recon)*.[WEB]
Accessible a l'adreça:
http://www.tdsway.com/products/recon

APÈNDIXS

APÈNDIX A. SOFTWARE DESENVOLUPAT

A.1. Llibreria S4000AppPC.py

```
"""
-------------------------------------------------------------------------------------------
    (C)2008 Núria Pujol Vilanova <nuriapujolvilanova@gmail.com>

    FILE NAME: S4000AppPC.py

    DESCRIPTION:

        Simple PC program to control communication between PDA and USB4000 spectrometric device
        (Ocean Optics).

        Communication scenario:

        ------------------             --------------------          ---------
        |  PDA Software   |<///RS232///>|   PC Software    |<///USB///>|USB4000|
        |(S4000AppPDA.py) |             |  (S4000AppPC.py) |           ---------
        ------------------             --------------------

        This new version is an AppPda.py improvement divided into 2 files (S4000AppPC.py and
        S4000AppPDA.py)

        NO CONFIGURABLE PARAMETERS (TO DO*)

    GPL License
-------------------------------------------------------------------------------------------
"""
from struct import pack,unpack
from numpy import array
from random import uniform
from time import time,strftime
from serial import Serial
from OOS4000v2 import*
from Scientific.IO.NetCDF import*
from CRC import*
from SaveNetCDF import*
import threading

#-----------------------------------------------------------------------
####MANAGING THREADS CLASSES#################
#-----------------------------------------------------------------------

#-----------------------------------------------------------------------
#-- This classes should be defined in other file (TO DO)*
#-----------------------------------------------------------------------

class WaitingPacket(threading.Thread):
    def __init__(self,timeout,ser,length):
        """
        ------------------------------------------------------------------------
        WaitingHeader(timeout,ser,length) -- INIT

        DESCRIPTION:

        WaitingHeader thread 'wait' until a header is found ('*' is received by
        serial port) and are available a minimum of bytes ('length') to be read
        or timeout occurs-

        ARGUMENTS:

        timeout -- Maximum waiting packet (header + minimum bytes to read)
        time in seconds. This parameter can be 'None' and thread wait indefinitely
        ser -- Serial connection object.
        length -- Minimum packet length waiting for
```

```
        Has start() and exit() methods.
        ----------------------------------------------------------------------------
        """
        threading.Thread.__init__(self)

        self.ser=ser #--Serial communication

        self.length=length
        self.timeout=timeout

        self.Found=threading.Event()

        #----------------------------------------------------------------------------
        #-- 'Found' is set when a header is found and there are a minimum of waiting
        #-- bytes on serial buffer.
        #----------------------------------------------------------------------------

        self.Exit=threading.Event()

        #----------------------------------------------------------------------------
        #-- 'Exit' is set when program has wait too much time and any packet has
        #-- been received
        #----------------------------------------------------------------------------

        self.timer=threading.Timer(self.timeout,self.exit)

        #----------------------------------------------------------------------------
        #-- self.timer is a Timer object that call exit() thread method when
        #-- thread has been waiting 'timeout' seconds until start() method
        #-- function has been call
        #----------------------------------------------------------------------------

    def run(self):
        """
        ----------------------------------------------------------------------------
        WaitingPacket -- start() METHOD

        DESCRIPTION:

        Count the number of waiting bytes on serial port until 'length' bytes are
        available and in this available bytes found a header ('*') (Found event is
        set) or has been waiting 'timeout' (init parameter) seconds (Exit event
        is set).
        ----------------------------------------------------------------------------
        """
        self.timer.start()

        while (self.Found.isSet()==False)and(self.Exit.isSet()==False):
            if self.ser.inWaiting()>=self.length:
                byte=self.ser.read(1)
                if byte=='*':
                    self.Found.set()
                    self.timer.cancel()

    def exit(self):
        """
        ----------------------------------------------------------------------------
        WaitingPacket -- exit() METHOD

        DESCRIPTION:

        Set Exit event when is called by self.timer (Timer Object). One of the
        possible events to terminate WaitingPacket thread (Waiting too much time).
        ----------------------------------------------------------------------------
        """
        if self.timeout!=None:
            self.Exit.set()

        #print "Non packet Found" #DEBUG

class Acquiring(threading.Thread):
    def __init__(self,packet,ser):
```

```
        """
        ------------------------------------------------------------------------
        Acquiring(packet,ser) -- INIT

        DESCRIPTION:

        Acquiring thread manages all acquiring continuous process on PC side when
        and 'AD'packet is received from PDA.

        ARGUMENTS:

        packet -- Packet on string format received from PDA
        ser -- serial communication object

        Has start() and Acquire() method
        ------------------------------------------------------------------------
        """
        threading.Thread.__init__(self)
        self.packet=packet
        self.ser=ser
        self.received=unpack('c2sHiHHHH',self.packet)#unpack information

        #------------------------------------------------------------------------
        #-- From 'packet' unpacked information some variables are initialized
        #-- (acquisition necessary parameters)
        #------------------------------------------------------------------------

        self.nsamples=self.received[2]
        self.itime=self.received[3]
        self.interval=self.received[4]
        self.SamplesxFile=self.received[5]
        self.ToSave=self.received[6]
        self.buf_tx=chr(0x02)+ packet[8:12]
        self.acquired=0

        #------------------------------------------------------------------------
        #-- From 'packet' unpacked information some variables are inicialized
        #-- (acquisition necessary parameters)
        #------------------------------------------------------------------------

        #------------------------------------------------------------------------
        #-- If SamplesxFile!=0 and ToSave!=0 data has to be stored in NetCDF every
        #-- 'ToSave' samples in 'SamplesxFile' samples in a NetCDF File maximum
        #-- Some variables to store samples not stored yet has to be initialized
        #-- (minimum length necessary)
        #------------------------------------------------------------------------

        if self.SamplesxFile!=0 and self.ToSave!=0:
            #print "NetCDF file created" #DEBUG
            self.datafile=DataFile(self.ToSave,self.SamplesxFile)

        self.Exit=threading.Event()

    def run(self):
        while (self.Exit.isSet()==False):
            self.AdCon=threading.Timer(self.interval,self.Acquire)
            self.AdCon.start()
            self.AdCon.join()
            self.AdCon.cancel()

    def Acquire (self):

        #------------------------------------------------------------------------
        #-- One acquisition is done. Spectral signature data and what 'time'
        #-- has been done are obtained
        #------------------------------------------------------------------------

        [specsignature,gtime]=GetSample(self.buf_tx,self.itime)

        if self.SamplesxFile!=0 and self.ToSave!=0:

        #------------------------------------------------------------------------
        #-- Data has to be saved on NetCDF format
```

```
        #-----------------------------------------------------------------------

            self.datafile.SaveFile(self.acquired,specsignature,gtime)

        #print gtime #DEBUG

        idSample=int(uniform(0,100))

        ShortSignature=GetReducedSample(specsignature,screenV=100,screenH=240)
        tosend=pack('c2sB','*','IN',idSample)

        for y in range(len(ShortSignature)):
            tosend+=chr(ShortSignature[y])
        tosend+=pack('H',crc16(tosend))

        self.ser.write(tosend)

        #-----------------------------------------------------------------------
        #-- Info to 'draw' on PDA has been sent to
        #-----------------------------------------------------------------------

        #print "Spectral signature data sent" #DEBUG

        correct=WaitingAK(3,idSample,tosend,self.ser)

        #-----------------------------------------------------------------------
        #-- Valid ACK packet received
        #-----------------------------------------------------------------------

        if correct==1:

            self.acquired=self.acquired+1
            if self.acquired>=self.nsamples and self.nsamples != 0:
                #-------------------------------------------------------------------
                #-- Acquiring process has to be stopped
                #-------------------------------------------------------------------
                #print "self.acquired value:" #DEBUG
                #print self.acquired #DEBUG
                tosend='*EX'
                self.ser.write(tosend)

                #print "Acquiring process end" #DEBUG

                #-------------------------------------------------------------------
                #--    ### NETCDF FUNCTIONS ###
                #-------------------------------------------------------------------

                if self.SamplesxFile!=0 and self.ToSave!=0:
                    self.datafile.SaveEnd(self.acquired)
                self.Exit.set()

        if correct==2:

            #-----------------------------------------------------------------------
            #--    ### NETCDF FUNCTIONS ###
            #-----------------------------------------------------------------------

            #print "Hem rebut un exit" #DEBUG
            #print self.acquired #DEBUG

            if self.SamplesxFile!=0 and self.ToSave!=0:
                self.datafile.SaveEnd(self.acquired)

            self.Exit.set()

def WaitingAK(maxret,idSample,tosend,ser):
    """
    -------------------------------------------------------------------------
    WaitingAK(ser,maxret,idSample,tosend)

    DESCRIPTION:

    WaitingAK function wait a valid 'ACK' packet (valid idSample and CRC16)
```

```
and retransmit sample data if its necessary

ARGUMENTS:

ser -- serial communication object
maxret -- maximum number of retransmissions
idSample -- ACK packet identifier waiting for
tosend -- information to be retransmitted if its necessary

WaitingAK is used next information has been sent

WaitingPacket thread class is used
------------------------------------------------------------------------
"""
WaitingACK=WaitingPacket(2,ser,5)
WaitingACK.start()
WaitingACK.join()

if WaitingACK.Exit.isSet():
    #------------------------------------------------------------------
    #-- 2 seconds waiting and nothing valid received.
    #-- Waiting time has to be optimized (TO DO*)
    #------------------------------------------------------------------

    #print "Waiting too much" #DEBUG

    if maxret>0:
        ser.write(tosend)
        maxret=maxret-1

        #print "Retransmisions yet availabe" #DEBUG
        #print maxret #DEBUG

        WaitingAK(maxret,idSample,tosend,ser)
        #------------------------------------------------------------------
        #-- ACK valid packet waiting again
        #------------------------------------------------------------------

    else:

        #print "Too many errors"

        #------------------------------------------------------------------
        #-- This 'error' not catched (TO DO*)
        #------------------------------------------------------------------

        return 0

elif WaitingACK.Found.isSet():

    #------------------------------------------------------------------
    #-- One possible packet found
    #------------------------------------------------------------------

    packettype=ser.read(2)

    if packettype=='AK':

        #------------------------------------------------------------------
        #-- ACK packet received
        #------------------------------------------------------------------

        #print "ACK packet received" #DEBUG

        packet="*AK"+ser.read(3)

        if (checkcrc16(packet))==1:

            print "Valid ACK"
            return 1

        else:
```

```
                #print "Non valid ACK Received" #DEBUG

                WaitingAK(maxret,idSample,tosend,ser)

                #-------------------------------------------------------------
                #-- ACK valid packet waiting again
                #-------------------------------------------------------------

        elif packettype==’EX’:

            #-------------------------------------------------------------------
            #-- EXIT packet received
            #-------------------------------------------------------------------

            tosend=’*EX’
            ser.write(tosend)

            return 2

        else:

            WaitingAK(maxret,idSample,tosend,ser)

        #-----------------------------------------------------------------------
        #-- ACK valid packet waiting again
        #-----------------------------------------------------------------------

def main():

    while(1):

        #ser=Serial(’/dev/ttyS0’,57600)
        ser=Serial(’/dev/ttyS0’,57600)
        TWaiting=WaitingPacket(None,ser,18)

        #print "Waiting packets" #DEBUG

        TWaiting.start()
        TWaiting.join()

        #print "Packet received" #DEBUG

        packettype=ser.read(2)

        if packettype==’AD’:

            #print "’AD’ packet found" #DEBUG

            packet="*AD"+ser.read(17)

            if (checkcrc16(packet))==1:

                #print "Valid AD packet received" #DEBUG

                TAcquiring=Acquiring(packet,ser)
                TAcquiring.start()
                TAcquiring.join()

                #print "Acquisition process stopped" #DEBUG

        #if packettype==’EX’:

            #print "Stop packet received" #DEBUG
            #ser.write(’*EX’)

if __name__ == ’__main__’:
    main()
```

A.2. Llibreria S4000AppPDA.py

```
"""
-------------------------------------------------------------------------------------------
    (C)2008 Núria Pujol Vilanova <nuriapujolvilanova@gmail.com>

    FILE NAME: S4000AppPDA.py

    DESCRIPTION:

        Simple control interface for USB4000 spectrometric device (Ocean Optics)
        in PDA format.

        S4000AppPDA.py is a new PdaApp.py version using another configuration scenario:

        ------------------             --------------------          ---------
        |  PDA Software   |<///RS232///>|   PC Software    |<///USB///>|USB4000|
        |(S4000AppPDA.py) |             |  (S4000AppPC.py) |           ---------
        ------------------             --------------------

        Interface is able to configure some USB4000 parameters and show
        spectral signatures on the screen.

        'Start' and 'Stop' Buttons control main program execution and some
        parameters can be configured by the user but a lot of work is done by
        the computer not PDA.

        Save File on NetCDF format is able in this new version but are stored
        on PC.

        Program use Threads and Event to do the different tasks, specially
        all related task with RS232 communication

    CONFIGURABLE PARAMETERS (using GUI):

        - Number of samples: Number of samples to acquire (if a '0' is introduced
        program enter to continuous acquisition mode until 'Stop' Button is
        pressed).
        - Samples interval: Waiting time from one sample to other (interval
        time).
        - Integration time: USB4000 configuration parameter
        - Configure is files have to be saved (only netCDF available) by selecting
        'Save Data' and configure 'Samples per file' and 'Backup Samples'.

    GPL License

-------------------------------------------------------------------------------------------
"""
import serial, time, sys
import threading
from struct import pack,unpack
from qt import*
from CRC import*

#--------------------------------------------------------------------------
#       ###     MANAGING THREADS CLASSES     ###
#--------------------------------------------------------------------------

#--------------------------------------------------------------------------
#-- This classes should be defined in other file (TO DO)*
#--------------------------------------------------------------------------

class WaitingHeader(threading.Thread):
    def __init__(self,timeout,ser):
        """
        ------------------------------------------------------------------------
        WaitingHeader(timeout,ser) -- INIT

        DESCRIPTION:

        WaitingHeader thread 'wait' until a header is found ('*' is received by
```

```
        serial port) or timeout occurs.

        ARGUMENTS:

        timeout -- Maximum waiting header time in seconds.
        ser -- Serial connection object.

        Has start() and exit() methods.
        ------------------------------------------------------------------------
        """
        threading.Thread.__init__(self)

        self.ser=ser

        #------------------------------------------------------------------------
        #-- Serial communication
        #------------------------------------------------------------------------

        self.Found=threading.Event()

        #------------------------------------------------------------------------
        #-- Found is set when a '*' byte is found (packet headers identifier)
        #------------------------------------------------------------------------

        self.Exit=threading.Event()
        #------------------------------------------------------------------------
        #-- Exit is set when program has wait too much time and any packet has
        #-- been received
        #------------------------------------------------------------------------

        self.timer=threading.Timer(timeout,self.exit)
        #------------------------------------------------------------------------
        #-- self.timer is a Timer object that call exit() thread method when
        #-- thread has been waiting 'timeout' seconds until start() method
        #-- function has been call
        #------------------------------------------------------------------------

    def run(self):
        """
        ------------------------------------------------------------------------
        WaitingHeader -- start() METHOD

        DESCRIPTION:

        Read bytes from serial connection until '*' byte is found (Found event is
        set) or has been waiting 'timeout' (init parameter) seconds (Exit event
        is set).
        ------------------------------------------------------------------------
        """
        self.timer.start()
        while (self.Found.isSet()== False)and(self.Exit.isSet()==False):
            if self.ser.inWaiting()>=3:
                byte=self.ser.read(1)

                if byte=='*':
                    self.Found.set()
                    self.timer.cancel()

    def exit(self):
        """
        ------------------------------------------------------------------------
        WaitingHeader -- exit() METHOD

        DESCRIPTION:

        Set Exit event when is called by self.timer (Timer Object). One of the
        possible events to terminate WaitingHeader thread (Waiting too much time).
        ------------------------------------------------------------------------
        """

        self.Exit.set()
        print "Waiting too much. Header not found" #DEBUG
        self.timer.cancel()
```

```
class WaitingPacket(threading.Thread):
    def __init__(self,timeout,ser,length):
        """
        ------------------------------------------------------------------------
        WaitingPacket(timeout,ser,length) -- INIT

        DESCRIPTION:

        WaitingHeader thread 'wait' until a packet is found (there are many waiting
        byte on serial port as length variable value)or timeout occurs.

        ARGUMENTS:

        timeout -- Maximum waiting packet time in seconds.
        ser -- Serial connection object.
        length -- Packet length waiting for

        *'length' value has to be packet length without header
        (packet length - 1)

        Has start() and exit() methods.
        ------------------------------------------------------------------------
        """
        threading.Thread.__init__(self)
        self.ser=ser
        self.Found=threading.Event()
        self.timer=threading.Timer(timeout,self.exit)
        self.Exit=threading.Event()
        self.length=length

    def run(self):
        """
        ------------------------------------------------------------------------
        WaitingPacket -- start() METHOD

        DESCRIPTION:

        Get waiting packets value until agree with length 'init parameter' (Found
        event is set) or has been waiting 'timeout' (init parameter) seconds (Exit
        event is set).
        ------------------------------------------------------------------------
        """
        self.timer.start()
        while (self.Found.isSet()== False)and(self.Exit.isSet()==False):
            if self.ser.inWaiting()>=self.length:
                self.Found.set()
                self.timer.cancel()

    def exit(self):
        """
        ------------------------------------------------------------------------
        WaitingPacket -- exit() METHOD

        DESCRIPTION:

        Set Exit event when is called by self.timer (Timer Object). One of the
        possible events to terminate WaitingPacket thread (waiting too much time).
        ------------------------------------------------------------------------
        """
        self.Exit.set()
        #print "Waiting too much. Packet not found" #DEBUG
        self.timer.cancel()

class Acquiring(threading.Thread):
    def __init__(self,app):
        """
        ------------------------------------------------------------------------
        Acquiring(app) -- INIT

        DESCRIPTION:

        Acquiring thread manages all acquiring process on PDA side and use
```

```
    WaitingHeader and WaitingPacket threads with this purpose.

    ARGUMENTS:

    app -- PdaApp class object ( GUI)

    Has start() method
    ------------------------------------------------------------------------
    """

    threading.Thread.__init__(self)
    self.self=app

    #------------------------------------------------------------------------
    #-- PdaApp class (GUI) is now Acquiring class self.self variable
    #------------------------------------------------------------------------

def run(self):

    #------------------------------------------------------------------------
    #-- Some PdaApp class (GUI) variables are declared as Acquiring class
    #-- variables
    #-- This should be done on init method (TO DO)*
    #------------------------------------------------------------------------

    self.serialCon=self.self.serialCon
    nsamples=self.self.nsamples.value()
    itime=self.self.itime.value()
    interval=self.self.interval.value()

    #------------------------------------------------------------------------
    #-- NetCDF variables initialization if SamplexFile and ToSave has '0'
    #-- value, data has no to be saved (NetCDF option is not checked).
    #-- Otherwise, SamplexFile and ToSave have values configured by the user
    #-- on GUI
    #------------------------------------------------------------------------

    SamplesxFile=0
    ToSave=0

    if (self.self.SaveCheck.isChecked()==1): #NetCDF File option is selected on GUI
        SamplesxFile=self.self.SamplesxFile.value()
        ToSave=self.self.ToSave.value()

    #------------------------------------------------------------------------
    #-- Some variables initialization to right Acquiring operation
    #------------------------------------------------------------------------

    self.data=[]#data vector to plot initialization
    end=0 #When this value is modified when acquisition ends

    #------------------------------------------------------------------------
    #-- An 'Acquisition' packet '*AD' is send to PDA with some acquisition
    #-- parameters
    #------------------------------------------------------------------------

    self.tosend=pack('c2sHiHHH','*','AD',nsamples,itime,interval,SamplesxFile,ToSave)
    self.tosend+=pack('H',crc16(self.tosend))
    self.serialCon.write(self.tosend)

    #print "An AD packet has been sent"  #DEBUG

    while end==0:#while end==0 continue waiting packets from PDA

    #------------------------------------------------------------------------
    #-- Some GUI elements are blocked during acquisition process
    #------------------------------------------------------------------------

        self.self.itime.setEnabled(0)
        self.self.nsamples.setEnabled(0)
        self.self.interval.setEnabled(0)

    #------------------------------------------------------------------------
```

```
#-- Waiting for an '*IN' packet (1 sampler)
#-- Waiting header ('*'), at the moment
#-------------------------------------------------------------------------

    TWaitingHeader=WaitingHeader(15.0,self.serialCon)
    #timeout has to be optimized
    TWaitingHeader.start()
    TWaitingHeader.join()

#-------------------------------------------------------------------------
#-- Execution continue if WaitingHeader thread has finished
#-- If Header found or timeout occurred has to be checked
#-------------------------------------------------------------------------
    if TWaitingHeader.Exit.isSet():

        #print "To much time waiting" #DEBUG
        #print "Provably AD packet or IN packet has been lost" #DEBUG

        pass

        #-----------------------------------------------------------------
        #-- Non retransmission available and this 'error' not catched
        #-- (TO DO)*
        #-----------------------------------------------------------------

    elif TWaitingHeader.Found.isSet():

        #print "Header Found" #DEBUG

        #-----------------------------------------------------------------
        #-- Two bytes next header ('*') indicate packet type ('packettype')
        #-- Only IN and EX type valid in this version
        #-----------------------------------------------------------------

        packettype=self.serialCon.read(2)

        if packettype=='EX':

        #print "Stop packet received" #DEBUG

            end=1

        #-----------------------------------------------------------------
        #-- Modifying end value main Acquiring loop ends
        #-- (TO DO)*
        #-----------------------------------------------------------------

        if packettype=='IN':

            #Info (sample) packet received

            TWaitingPacket=WaitingPacket(15.0,self.serialCon,243)
            TWaitingPacket.start()
            TWaitingPacket.join()

            #-------------------------------------------------------------
            #-- TWaitingPacket wait until 243 (info packet length) are
            #-- available on serial port
            #--
            #-- 'timeout' value should be optimized and number of bytes to
            #-- wait more generic variable (TO DO)*
            #-------------------------------------------------------------

            if TWaitingPacket.Exit.isSet():

                #print "No packet available in before timeout time" #DEBUG

                #---------------------------------------------------------
                #-- Non retransmission available and this 'error' not catched
                #-- (TO DO)*
                #---------------------------------------------------------

                pass
```

```
        elif TWaitingHeader.Found.isSet():

            #print "Possible valid IN packet received" #DEBUG

            packet='*IN'+self.serialCon.read(243)
            #-------------------------------------------------------------
            #-- Number of bytes to read should be in used in more generic
            #-- variable (TO DO)*
            #-------------------------------------------------------------

            if checkcrc16(packet)==1:
                #---------------------------------------------------------
                #-- CRC value is checked and NON errors found
                #---------------------------------------------------------

                received=unpack('c2s241BH',packet)
                idSample=received[2]
                tosend='*AK'+chr(idSample)
                tosend+=pack('H',crc16(tosend))
                self.serialCon.write(tosend)
                #print "AK sent" #DEBUG

                self.info=received[3:243]
                #---------------------------------------------------------
                #-- Sample values length should be used in more generic
                #-- variable (TO DO)*
                #---------------------------------------------------------

                for x in range(len(self.info)):
                    self.data.append(x)
                    self.data.append(self.info[x])

                #---------------------------------------------------------
                #-- Sample information is unpacked and stored on self.data
                #-- in a format able to be printed with Painting thread
                #--
                #-- self.data -- what to "draw"
                #-- self.self.graph -- where to "draw" it
                #---------------------------------------------------------

                #---------------------------------------------------------
                #-- idSample is a random identifier to be sure that right
                #-- packet is confirmed to PDA
                #---------------------------------------------------------

                Paint=Painting(self.data,self.self.graph)
                Paint.start()
                #Paint.join() do nothing

                #---------------------------------------------------------
                #-- Not all samples are printed on PDA some problems with
                #-- GUI the focus (TO DO)*
                #---------------------------------------------------------

                self.data=[]#initialization

            else:

                #---------------------------------------------------------
                #-- CRC value is checked and errors found
                #---------------------------------------------------------

                #print "IN packet received with errors" #DEBUG

                #---------------------------------------------------------
                #-- Non retransmission available and this 'error' not
                #-- catched (TO DO)*
                #---------------------------------------------------------

                pass

    else:
```

```
                    #print "Packet without valid type received" #DEBUG

                    pass

                    #-------------------------------------------------------
                    #-- This 'error' not catched (TO DO)*
                    #-------------------------------------------------------

        #-----------------------------------------------------------------------
        #-- GUI elements blocked can be unblocked because acquisition process
        #-- ended
        #-----------------------------------------------------------------------

        self.self.itime.setEnabled(1)
        self.self.nsamples.setEnabled(1)
        self.self.interval.setEnabled(1)

        #-----------------------------------------------------------------------
        #-- This class should be improved using 'try' statement structure
        #-- (TO DO)*
        #-----------------------------------------------------------------------

class Painting(threading.Thread):

    def __init__(self,data,where):
        """
        -----------------------------------------------------------------------
        Painting(data,where) -- INIT

        DESCRIPTION:

        Draw data on where

        ARGUMENTS:

        data -- data to "draw"
        where -- where to "draw" data (QWidget object)

        * DATA NEEDED FORMAT:

        data=[x0,y0,x1,y1,...]= [0, pixel1value , 1, pivel2value,...]

        Has start() method
        -----------------------------------------------------------------------
        """
        threading.Thread.__init__(self)
        self.data=data
        self.where=where

    def run(self):
        """
        -----------------------------------------------------------------------
        Painting -- start() METHOD

        DESCRIPTION:

        Draw received data on a QWidget object from GUI
        -----------------------------------------------------------------------
        """

        points=QPointArray(self.data)
        self.p=QPainter(self.where)
        self.p.setPen(QColor(0,0,0))
        self.where.repaint()
        self.p.drawPolyline(points)

#---------------------------------------------------------------------------
#       ###     GUI CLASS       ###
#---------------------------------------------------------------------------

class PdaApp(QMainWindow):
    def __init__(self,*args):
```

```
"""
------------------------------------------------------------------------
INIT CLASS MODULE

Interface objects and some global variables definition.

------------------------------------------------------------------------
"""
#------------------------------------------------------------------------
#       ###     MAIN WINDOW     ###
#------------------------------------------------------------------------

apply(QMainWindow.__init__,(self,)+args)#PDA screen has 240x300 pixels

#------------------------------------------------------------------------
#           USER INTERACTION SUBWINDOW
#------------------------------------------------------------------------
self.setGeometry(0,20,240,300)

self.interaction=QTabWidget(self)
self.interaction.setGeometry(0,0,240,180)

#------------------------------------------------------------------------
#           Interaction subwindow ACQUISITION Tab
#------------------------------------------------------------------------

self.AdTab=QWidget(self)

self.label=QLabel('IntegrationTime(us):',self.AdTab)
self.label.setGeometry(10,10,132,20)
self.itime=QSpinBox(10,15000000,10,self.AdTab)
self.itime.setGeometry(155,10,70,20)
self.itime.setValue(10000)

self.label=QLabel('Number of Samples:',self.AdTab)
self.label.setGeometry(10,35,125,20)
self.nsamples=QSpinBox(0,20000,1,self.AdTab)
self.nsamples.setGeometry(155,35,70,20)
self.nsamples.setValue(1)

self.label=QLabel('Samples Interval(s):',self.AdTab)
self.label.setGeometry(10,60,132,20)
self.interval=QSpinBox(1,20000,1,self.AdTab)
self.interval.setGeometry(155,60,70,20)
self.interval.setValue(2)

self.AdButton=QPushButton("Start",self.AdTab)
self.AdButton.setGeometry(45,100,70,20)
self.connect(self.AdButton,SIGNAL("clicked()"),self.GetSample)

self.StopButton=QPushButton("Stop",self.AdTab)
self.StopButton.setGeometry(115,100,70,20)
self.connect(self.StopButton,SIGNAL("clicked()"),self.Stopped)

#self.label=QLabel('Maximum value:',self.AdTab)
#self.label.setGeometry(50,130,150,20)
#self.maxvalue=QLabel('0',self.AdTab) #Maxvalue does not work
#self.maxvalue.setGeometry(160,130,80,20)

#maxvalue does not work. This value is not sent by PC (TO DO)*

self.interaction.addTab(self.AdTab,"&ACQUISITION")

#------------------------------------------------------------------------
#           Interaction subwindow SAVE FILE Tab
#------------------------------------------------------------------------

self.FileTab=QWidget(self)

self.SaveCheck=QCheckBox("Save data (NetCDF format)",self.FileTab)
self.SaveCheck.setGeometry(20,20,190,20)
self.connect(self.SaveCheck,SIGNAL("clicked()"),self.FileOptions)
```

```
        self.label=QLabel('Samples per file:',self.FileTab)
        self.label.setGeometry(20,70,100,20)
        self.SamplesxFile=QSpinBox(1,30000,1,self.FileTab)
        self.SamplesxFile.setGeometry(130,70,70,20)
        self.SamplesxFile.setValue(10)
        self.SamplesxFile.setEnabled(0)

        self.Label1=QLabel('Backup time:',self.FileTab)
        self.Label1.setGeometry(20,100,100,20)
        self.ToSave=QSpinBox(1,30000,1,self.FileTab)
        self.ToSave.setGeometry(130,100,70,20)
        self.ToSave.setValue(10)
        self.ToSave.setEnabled(0)
        #ToSave: number of samples, at the moment.
        #A variable with time units will be better (TO DO)*

        self.interaction.addTab(self.FileTab,"&SAVE")

        #-------------------------------------------------------------------------
        #           GRAPH SUBWINDOW
        #-------------------------------------------------------------------------

        self.graph=QWidget(self)
        self.graph.setGeometry(0,180,240,100)

        #-------------------------------------------------------------------------
        #       ###     GLOBAL VARIABLES        ###
        #-------------------------------------------------------------------------

        self.serialCon=serial.Serial('/dev/ttySA0',57600)#DEBUG ON PDA
        #self.serialCon=serial.Serial('/dev/ttyS0',57600) #DEBUG ON PC
        #connection speed and port by default

        self.data=[]

        #-------------------------------------------------------------------------
        #       ###     METHODS CALL BY GUI EVENTS      ###
        #-------------------------------------------------------------------------


    def FileOptions(self,event=None):
        """
        -------------------------------------------------------------------------
        FILE OPTIONS METHOD

        Manage NetCDF saving files options: values only can be modified if option
        "Save Files" on GUI is selected.
        -------------------------------------------------------------------------
        """

        if (self.SaveCheck.isChecked()==1): #Selected
            #Enable Saving Files configurable values
            self.ToSave.setEnabled(1)
            self.SamplesxFile.setEnabled(1)
        else:#Unselected
            #Disable Saving Files configurable values
            self.ToSave.setEnabled(0)
            self.SamplesxFile.setEnabled(0)

    def GetSample(self,event=None):
        """
        -------------------------------------------------------------------------
        GET SAMPLE METHOD

        This function is call by GUI 'Start' Button and manages Acquiring process
        using threads (managed by timers) that is managed by PC at the same time.
        This process finishes using Stop Button or by itself when configured number
        of samples has been obtained (in this to cases PC send and exit string
        '*EX'to PDA).
        -------------------------------------------------------------------------
        """
        Adcuire=Acquiring(self)
        Acquire.start()
```

```
    def Stopped(self):
        """
        ------------------------------------------------------------------------
        STOP SAMPLING METHOD

        This function is call by GUI 'Stop' Button and stops Acquiring process by
        sending and exit string '*EX' to PC which returns it to PDA and stops the
        process.

        This method does not work at the moment,S4000AppPC.py software has to be
        modified (TO DO)*
        ------------------------------------------------------------------------
        """
        #print "Stop Button has been pressed" #DEBUG

        self.tosend='*EX'
        self.serialCon.write(self.tosend)

def main(args):
    app=QApplication(args)
    win=PdaApp()
    win.show()
    app.connect(app,SIGNAL("lastWindowClosed()"),app,SLOT("quit()"))
    app.exec_loop()

if __name__=="__main__":
    main(sys.argv)
```

A.3. Llibreria OOS4000v2.py

```
"""
-----------------------------------------------------------------------------------------
    (C)2008 Núria Pujol Vilanova <nuriapujolvilanova@gmail.com>

    FILE NAME: OOS4000v2.py

    DESCRIPTION:
        USB comunication module for USB4000 Device (Ocean Optics) in Python.

                - OpenDevice
                - ResetDevice
                - SentIntTime
                - GetSample
                - MaxSample
                - GetReducedSample

    OS4000.py file improvement according new scenario (S4000AppPc.py)

    GPL License
-----------------------------------------------------------------------------------------
"""
import sys, os,usb
from struct import pack,unpack
from numpy import*
from time import time
from pylab import plot,show

def OpenDevice(id=0x2457): #id=Hex identifier
    """
    -------------------------------------------------------------------------------------
    OpenDevice(id=0x2457)

    DESCRIPTION:

    Return a HandleDevice varible related with a USB device with the same 'id' identifier.
    (HandleDevice Object is the only able to write and read in USB communication)
```

```
    ARGUMENTS:

    id -- Hexadecimal idVendor identifier (default =0x2457)
    ----------------------------------------------------------------------------------------
    """
    buses = usb.busses()
    for bus in buses :
        for device in bus.devices :
            if device.idVendor == id:
                print "USB4000 Device Found"
                dev=device

            elif device.idVendor==0:
                pass
            else:
                print "Other type of Device Found"

    handle= dev.open()
    return handle

def ResetDevice(handle):#handle device
    """
    ----------------------------------------------------------------------------------------
    ResetDevice(handle)

    DESCRIPTION:

    Resets USB communication. Non output.

    ARGUMENTS:

    handle -- HandleDavice Object related to communication to reset. Object returned by
    OpenDevice function.
    ----------------------------------------------------------------------------------------
    """

    handle.reset()

def SetIntTime(handle,BufTx):
    """
    ----------------------------------------------------------------------------------------
    SetIntTime(handle,buf_tx)

    DESCRIPTION:

    Write new integration time value on the device. Non output.

    ARGUMENTS:

    handle -- HandleDavice Object related to device to set new integration time.
    Object returned by OpenDevice

    buf_tx --  New integration time in vendor described format.

    *buf_tx= chr(0x02) + struct.pack('i', IntegrationTimeValue)
    buf_tx obtained from S4000AppPc.py execution

    ----------------------------------------------------------------------------------------
    """

    o=handle.bulkWrite(0x01,BufTx)

def GetSample(BufTx,itime,id=0x2457):

    """
    ----------------------------------------------------------------------------------------
    GetSample(buf_tx,itime,id=0x257)

    DESCRIPTION:

    Return a single acquisition in a vector (3840 values)
```

```
    ARGUMENTS:

    buf_tx -- New integration time in vendor described format.
    itime -- Integration time in INT format
    id -- Hexadecimal idVendor identifier (default =0x2457)

    *buf_tx= chr(0x02) + struct.pack('i', IntegrationTimeValue)
    buf_tx obtained from S4000AppPc.py execution

    ---------------------------------------------------------------------------------------------
    """
    #-------------------------------------------------------------------------------------------
    #-- 'itime' is only used to calculate time that USB buffer have
    #-- to wait.
    #-- Provably, only 'buf_tx' will be enough, buf_tx can be easily
    #-- translated to itime (TO DO)*
    #-------------------------------------------------------------------------------------------

    handle=OpenDevice(id)
    #ResetDevice(handle)
    SetIntTime(handle,BufTx)

    buffer=chr(0x09)
    o=handle.bulkWrite(0x01,buffer)

    SampleTime=int(time())

    #-------------------------------------------------------------------------------------------
    #-- 'sampletime' is when sample is 'taken' in seconds (UTC format).
    #-- This value has to be stored in netCDFFile if it is necessary
    #-------------------------------------------------------------------------------------------

    b0=handle.bulkRead(0x86,2048,itime/7)
    b4=handle.bulkRead(0x82,5632,itime/11)
    sincro=handle.bulkRead(0x82,1)
    #'sincro' not valid data but has to be read, if not it remains in buffer

    data=b0+b4
    #'data' is valid data read from USB4000

    #-------------------------------------------------------------------------------------------
    #-- Every spectrometric data (for every photodiode) has 2 bytes and
    #-- become from USB4000 in this format:
    #--
    #-- [LSB photodiode1,MSB phot1,LSB phot2,MSB phot2,...]
    #--
    #-- 'SpecSignature' contain LSB and MSB combined getting values
    #-- from '0' (ideally, always there is some noise) to 32000 that can be
    #-- represented graphically
    #-------------------------------------------------------------------------------------------

    data=[x for x in data]

    dataLSB=[((256+data[2*i])&0xFF) for i in range(len(data)/2)]
    dataMSB=[((256+data[2*i+1])&0xFF)<<8 for i in range(len(data)/2)]

    SpecSignature=[dataLSB[i]^dataMSB[i] for i in range(len(dataLSB))]

    return SpecSignature,SampleTime

    #-------------------------------------------------------------------------------------------
    #--'SpecSignature can be represented on Pc screen but not in PDA
    #-- (there are more points (values) than screen pixels.
    #-- For this reasons we have to decrease the number of SpecSignature
    #-- values (we loose resolution) but we can still recognize signal
    #-- obtained.
    #-- Entire SpecSignature will be saved on a database in netCDF format
    #-- (if this option is selected on PDA) and this reduced sample sent
    #-- to PDA to be represented
    #-------------------------------------------------------------------------------------------

def MaxSample(SpecSignature):
    """
```

```
    ---------------------------------------------------------------------------------------
    MaxSample(SpecSignature)

    DESCRIPTION:

    Return maximum value of a sample

    ARGUMENTS:

    SpecSignature -- One single sample values vector

    ----
    """
    length =len(SpecSignature)
    maximum=max(SpecSignature[2:length])

    return maximum

def GetReducedSample(SpecSignature,screenV=120,screenH=240):
    """
    ---------------------------------------------------------------------------------------
    GetReducedSample(SpecSignature,screenV=100,screenH=240)

    DESCRIPTION:

    From entire sample obtained from USB4000 (SpecSignature) obtain a
    'reduced'  sample able to be 'drawn' on PDA screen size

    ARGUMENTS:

    SpecSignature -- One single sample values vector
    screenV -- number of able PDA screen pixels on vertical
    screenH -- number of able PDA screen pixels on horizontal
    ----
    """
    resolution=int(len(SpecSignature)/screenH)
    short_signature=[]
    maximum=MaxSample(SpecSignature)

    for x in range(screenH):
        short_signature.append(screenV-((SpecSignature[x*resolution]*screenV)/maximum))
        #-----------------------------------------------------------------------------------------
        #-- Samples values adjustments according to screen size
        #-- (screenH x screenV in pixels)
        #-----------------------------------------------------------------------------------------

    return short_signature

def main():
    itime=300000
    buf_tx=chr(0x02)+pack('i',itime)
    [lectura,time]=GetSample(buf_tx,itime)

    plot(lectura)
    show()

if __name__ == '__main__':
    main()
```

A.4. Llibreria SaveNetCDF.py

```
"""
---------------------------------------------------------------------------------------
    (C)2008 Núria Pujol Vilanova <nuriapujolvilanova@gmail.com>

    FILE NAME: SaveNetCDF.py

    DESCRIPTION:
```

```
    DataFile class definition and its methods related with NetCDF files
    saving.
-----------------------------------------------------------------------------------------
"""
from Scientific.IO.NetCDF import*
from time import time,strftime,localtime
from Numeric import*
from numpy import array

class DataFile:

    def __init__(self,ToSave,SamplesxFile):
        """
        ------------------------------------------------------------------------
        DataFile(ToSave,SamplesxFile) -- INIT

        DESCRIPTION:

        DataFile class manages all saving files processes using its
        methods.

        ARGUMENTS:

        ToSave -- Back up number of samples. Samples are saved into
        NetCDF file by blocks of this length.
        SamplesxFile -- Maximum number of samples per file.

        This values are used to initialize variables to store
        spectrometric data while waits to be saved.

        Has SaveFile() and SaveEnd()
        ------------------------------------------------------------------------
        """

        self.ToSave=ToSave
        self.SamplesxFile=SamplesxFile
        self.STime=zeros(self.ToSave,Int)
        self.SBuffer=zeros([self.ToSave,3840],Int)

    def SaveFile(self,acquired,specsignature,gtime):

        """
        ------------------------------------------------------------------------
        SaveFile(acquired,specsignature,gtime) -- DataFile method

        DESCRIPTION:

        This method save data into NetCDf file while acquisition process is active
        according to DataFile class initialization values.

        ARGUMENTS:

        acquired -- Number of samples acquired at the moment to
        invoke this method.
        specsignature -- Spectrometric data vector from one
        acquisition.
        gtime -- 'specsignature' acquisition time reference.

        This values are related with a single sample, for this
        reasons this method has to be invoked next every
        acquisition.

        ------------------------------------------------------------------------
        """

        self.SBuffer[acquired%self.ToSave]=specsignature
        self.STime[acquired%self.ToSave]=gtime

        if(acquired%self.SamplesxFile==0):#first File element
        #-----------------------------------------------------------------
        #-- NetCDF first sample. New NetCDF File is created
        #-----------------------------------------------------------------
```

```
    print 'New NetCDF' #DEBUG
    reftime=localtime(gtime)
    filename='SP'+strftime('%Y',reftime)+strftime('%m',reftime)+strftime('%d',reftime)+
    'T'+strftime('%H',reftime)+strftime('%M',reftime)+strftime('%S',reftime)+'.nc'
    self.ncfile=NetCDFFile(filename,'w')
    self.ncfile.createDimension('lonsample',3840)
    self.ncfile.createDimension('time',None)
    self.reltime=self.ncfile.createVariable('reltime',Int,('time',))
    self.reltime.unit='s'
    self.specdata=self.ncfile.createVariable('specdata',Int,('time','lonsample'))
    self.specdata.unit='None'

    #---------------------------------------------------------------
    #-- GLOBAL VARIABLES
    #---------------------------------------------------------------
    self.ncfile.title= 'Spectrometric data-' + filename
    self.ncfile.institution_='CMIMA-CSIC'
    self.ncfile.source_='Spectrometric data obtained from USB4000-N1'
    self.ncfile.history='Not processed'
    self.ncfile.comment_='3840 values per sample obtained but wavelength non calibrated'
    self.ncfile.conventions="CF-1.0"

if((acquired+1)%self.ToSave==0)or((acquired+1)%self.SamplesxFile==0):
#---------------------------------------------------------------
#-- Data has to be saved on NetCDF.
#-- Every 'ToSave' samples or at NetCDF File End (SamplesxFile
#-- multiple)
#---------------------------------------------------------------

    if((acquired+1)%self.SamplesxFile==0):

    #-----------------------------------------------------------
    #-- End of File
    #-----------------------------------------------------------

        nsave=((acquired+1)%self.ToSave)

        if nsave == 0:
            nsave=self.ToSave

        self.specdata[self.SamplesxFile-nsave:self.SamplesxFile]=self.SBuffer[0:nsave]
        self.reltime[self.SamplesxFile-nsave:self.SamplesxFile]=self.STime[0:nsave]

        print "Reltime:" #DEBUG
        print  self.STime[0:nsave] #DEBUG

        self.ncfile.sync()
        self.ncfile.close()

        print "End of File" #DEBUG
        print nsave
        print "samples saved"

    else:

    #-----------------------------------------------------------
    #-- 'ToSave' multiple
    #-----------------------------------------------------------

        nsave=(acquired+1)%self.SamplesxFile

        if nsave>self.ToSave:
            nsave=self.ToSave

        TO=(acquired+1)%self.SamplesxFile
        FROM=TO-nsave

        self.specdata[FROM:TO]=self.SBuffer[self.ToSave-nsave:self.ToSave]
        self.reltime[FROM:TO]=self.STime[self.ToSave-nsave:self.ToSave]

        self.ncfile.sync()
```

```
    def SaveEnd(self,acquired):
        """
        ------------------------------------------------------------------------
        SaveEnd(acquired) -- DataFile method

        DESCRIPTION:

        This method save data into NetCDf file when acquisition process is stopped
        and save data that remains into application auxiliary variables.

        ARGUMENTS:

        acquired -- Number of samples acquired at the moment to
        invoke this method.

        ------------------------------------------------------------------------
        """

        if ((acquired)%self.ToSave!=0)and((acquired)%self.SamplesxFile!=0):

        #----------------------------------------------------------------------
        #-- Acquiring process has to been stopped and there some samples to save
        #----------------------------------------------------------------------

            nsave=((acquired)%self.ToSave)

            TO=(acquired%self.SamplesxFile)
            FROM=TO-nsave

            self.specdata[FROM:TO]=self.SBuffer[0:nsave]
            self.reltime[FROM:TO]=self.STime[0:nsave]

            self.ncfile.sync()
            self.ncfile.close()
```

A.5. Llibreria CRC.py

```
"""
--------------------------------------------------------------------------------------------
    (C)2008 Núria Pujol Vilanova <nuriapujolvilanova@gmail.com>

    FILE NAME: CRC.py

    DESCRIPTION:

    CRC Functions

    This module has two functions:

        crc16(s) that calculate CRC-16 value from 's' string

        checkcrc16(data) that calculate if crc16 value from
        'data' string is correct

    Based on CRC-16 function by Nathan Wallace, Jul 5th 2000
    http://www.faqts.com/knowledge_base/view.phtml/aid/4342

--------------------------------------------------------------------------------------------
"""

from struct import unpack

def crc16(s):
    """
    ----------------------------------------------------------------------------------------
    crc16(s)

    DESCRIPTION:
```

```
    Calculate CRC-16 value from 's' string

    ARGUMENTS:

    s -- string to calculate CRC-16 from (of any length)

    RETURNED:

    crc -- CRC-16 value

    --------------------------------------------------------------------------------------
    """
    crc = 0

    for index1 in range(len(s)):
        crc = crc ^ (ord(s[index1]) << 8)

        for index2 in range(1, 9):
            if crc & 0x8000 != 0:
                crc = ((crc << 1) ^ 0x1021)#0x1021 commonly used polynomial (10001000000100001)
                crc=crc &0xFFFF
            else:
                crc = crc << 1
                crc=crc&0xFFFF

    return crc

def checkcrc16(data):# TWO last data bytes corresponds wit CRC bytes
    """
    --------------------------------------------------------------------------------------
    crc(s)

    DESCRIPTION:

    Check if CRC-16 value of 'data' string is correct.

    ARGUMENTS:

    data -- string to check CRC-16 from it (any length)

    RETURNED:

    0 -- CRC-16 incorrect value
    1 -- CRC-16 correct value
    --------------------------------------------------------------------------------------
    """

    #--------------------------------------------------------------------------------------
    #-- 2 last bytes from 'data' are crc value
    #--------------------------------------------------------------------------------------

    s=data[0:len(data)-2]#String format
    crc16R=unpack('H',data[len(data)-2:len(data)])[0]#String format

    if (crc16R == crc16(s)):
        return 1
    else:
        return 0
```