Treball Fi de Carrera

Enginyeria Tècnica de Telecomunicació

Especialitat en Sistemes Electrònics

Implementació de codificació distribuïda de font en una xarxa de sensors real

Xavier Colomer Núñez

Director: Antoni Morell Pérez

Departament de Telecomunicació i Enginyeria de Sistemes

Co-Director: Joan Enric Barceló i Lladó

Departament de Telecomunicació i Enginyeria de Sistemes

Escola Tècnica Superior d'Enginyeria (ETSE)

Universitat Autònoma de Barcelona (UAB)

Juliol 2011

UAB

El tribunal d'avaluació d'aquest Treball Fi de Carrera, reunit el dia 8 de Juliol de 2011, ha acordat concedir la següent qualificació:

President:

Vocal:

Secretari:

El sotasignant, *Antoni Morell Pérez*, Professor de l'Escola Tècnica Superior d'Enginyeria (ETSE) de la Universitat Autònoma de Barcelona (UAB),

CERTIFICA:

Que el treball presentat en aquesta memòria sobre Implementació de codificació distribuïda de font en una xarxa de sensors real ha estat realitzat sota la seva direcció per l'alumne *Xavier Colomer Núñez*.

I, perquè consti a tots els efectes, signa el present certificat.

Bellaterra, a 30 de Maig de 2011

Signatura: *Antoni Morell Pérez*

Índex

Índex de Figures

Índex de Taules

1. Prefaci

1.1. Definició dels objectius

L'objectiu principal d'aquest estudi és l'estalvi d'energia en sensors sense fils mitjançant un algoritme anomenat *Distributed Source Coding* el qual es basa en l'estudi de la correlació que hi ha entre els diferents sensors de la xarxa per a extreure'n la relació entre ells. D'aquesta manera es pot calcular un valor mínim de bits a enviar, amb el qual es pot codificar la mostra enviada escurçant-la aquest nombre de bits i que un cop al receptor es podrà descodificar sense problema. Aquest valor de bits a enviar s'anomena *ibit*.

Per assolir aquest objectiu s'hauran de programar els sensors IRIS XM2110 i crear un programa en Matlab® que sigui l'enllaç amb el *PC* i faci la codificació anomenada per, finalment, monitoritzar la xarxa i comprovar l'error afegit i el nivell de compressió aconseguit per l'algoritme.

1.2. Estructura de la memòria

La memòria consta d'una introducció la qual posa en el context de les WSN (*Wireless Sensor Networks*) i comenta l'estat del art d'aquestes. En la part final d'aquesta introducció es defineix el problema sobre l'estalvi d'energia dels sensors.

Seguidament es presenten l'escenari que es crearà, el hardware utilitzat, és a dir, els models dels sensors utilitzats, i el software per a programar l'algoritme. Després d'aquesta presentació es fa una petita explicació sobre la programació dels sensors i d'alguna funció important de Matlab® que serveix per comprendre millors els apartats següents.

Després es fa una explicació sobre *Distributed Source Coding* on es veu el seu funcionament, s'introdueix una part important del algoritme anomenada filtre de Wiener i per últim es mostren els passos teòrics a seguir i el programa utilitzat en Matlab® per al càlcul del *ibit*.

Més endavant es presenta mitjançant diagrames de blocs els algoritmes que implementa cadascuna de les parts del sistema: *PC, Fusion Center* i *Sensor Nodes.* Un

cop comentats els diagrames de blocs s'explica el funcionament més extensament de les parts que formen el programa, posant els codis i descrivint-los.

En l'apartat següent es mostren els resultats un cop implementat l'algoritme. Aquest és l'apartat més important de la memòria on s'analitzen les millores obtingudes i es veu, mitjançant diverses gràfiques, la monitorització de diferents escenaris de la xarxa de sensors, es compara l'estimació utilitzada amb un altre de proposada i l'estudi de la compressió que assoleixen les mostres.

Finalment s'exposen les conclusions del treball, i es mostren les línies futures del projecte.

1.3. Agraïments

Primerament m'agradaria agrair a la família, als amics i a la xicota tot el suport mostrat durant aquests mesos, sense ells encara hi seria.

En segon lloc l'ajuda que m'han brindat els usuaris del servei d'ajuda de TinyOS (TinyOS-help) i el webmaster *khardell* del fòrum www.sensoresinalambricos.es, una web altament recomanable per els que s'iniciïn en aquest sistema operatiu.

També agrair als meus companys de classe els consells rebuts i la paciència quan m'he retrassat en fer alguna feina a causa d'aquest projecte i a la gent del departament de comunicacions que algun cop m'ha solucionat algun dubte amb els sensors.

Finalment i no per això menys important, també voldria mostrar el meu agraïment als meus tutors del projecte. A Antoni Morell Pérez, per la confiança en mi dipositada i al Joan Enric Barceló Lladó per tota l'ajuda que m'ha proporcionat encara que això l'hagi fet plegar més tard.

Gràcies a tots.

2. Introducció

2.1. Context

L'evolució de la tecnologia és cada cop més ràpida i sorprenent. No ho és menys en el món de les comunicacions on en l'últim segle, i més pronunciadament en les últimes dècades s'ha tornat un món ple de possibilitats. El telèfon, la ràdio, la televisió i ara internet han aconseguit un nivell de coneixement del món en temps real impossible d'imaginar pels nostres avantpassats.

Gràcies a la radiofreqüència, la comunicació per satèl·lit i d'altres tipus de comunicació sense fils s'han pogut superar barreres abans inexpugnables i aconseguir un abast gairebé total de comunicació al planeta.

El següent pas és el control de les variables que ens rodegen. Temperatura, humitat, llum, radiació, etc. són paràmetres que monitoritzats proporcionen una informació molt valuosa per al control del clima, de la contaminació, d'incendis o fins i tot de la salut.

Això es pot aconseguir mitjançant xarxes de sensors sense fils (*Wireless Sensor Networks* WSN), les quals, mitjançant un mòdul autònom amb un sensor, són capaces de mesurar el paràmetre a determinar i enviar tal mesura mitjançant radiofreqüència, a una estació de processat per així tenir un control en temps real d'aquesta dada.

Aquesta idea té un potencial molt gran i ja s'està investigant en moltes universitats d'arreu del món.

2.2. Estat de l'Art

Ja s'ha comentat el potencial que tenen les xarxes de sensors sense fils. Tot seguit es demostrarà, mitjançant exemples d'utilització de dites xarxes, que aquesta afirmació és certa.

Aplicacions agrícoles

Les WSN permeten controlar el grau d'humitat i temperatura d'una zona per tal de saber quan ha de ser regada o no, si s'acosta una gelada o algun efecte climàtic, si és possible el sorgiment d'una plaga, etc.

Figura 1. Esquema d'una xarxa de sensors en un hivernacle [4]

Els beneficis d'aquestes dades són l'estalvi d'aigua i pesticida , ja que en el segon cas es pot detectar la zona afectada i alliberar només en aquell espai el repel·lent.

Aplicacions sanitàries

Mitjançant sensors que monitoritzin les constants vitals, els nivells de segons quins components a la sang, etc. es pot dur un control exhaustiu de l'estat físic d'un pacient o una persona gran sense necessitat d'atacar la seva privacitat i millorant la seva qualitat de vida.

Un projecte que s'ha investigat a la Universitat de Harvard i que té a veure amb aquesta aplicació és el *CodeBlue* (Figura 2) el qual monitoritza la tassa de batecs del cor, el nivell d'oxigen a la sang, etc.

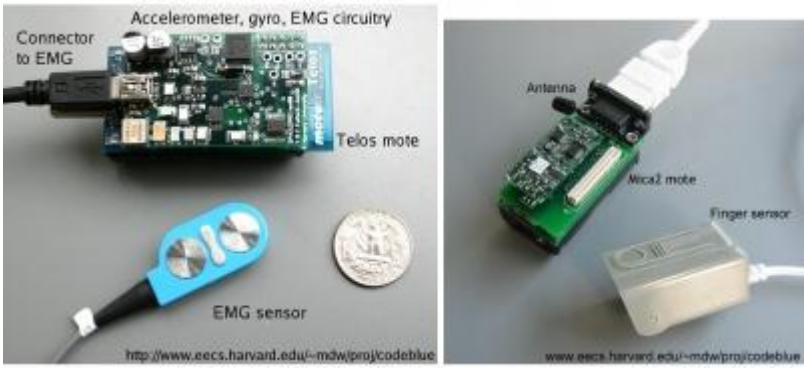


Figura 2. Sensors CodeBlue per fer electromiogrames (EMG) i controlar els batecs del cor [5]

Aplicacions a la natura

El principal objectiu en aquest cas és la prevenció de possibles incendis. Una WSN situada en un bosc monitoritzarà paràmetres, com la temperatura o la humitat, que indicaran en temps real la possibilitat d'un incendi en aquella zona.

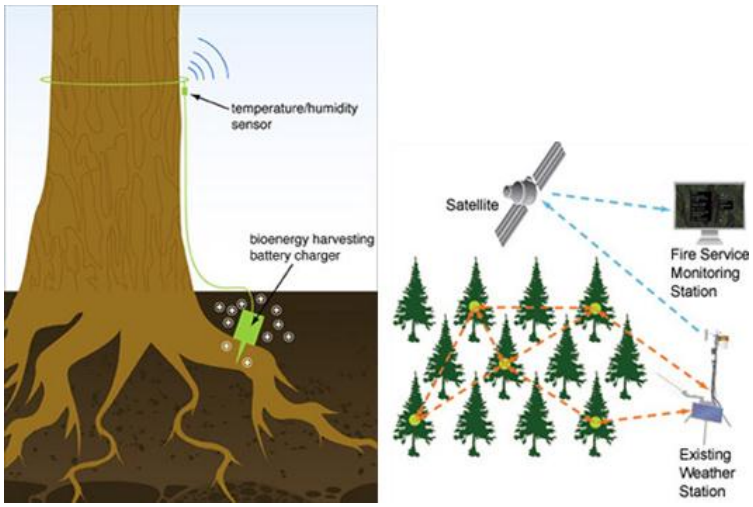


Figura 3. Prototip de sensor d'humitat alimentat per bioenergia [6]

Però hi ha altres possibilitats a mencionar, com el control d'espècies protegides en parcs naturals, on moltes vegades l'accés és complicat i s'intenta minimitzar l'actuació de personal i vehicles per la preservació de la natura. Aquesta xarxa pot donar un control molt extens del cabal de rius, l'estat del creixement de la vegetació, comptar individus d'alguna espècie animal, etc.

Aplicacions civils i militars

Aquestes aplicacions són nombroses i de moltes formes: tenir un control exhaustiu del trànsit, monitoritzar el nivell de contaminació, crear un mapa en temps real dels pàrquings lliures en les ciutats, vigilància dels edificis, etc.

A les militars per altra banda, les WSN permeten un seguiment de tropes molt acurat i la detecció d'armes químiques o biològiques, així com millors sistemes de vigilància davant l'atac enemic.

Figura 4. WSN integrada a la ciutat de Cambridge [7]

El terme “Ciutat Sensorial” ja es comença a sentir, i és una prova més que les xarxes de sensors sense fils cada vegada tenen més importància.

2.3. Definició del problema

El funcionament de les WSN té una similitud amb els eixams d’insectes fent una comunicació sincronitzada entre els mateixos per aconseguir una mesura intel·ligent i un mínim consum amb el màxim d’extensió. És a dir, es procura mesurar només el necessari per així incrementar l’estalvi d’energia.

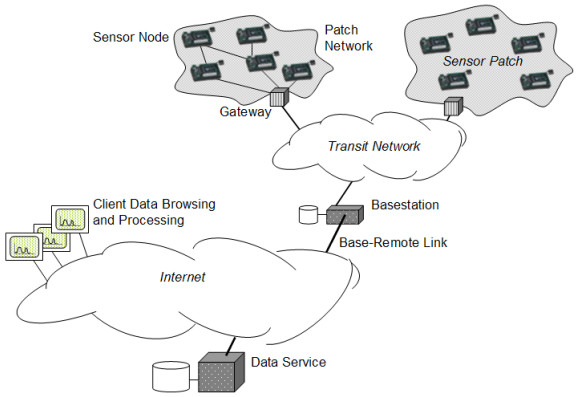


Figura 5. Estructura d'una WSN **[8]**

Mitjançant la comunicació AdHoc, una comunicació basada en la transmissió d’informació des dels sensors més llunyans a la base de processat a través dels sensors que es troben més a prop, s’aconsegueix un abast molt gran, però també un consum energètic elevat ja que cada sensor enviarà la seva informació més la dels sensors llunyans, fent que el consum sigui inversament proporcional a la distància a la que està el sensor de la base.

Ja que els sensors són autònoms, si la font d’alimentació s’extingeix, s’haurà de canviar la font, i si aquest procés es repeteix molt en un espai curt de temps, la xarxa sense fils no té cap sentit, per tant la preservació i millora del consum energètic és un dels camps més importants en la investigació de les WSN juntament amb la miniaturització d’aquest sensors.

Figura 6. Font d'energia que utilitzen els sensors: piles AA

Un dels mètodes utilitzats per a l'estalvi d'energia és posar els sensors en un estat de *StandBy* quan no necessitin treballar al màxim, això minimitza molt el seu consum, però per determinats sensor no és possible o suficient.

Un altre mètode és minimitzar la informació a enviar. Aquest és el que s'estudia en aquesta memòria.

En la majoria de casos pràctics, la informació dels sensors de la xarxa està correlada i és possible fer una predicció de les mostres a partir de les altres. Mitjançant un processat de senyal anomenat *Distributed Source Decoding* el que es vol aconseguir es una reducció de les dades a enviar per el sensor en funció de les dades dels sensors que hi ha situats al voltant. Aquesta reducció té a veure amb la redundància existent entre les mostres d'un i l'altre sensor, aconseguint així un estalvi d'uns quants bits que a la llarga es traduiran en un gran estalvi en energia per als sensors de la xarxa.

3. Aplicació pràctica

3.1. Explicació del escenari

L'escenari que es veu en aquest cas és el més simple que es pot muntar, dos *Sensor Nodes* i un *Fusion Center* connectat a un *PC* amb el software Matlab®. A la Figura 7 es veu l'escenari descrit.

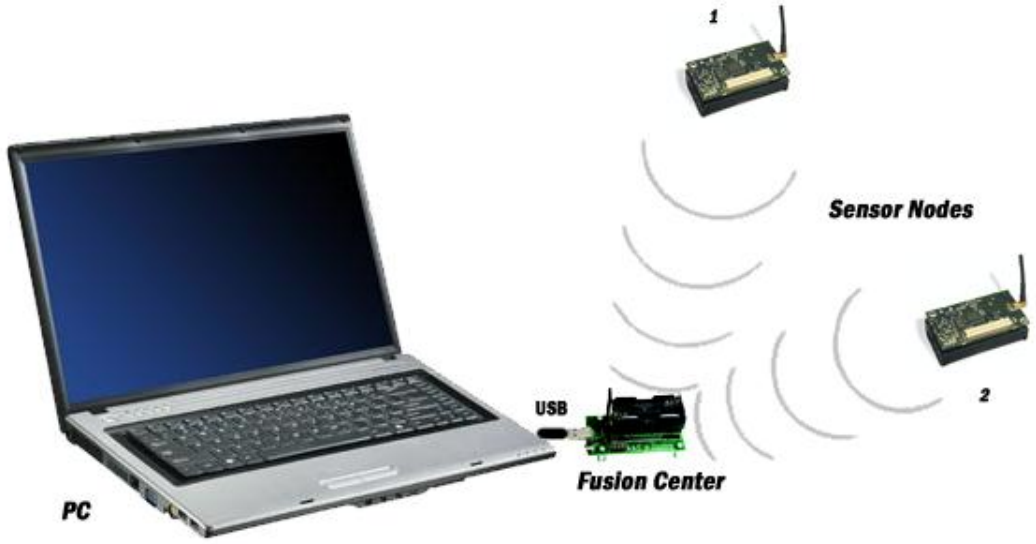


Figura 7. Escenari simple

Encara que l'escenari sigui aquest, l'algoritme contempla el fet de tenir més quantitat de *Sensor Nodes*, per tant és flexible a aquesta variació, però s'ha volgut simplificar per fer-lo el més simple.

Els *Sensor Nodes* prenen mostres de llum i les envien al *Fusion Center*, després aquest envia la mostra al *PC* per a ser processada.

Per a muntar aquest escenari s'utilitzen elements de la plataforma MoteWorks™, la qual es presenta tot seguit.

3.2. MoteWorks™

MoteWorks™ és un kit de desenvolupament per la creació de xarxes de sensors sense fils (WSN). Aquesta plataforma inclou *hardware* de processat i ràdio optimitzats i *software* especialitzat en aquests escenaris que permet multitud d'accions.

El sistema operatiu emprat per a sostenir aquesta WSN és el sistema lliure TinyOS, desenvolupat per la Universitat de California, Berkeley. Aquest sistema operatiu funciona mitjançant components, i s'utilitza l'activació d'esdeveniments per a la seva

programació. Està especialment dissenyat per dispositius de baix consum i poca memòria.

El llenguatge de programació empleat en TinyOS és el *nesC*, un dialecte del *C* modificat per a l'ús d'aquests esdeveniments abans esmenats.

Més endavant, quan es presenti el codi del programa s'explicarà tant la funció dels components com el funcionament dels esdeveniments.

Seguidament es presenten els mòduls[1] de *hardware* que s'utilitzaran i després el *software* emprat per dur a terme aquest cas.

3.3. Hardware

3.3.1. IRIS XM2110

IRIS (Figura 8) és un mòdul que s'utilitza com a plataforma d'enviament per a xarxes sense fils de sensors. Treballa en un rang de freqüències de 2.405 a 2.48 GHz amb canals de 5MHz(reduïble a canals de 1MHz) i està dissenyada per a treballar amb totes les plataformes de sensors sense fils MoteWorks™.

Figura 8. IRIS XM2110

Porta un microcontrolador ATmega1281 el qual executa MoteWorks™ des de la seva memòria flash interna. IRIS suporta la xarxa sense fils i el processat de les dades que recullen els sensors alhora.

És interessant de cara a l'apartat de programació afegir que la placa conté tres LEDs (verd, groc i vermell) fàcilment programables i que l'alimentació tant pot ser mitjançant dues piles AA o a través del connector 51-pin.

[1] Els datasheets dels tres mòduls es poden trobar al Annex C

3.3.2. *Fusion Center*: MIB520CB

El mòdul MIB520 és una placa USB que tant proveeix d'una estació base per a les nostres xarxes de sensors sense fils, com d'una base d'enllaç per a carregar i/o programar els nostres mòduls IRIS que després s'utilitzaran com a *Sensor Nodes* com també el que s'utilitzarà com a *Fusion Center*.

Figura 9. IRIS muntada sobre MIB520CB

El MIB520 va connectat per USB al *PC* i també disposa de tres LEDs (verd, groc i vermell). El mòdul IRIS es connecta a través del connector 51-pin. És important saber que si es connecta un mòdul IRIS a un MIB520, cal apagar l'alimentació de les bateries del primer per a no causar-li danys.

3.3.3. Sensor: MDA100CB

Al mercat hi ha dos famílies de sensors amb suport per a MoteWorks™, els MTS i els MDA. Els MTS són sensors simples i els MDA estan dotats d'un sistema d'adquisició de dades (MoteWorks Data Acquisition).

En aquest cas s'utilitzen sensors de la família MDA, concretament els MDA100CB (Figura 10). Aquests sensors disposen d'un termistor (YSI 44006) que pot mesurar entre -40 ^{o}C fins 70 ^{o}C i d'un sensor de llum del tipus *CdS photocell*. Aquests sensors de llum varien la seva resistència segons la llum que detecten, en aquest cas el sensor quan rep la màxima llum té una resistència de 2kΩ i quan és a les fosques és de 520kΩ. Per tant pel primer cas la sortida serà propera a VCC i pel segon serà el valor de massa o zero.

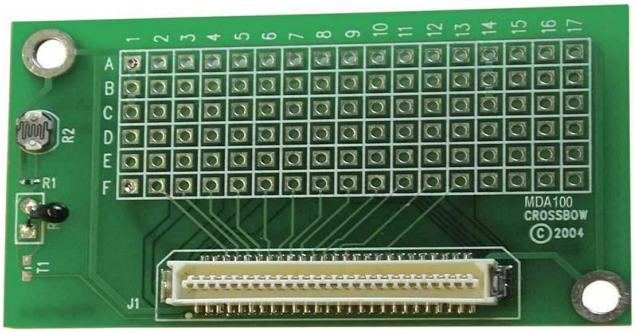


Figura 10. MDA100CB

Si s'observa un cop més la Figura 10 es pot veure el termistor marcat sota la resistència R1 i el sensor de llum que s'identifica a la placa com R2.

3.4. Software

3.4.1. Programmer's Notepad 2®

Aquest entorn de desenvolupament integrat (IDE[2]) és un programa que permet programar el *Fusion Center* i els *Sensor Nodes* amb el llenguatge de programació *nesC*, un dialecte del llenguatge *C*.

El PN2 permet editar el programa, compilar-lo i carregar-lo als mòduls IRIS de la nostra xarxa. S'ha de tenir en compte que s'ha d'especificar el model de plataforma d'enviament que s'utilitza, en aquest cas una *m2110*, per a compilar el programa adientment.

A l'hora de carregar el programa al node, s'obre el *Shell* (tecla F6) i s'escriu el següent codi:

```
make m2110 install,# mib520,com@
```

On el paràmetre # és el número identificador únic que s'assigna al node, normalment zero per al *Fusion Center* i la resta pels *Sensor Nodes*. Per altra banda, el paràmetre @ és el número de port sèrie al que està connectada la base *mib520* per a carregar les dades, aquest valor es troba al administrador de dispositius del *PC*.

Un cop executada satisfactòriament la comanda anterior al *Shell* (tecla F6), ja es té el programa carregat al node i pot començar a fer la seva funció.

[2] Integrated Development Environment

A la Figura 11 s'observa la finestra del programa comentat. Es poden veure les diferents seccions d'aquest i on es treballarà per a cada fi. Al utilitzar la comanda en el *Shell* comentada anteriorment, la sortida sortirà en la finestra de baix, amb els errors i *warnings* que hi pugui haver.

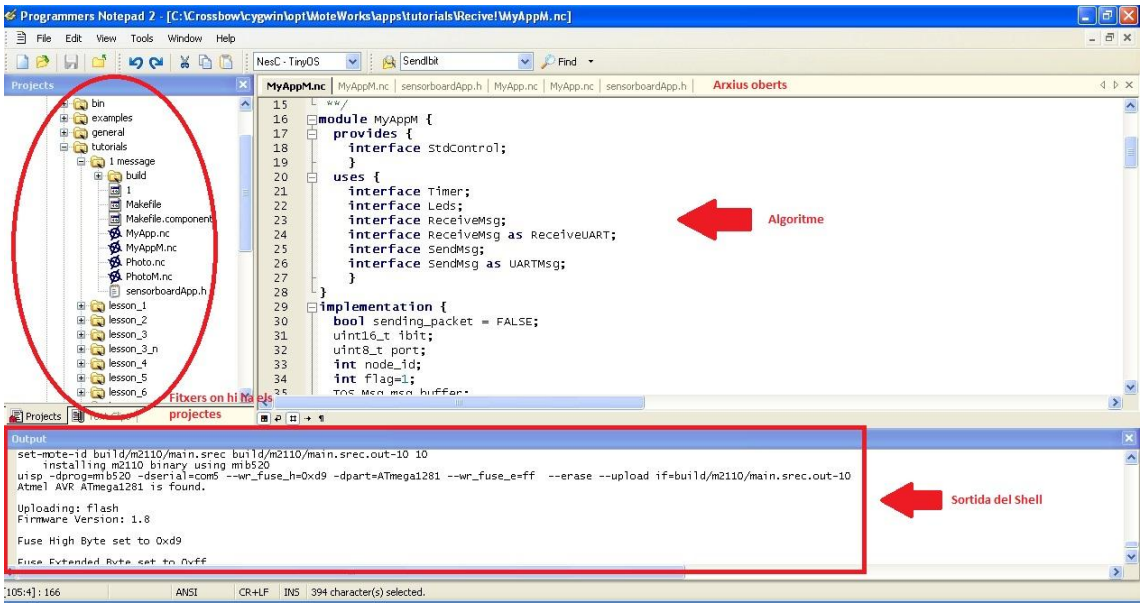


Figura 11. Finestra del Programmer's Notepad 2

3.4.2. XSniffer®

XSniffer® és un software creat per a rastrejar trames, en aquest cas, entre els *Sensor Nodes* i el *Fusion Center*. Per tant a través d'aquest programa es verificarà el comportament de la xarxa, la informació, la estructura de les trames que envien els sensors i la política d'enviament (*Broadcast*[3] o a un sol node).

Es pot observar en la Figura 12, on es té una captura del programa. Com s'ha dit, es veuen totes les parts del paquet, el moment en que es reben, la potència amb la què es reben, etc.

[3] Política d'enviament que envía una dada a tots els nodes de la xarxa.

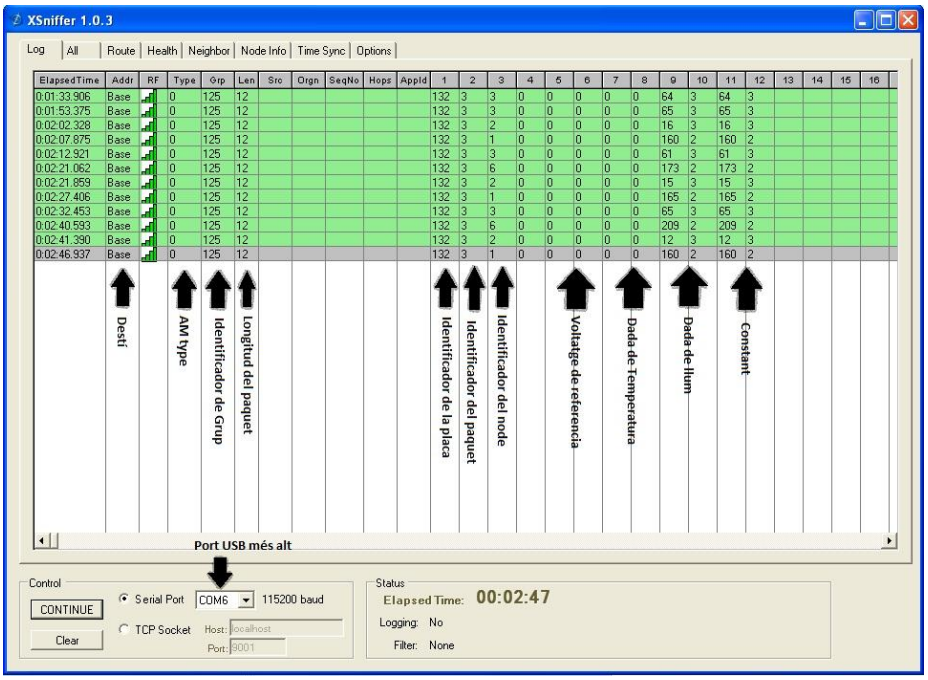


Figura 12. Finestra del XSniffer 1.0.3

3.4.3. Matlab®

Matlab® és un altre IDE que treballa amb un llenguatge de programació propi (llenguatge M). Fonamentalment s'utilitza per a entorns matemàtics, i permet fer varies accions com crear algoritmes, representar funcions i la comunicació amb programes o *hardware* que utilitzin altres llenguatges.

En aquest cas s'utilitza pel processat de les mostres de llum dels *Sensor Nodes* en el *PC*. Mitjançant el port sèrie (USB) rebrem les mostres des del *Fusion Center* i passarem per les etapes pertinents per al càlcul del *Distributed Source Coding*, el qual s'explicarà més endavant.

A la Figura 13 es pot observar de quina manera es divideix la finestra en aquest programa, al mig es té l'editor on escriure funcions que s'executaran per la consola de comandes, la qual es troba a la part de sota. A l'esquerra es tenen dues pestanyes les quals són el fitxer de treball (*Current Folder*) i un espai on es guardarà el valor de totes les variables que anem utilitzant en la funció (*Workspace*).

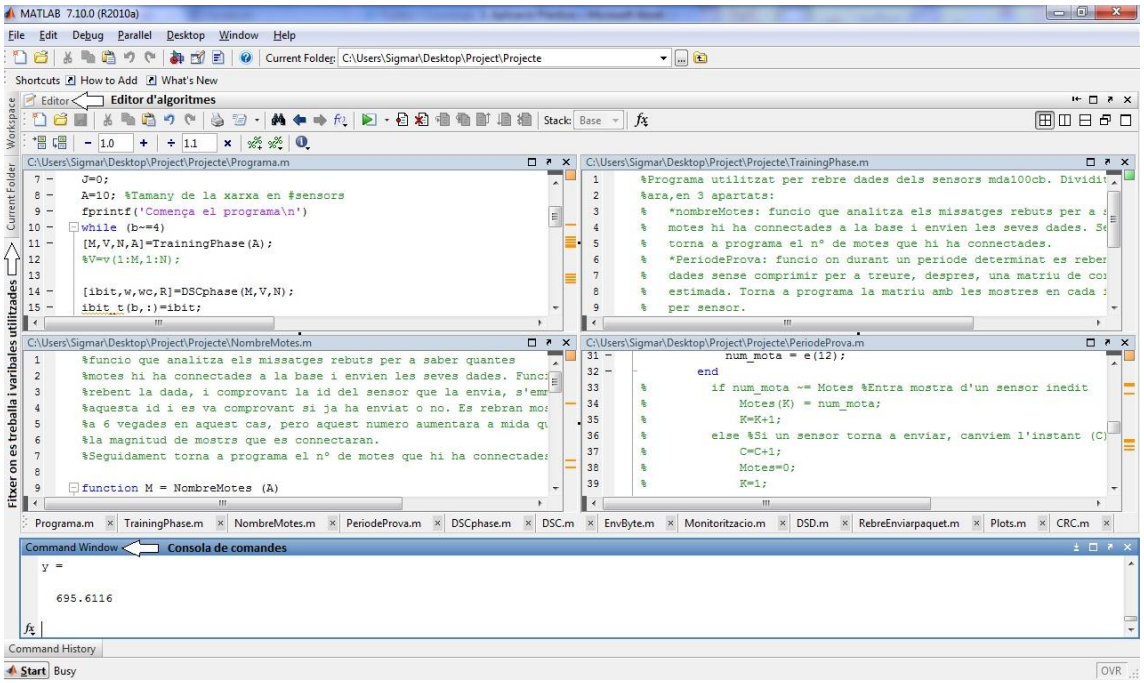


Figura 13. Finestra del Matlab® 7.10

3.5. Programa TinyOS

Un programa implementat amb TinyOS es divideix en tres seccions, les quals s'expliquen tot seguit:

3.5.1. Configuration

Mecanisme per a crear un component mitjançant la combinació directa d'altres.

3.5.2. Implementation

Ens permet definir connexions entre components que proporcionen les interfícies a l'aplicació. Es divideix en dues parts:

1a part: Especificació de components.

Exemple 0
components Main,MyAppM;

2a part: Definició de les relacions entre interfícies de la nostra aplicació i els components.

Si es vol definir que una interfície d'un component correspon a la que utilitza l'aplicació es fa de la següent manera:

- Component.interfície -> MyApp.interficie

Si es volen vincular dues interfícies de components diferents a l'aplicació, es pot utilitzar un pseudònim, que després s'ha d'especificar al *Module,* així:

- Component.interfície -> MyApp.pseudonim

Per un sensor, s'ha d'utilitzar el component *Main* el qual proporciona una interfície anomenada *StdControl*. Aquesta interfície és necessària per a l'aplicació implementada i es descriu més endavant.

Exemple 1

```
implementation {
  components Main, MyAppM, TimerC, LedsC, Photo, GenericComm as Comm;
  Main.StdControl -> TimerC.StdControl;
}
```

Interfícies Paramètriques

Aquestes interfícies serveixen bàsicament per a rebre o enviar paquets. El que es fa es associar la interfície a un paràmetre per a distingir els paquets mitjançant aquest paràmetre. Tenen la forma del exemple:

Exemple 2

```
MyAppM.SendMsg -> Comm.SendMsg[AM_XSXMSG];
MyAppM.ReceiveMsg -> Comm.ReceiveMsg[AM_XSXMSG];
MyAppM.ReceiveIbit -> Comm.ReceiveMsg[AM_XSYMSG];
```

Configuration i *Implementation* es posen al mateix fitxer, el qual té la extensió MyApp.nc.

3.5.3. Module

En aquesta secció és on es programa el comportament de l'aplicació. Aquest anirà a un altre fitxer anomenat com el que conté la *Configuration* i *Implementation* però afegint una M majúscula al final, per tant, anomenat MyAppM.nc.

El *Module* es divideix en tres subseccions:

Provides

Proporciona al llenguatge les interfícies de l'aplicació.

Exemple 3

```
provides {
```

```
   interface StdControl;
}
```

Uses

Informa al llenguatge la utilització d'una interfície.

Exemple 4

```
uses {
   interface Timer;
   interface Leds;
   interface ReceiveMsg as ReceiveUART;
   interface SendMsg;
}
```

Implementation

Conté els mètodes per programar el comportament de l'aplicació. S'hi troben:

- Variables globals per l'aplicació.
- Funcions a implementar degut a les interfícies.
- Esdeveniments a implementar també deguts a les interfícies de l'aplicació.

3.5.4. Header.h

A més de MyApp.nc i MyAppM.nc es pot afegir un arxiu .h que serveix per vincular constants, enumeracions, dades, registres, etc.

Per afegir aquest arxiu als anteriors afegirem la següent línia al principi del fitxer:

- includes header;

En el nostre cas es declara l'estructura dels paquets que s'enviaran en un arxiu amb aquesta extensió.

3.5.5. Tipus de dades

Les dades utilitzades són els típics que trobem al llenguatge C, amb alguns afegits:

- uint16_t : enter sense signe que ocupa 16 bits.
- uint8_t : enter sense signe de 8 bits.
- result_t : ens indica si una funció s'executa amb èxit o no mitjançant un valor *SUCCESS* o *FAIL*.
- bool : valor booleà que pot valer *TRUE* o *FALSE.*

3.5.6.Tipus de funcions

Command

Comanda que s'executen de forma síncrona quan es criden mitjançant la funció *call*:

Exemple 5

```
command result_t StdControl.init() {
   call Leds.init();
   call PhotoControl.init();
}
```

Task

Tasca que s'executa concurrentment, immediatament després de la seva execució el programa continua on s'havia quedat. Es criden amb la funció *post*.

Exemple 6

```
void task IbitMsg(){
        return;
}
post IbitMsg();
```

Event

Esdeveniment que s'activa quan salta una senyal al sistema, també es pot cridar manualment mitjançant la comanda *call*.

Exemple 7

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr msg){
        return msg;
}
```

Aquests tres tipus de funcions es poden executar de forma asíncrona utilitzant el mot *async*, d'aquesta manera la funció s'activarà quan salti una senyal hardware.

Exemple 8

```
async event result_t Light.dataReady(uint16_t data) {
        return SUCCESS;
 }
```

Per últim es destaca la funció *atomic*, la qual permet executar funcions sense por de que es modifiquin les variables utilitzades.

3.5.7. Components primitius

StdControl

El component *Main* i qualsevol programa que vulgui ser una aplicació necessita d'aquest component per funcionar. S'han d'implementar aquestes tres comandes:

- command result_t init();

  S'invoca quan el sensor s'engega o s'endolla.

- command result_t start();

  S'invoca després de l'anterior i quan el sensor passa d'apagat a encès.

- command result_t stop();

  S'invoca quan el sensor passa d'encès a apagat.

GenericComm

Proporciona mecanismes per enviar i rebre paquets via radio i port sèrie. Fa la funció de *switch*.

Es basa en l'enviament de paquets TOS_Msg, el qual té uns paràmetres ja assignats que es mostren tot seguit:

- uint16_t addr;

  Direcció on va el paquet.

  TOS_BCAST_ADDR – direcció *broadcast*

  TOS_LOCAL_ADDRESS – direcció local

  TOS_UART_ADDR – direcció port sèrie

- uint8_t type;

  Tipus de paquet que s'envia.

  SendMsg[TIPUS_PAQUET] – interfície per l'enviament de paquets a un destí, deuen ser del tipus que s'especifica.

  ReceiveMsg[TIPUS_PAQUET] – interfície que permet rebre paquets d'un tipus determinat.

- uint8_t group;

  Grup al que pertany el sensor que envia el paquet.

  El valor per defecte és 0x7D en hexadecimal (125 en decimal)

- uint8_t length;

  Longitud total del paquet.

- uint8_t data [TOSH_DATA_LENGTH];

  El paquet en sí (les dades que s'envien), també anomenat *payload*.

- uint16_t crc;

  Dos bytes que permeten fer la correcció d'errors mitjançant el CRC (*Cyclic Redundancy Check*).

Aquests paràmetres són l'encapsulat que posa l'aplicació per a l'enviament dels paquets. Als paràmetres ja indicats, s'ha d'afegir un byte de principi i final anomenat *Framing byte* que ens indica l'inici i el final del paquet.

Per a que un sensor rebi un paquet, el grup i el tipus d'aquest han de ser els mateixos que els de la interfície i l'adreça la correcta o la de *broadcast*.

SendMsg

És la interfície que permet l'enviament de paquets mitjançant la ràdio o el port sèrie. Cal implementar-la mitjançant la següent comanda:

- command result_t SendMsg.send(uint16_t address,uint8_t length,TOS_MsgPtr msg)

  Per tant cal afegir l'adreça a la que s'envia i la longitud del missatge. L'últim paràmetre és un punter apuntant al missatge.

Si utilitzem aquesta interfície, s'ha d'afegir el següent esdeveniment:

- event result_t sendDone(TOS_MsgPtr msg,result_t succes);

  Aquest esdeveniment s'activa quan l'enviament del paquet es fa satisfactòriament.

ReceiveMsg

És la interfície que ens permetrà rebre paquets, aquesta s'implementa en forma d'esdeveniment ja que no es crida de forma síncrona sinó que s'activarà quan rebem un paquet del tipus correcte.

- event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr msg);

Interfície ADC

Ens permet obtenir el valor del sensor a l'aplicació que l'utilitza. Hi ha dos mètodes de crida de la comanda per a recollir un valor:

- async command result_t getData();

  Es recull la mostra del sensor, un cop capturada s'activa l'esdeveniment dataReady().

- async command result_t getContinuousData();

  Recull una seqüència continua de mostres.

Es pot deduir que amb aquesta interfície s'ha d'implementar el següent esdeveniment:

- async event result_t dataReady(uint16_t data);

  S'activa quan es té la mostra del sensor i es guarda el valor a la variable *data*.

Cal ressaltar que tant les comandes com l'esdeveniment són asíncrones ja que s'activaran quan rebin una resposta hardware.

TimerC

És la interfície comptador, ens permet fer accions cada cert temps. Es poden tenir fins a 10 *Timers* diferents en una aplicació. Forma part de les interfícies parametritzades i cal afegir-hi un identificador durant la implementació:

- TimerC.Timer[Id];

La interfície s'inicialitza mitjançant la següent crida:

- call Timer.start(char tipus,uint32_t interval);

  L'interval és el temps que es vol que passi entre una activació i un altra i el tipus pot ser o TIMER_ONE_SHOT (s'activa una sola vegada) o TIMER_REPEAT (s'activa repetidament).

- call Timer.stop();

  Aquesta crida serveix per parar el comptador.

Si s'utilitza aquesta interfície s'ha d'implementar el següent esdeveniment:

- event result_t Timer.fired();

  Aquest esdeveniment s'activa cada vegada que es crida Timer.start() i cada interval de temps mentre el tipus del *Timer* sigui TIMER_REPEAT.

3.6. Programar en Matlab®

Matlab® és un software molt empleat en entorns de desenvolupament. Utilitza el llenguatge M, molt similar al C i que per segons quines tasques és més senzill. Al ser un sistema més conegut que TinyOS, la intenció d'aquest apartat no és fer un tutorial, si no introduir alguna funció menys vista en l'entorn i que s'utilitza per implementar l'algoritme aquí estudiat.

Per tant, el que es veurà en aquest apartat és com implementar la comunicació a través del port sèrie entre el *Fusion Center* i el *PC*. Per a aquest fi, es disposa d'una sèrie de funcions creades per a la recepció i enviament de dades a través del port, però primerament s'han d'inicialitzar uns valors per a establir la comunicació:

Inicialització del port sèrie

```
s=serial ('com8'); % creació del port
set(s,'Baudrate',57600); % velocitat de transmissió a 57600 Baudios
set(s,'StopBits',1); % es llegeix cada 1 bit
set(s,'DataBits',8); % la dada és de 8 bits
set(s,'Parity','none'); % sense bit de paritat
set(s,'Terminator','CR/LF');% "c" caràcter amb el que finalitza l'enviament
set(s,'OutputBufferSize',length(msg)); % "length(msg)" és el número de bytes a enviar
set(s,'InputBufferSize' ,24); % "24" és el número de bytes a rebre
set(s,'Timeout',30); % 30 segons d'espera fins a rebre alguna dada
```

Amb la funció *set* s'inicialitza el port sèrie segons els paquets, la velocitat i el número de port per on es reben. El següent és mostrar les funcions utilitzades per obrir, tancar, llegir i escriure al port sèrie.

Funcions de manipulació del port sèrie

```
fopen(s); %s'obre el port sèrie
packet = fread(s); %es llegeix el paquet del port sèrie i es guarda en la variable packet
fwrite(packet); %s'escriu al port sèrie la variable packet
fclose(s); %es tanca el port sèrie
```

Per tant ja s'han vist les funcions utilitzades per a la comunicació a través del port sèrie. És important tancar sempre el port sèrie després de llegir o escriure si no s'ha de tornar a utilitzar ja que sinó aquest quedarà obert i cap altra aplicació hi podrà accedir.

4. Distributed Source Coding

4.1. Funcionament

La finalitat del *Distributed Source Coding* (DSC) és l'estalvi de bits d'informació en els paquets enviats, i per tant, d'energia de transmissió.

El DSC treballa de tal manera que elimina la redundància entre lectures correlades, és a dir, es troba la informació comuna entre mesures i es redueix mitjançant una codificació distribuïda.

Aquesta reducció es fa mitjançant el càlcul d'una variable anomenada *ibit*, la qual ens indica fins a quin punt es pot reduir la mostra en nombre de bits sense perdre informació. Per tant el sensor codificarà la seva dada utilitzant el *ibit* que li pertoca com a referència i més tard, sabut aquest valor, es podrà descodificar la mostra sense cap problema.

Aquests bits que es guanyen amb la codificació es tradueixen en energia estalviada en cada transmissió, el que significa un estalvi total gran considerant que la majoria de sensors de les WSN utilitzen fonts d'alimentació amb poca capacitat i que cal administrar.

4.2. Coneixements previs

4.2.1 Filtre de Wiener

El filtre de Wiener és un filtre digital òptim que va néixer durant la dècada dels 40 donada la necessitat de dissenyar un filtre que fes l'estimació òptima d'una mesura sorollosa[1]. Es pot veure a la Figura 14 la forma discreta del filtre de Wiener que pretén recuperar una senyal *d(n)* d'una mostra sorollosa.

$$x(n) = d(n) + v(n)$$

Assumint que $d(n)$ i $v(n)$ són processos aleatoris estacionaris en sentit ampli (wss), Wiener considera el problema de dissenyar un filtre que produeixi una mínima estimació del *Mean-Square Error* de $d(n)$:

$$\xi = E\{|e(n)|^2\}$$

On

$$e(n) = d(n) - \hat{d}(n)$$

$$W(z) = \sum_{l=0}^{p-1} w(n) z^{-n}$$

$$\hat{d}(n) = \sum_{l=0}^{p-1} w(l) x(n-l)$$

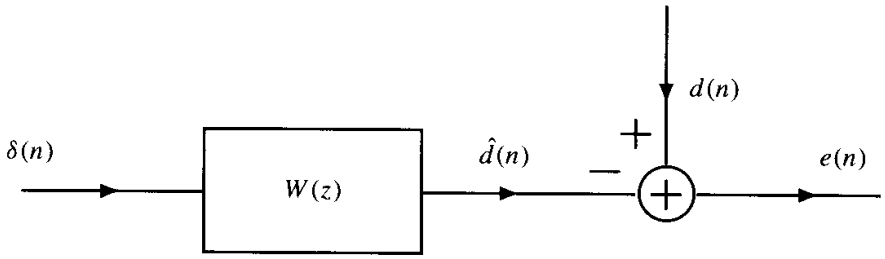


Figura 14 Problema per a la utilització d'un filtre de Wiener. Quan $\boldsymbol{\delta(n)}$ i $\boldsymbol{d(n)}$ son estacionaris en sentit ampli (wss), el filtre W(z) ens produirà la mínima estimació del Mean-Square Error, $\boldsymbol{\hat{d}(n)}$ de $\boldsymbol{d(n)}$.

Per a trobar la solució del problema de disseny del filtre de Wiener s'han de trobar els coeficients del filtre, *w(k)*, que minimitzin el MSE

$$\xi = E\{|e(n)|^2\} = E\{\left|d(n) - \hat{d}(n)\right|^2\}$$

Per minimitzar el MSE s'han de trobar els valors que facin que la derivada en funció de *w*(k)* sigui igual a zero

$$\frac{\delta\xi}{\delta w^*(k)} = \frac{\delta}{\delta w^*(k)} E\{e(n)e^*(n)\} = E\left\{e(n)\frac{\delta e^*(n)}{\delta w^*(k)}\right\} = 0$$

Amb

$$e(n) = d(n) - \sum_{l=0}^{p-1} w(l) x(n-l)$$

Seguit de

$$\frac{\delta e^*(n)}{\delta w^*(k)} = -x^*(n-k)$$

Es troba

$$E\{e(n)x^*(n-k)\} = 0 \;\; ; \;\; k = 0, 1, \ldots, p-1$$

Seguint el principi d'ortogonalitat del teorema de la projecció es té

$$E\{d(n)x^*(n-k)\} - \sum_{l=0}^{p-1} w(l)E\{x(n-l)x^*(n-k)\} = 0$$

Finalment, tenint en compte que $x(n)$ i $d(n)$ són wss (estacionaris en sentit ampli) llavors $E\{x(n-l)x^*(n-k)\} = r_x(k-l)$ i $E\{d(n)x^*(n-k)\} = r_{dx}(k)$, per tant

$$\sum_{l=0}^{p-1} w(l)r_x(k-l) = r_{dx}(k) \quad ; \quad k = 0, 1, \dots, p-1$$

Si es desenvolupa el sumatori es troba un grup de *p* equacions lineals amb les *p* incògnites $w(k), k = 0, 1, \dots, p-1$. Si es posa en forma de matriu, sabent que la autocorrelació és simètrica conjugada, $r_x(k) = r_x^*(-k)$, es veu el següent

$$\begin{bmatrix} r_x(0) & r_x^*(1) & \dots & r_x^*(p-1) \\ r_x(1) & r_x(0) & \dots & r_x^*(p-2) \\ r_x(2) & r_x(1) & \dots & r_x^*(p-3) \\ \vdots & \vdots & & \vdots \\ r_x(p-1) & r_x(p-2) & \dots & r_x(0) \end{bmatrix} \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \vdots \\ w(p-1) \end{bmatrix} = \begin{bmatrix} r_{dx}(0) \\ r_{dx}(1) \\ r_{dx}(2) \\ \vdots \\ r_{dx}(p-1) \end{bmatrix}$$

Aquesta és la forma matricial de les equacions de Wiener-Hopf. Es pot expressar de forma reduïda així

$$\boldsymbol{R_x w = r_{dx}}$$

On $\boldsymbol{R_x}$ és una matriu Hermítica Toeplitz de dimensió *p x p*, $\boldsymbol{w}$ és el vector dels coeficients del filtre i $\boldsymbol{r_{dx}}$ és el vector de correlacions creuades entre la senyal desitjada $d(n)$ i la senyal observada $x(n)$.

Per tant, si s'aïlla el vector de coeficients del filtre es troba l'equació que utilitzarem en el DSC

$$\boldsymbol{w = R_x^{-1} r_{dx}}$$

4.3. Algoritme

Bàsicament, el DSC permetrà calcular el *ibit*, que serà el nombre de bits al qual es pot reduir una mostra per després descodificar-la satisfactòriament.

Prèviament a això, s'ha de saber quants sensors tenim a la xarxa i treure'n la matriu de correlacions, la qual permetrà saber fins a quin punt estan les mostres correlades i extreure'n la redundància.

Aquesta matriu[4] de correlacions es troba a partir de les correlacions de les mostres dels sensors amb l'equació següent:

$$\boldsymbol{R} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}(n)\boldsymbol{x}(n)^H$$

On $x(n)$ es el vector de mostres de tots els sensors en un instant *n* determinat i N són el nombre d'instants totals que s'han mesurat per a extreure la matriu.

Aquest pas previ s'explica més endavant, ara ens centrem en el DSC solament.

Un cop sabudes aquestes dades, s'extreuen una sèrie de paràmetres a partir d'elles com el vector de correlacions del sensor al que es sotmet el DSC ($\boldsymbol{r}_{\boldsymbol{x}}'$), i una nova matriu de correlacions també en funció del sensor sotmès ($\boldsymbol{R}'$).

Calculades aquestes dues dades, es pot trobar el pes del sensor (la importància del sensor dins la xarxa, donada per la proximitat, el nivell de redundància, etc.) mitjançant el càlcul dels coeficients d'un filtre de Wiener, explicat en l'apartat anterior i que proporciona el mínim Mean Square Error (MSE).

Tenint els coeficients es pot calcular una estimació de la mostra rebuda i a través d'aquesta tenir el valor del mínim MSE. Aquests dos termes es poden utilitzar com a valors de control per a saber quan s'ha d'actualitzar la matriu de correlacions.

4.3.1 Càlcul del *ibit*

Es calcula el valor del *ibit* que és el que realment interessa per aconseguir l'objectiu proposat.

[4] A partir d'aquest punt, les matrius es representaran en majúscula i negreta (**R**) i els vectors en minúscula i negreta (**r**).

Suposem que codificant la mostra amb el nombre d'*ibits* adequat, la mostra descodificada i la mostra real han de ser iguals, per tant es compleix $2^{i-1}\Delta > |\boldsymbol{N}_{k,j}|$ on Δ és l'espai entre dues mostres reals consecutives, $N_{k,j}$és la diferència entre el valor a la entrada i el de la sortida ($\boldsymbol{N}_{k,j} = \boldsymbol{x}_k^{(j)} - \boldsymbol{y}_k^{(j)}$) i *i* és l'*ibit*. Si no és el cas, la desigualtat és inversa $\left|\boldsymbol{N}_{k,j}\right| > 2^{i-1}\Delta$ i per tant hi ha un error de descodificació. Si s'utilitza la desigualtat de Chebyshev es pot trobar la probabilitat d'error.

$$P\left[\left|N_{k,j}\right| > 2^{i-1}\Delta\right] \leq \frac{\sigma_{N_j}^2}{(2^{i-1}\Delta)^2}$$

$N_{k,j}$ té una distribució de mitja zero i variança $\sigma_{N_j}^2$. Per tant la màxima probabilitat d'error que es pot tenir és la que compleix la igualtat, i conseqüentment es pot trobar el valor del *ibit* òptim en funció d'aquests paràmetres.

La formula per a fer-ho la trobem a [2] i és la següent:

$$ibit = \left\lfloor \frac{1}{2}\log_2\left(\frac{\sigma_{N_j}^2}{\Delta^2 P_e}\right) + 1 \right\rfloor$$

On $\sigma_{N_j}^2$ és la potencia d'error, la qual es calcula a partir de la matriu **R** i $\boldsymbol{r}_N$ que és el vector de correlacions creuades que s'escull de la matriu **R** com es mostra tot seguit

$$\boldsymbol{R} = \begin{bmatrix} \sigma_1^2 & r_{12} & \dots & r_{1n} \\ r_{21} & \sigma_2^2 & \dots & r_{2n} \\ \vdots & \vdots & & \vdots \\ r_{n1} & r_{n2} & \dots & \sigma_n^2 \end{bmatrix}$$

$$\boldsymbol{r_N} = [r_{N1}, r_{N2}, \dots, r_{N,N-1]}]$$

Per tant, la potencia d'error es calcula així

$$\sigma_{N_j}^2 = \sigma_{N+1}^2 - \boldsymbol{r_{N+1}^T} \cdot \boldsymbol{R}^{-1}$$

Δ, com ja s'ha comentat, és l'espai entre dues mostres reals consecutives. En aquest cas, el nombre de mostres reals totals que podem mesurar és 2^{16}, que coincideix amb les combinacions de bits que podem enviar, per tant $\Delta = 1$.

Per últim, P_e és la probabilitat d'error que nosaltres creurem òptima o que ens demanaran com un mínim per a que el sistema funcioni correctament, en aquest cas l'escollirem nosaltres i serà $P_e=10^{-2}$.

Tenint aquests valors es podrà calcular un valor d'*ibit* òptim amb el qual es podrà codificar la mostra del sensor.

4.3.2. Càlculs de control

Com ja s'ha comentat més a dalt, es podrà mantenir el control sobre la matriu de correlacions calculada en l'etapa de *Training* (es veurà en profunditat al següent apartat) mitjançant el càlcul, bàsicament, dels coeficients del filtre de Wiener o pes dels sensors, el qual donarà una estimació de les mostres que es reben i un valor de MSE que es podrà monitoritzar per saber quan actualitzar la matriu de correlacions.

Aquest control s'ha de mantenir ja que amb el pas del temps poden haver-hi canvis (moviment de sensors, entrada i/o sortida de nous sensors a la xarxa, etc.) que variïn la matriu de correlacions i per tant, el sistema.

Per al càlcul dels coeficients de Wiener es segueixen els passos explicats més a dalt

$$\boldsymbol{w_N} = \boldsymbol{R_{NxN}^{-1}} \cdot \boldsymbol{r_{Nx1}}$$

Calculat el vector de pesos, es pot fer una estimació del valor de la mostra així

$$\widehat{\boldsymbol{y}}_{\boldsymbol{N}} = \sum_{i=0}^{N} \boldsymbol{w_i^T} \cdot \boldsymbol{x_i}$$

I finalment calcular el MSE com segueix

$$MSE = \left|\left|\boldsymbol{x_N} - \widehat{\boldsymbol{y}}_{\boldsymbol{N}}\right|\right|^2$$

4.3.3. Descodificació

El mètode per descodificar és bastant senzill, mitjançant la creació d'un vector de posicions igual a les combinacions possibles a rebre, es van destriant posicions distingint entre les parells i les senars, segons el valor del bit de la mostra codificada començant pel menys significatiu fins al més significatiu tal com es veu a la Figura 15 extreta de [1].

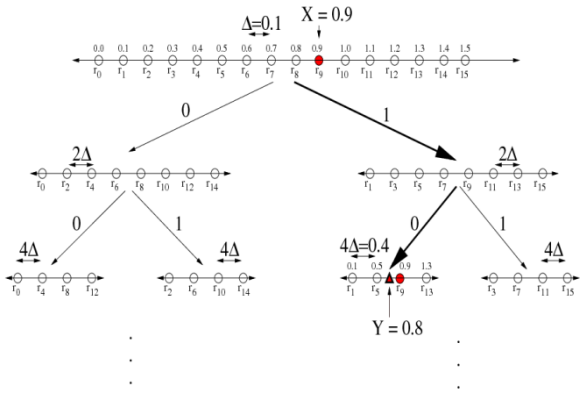


Figura 15. Arbre de descodificació a partir d'una paraula codificada 01 [1].

Així finalment fent una aproximació amb el valor estimat ($\hat{y}$) s'aconsegueix la descodificació.

4.3.3. Exemple

Seguidament es veurà amb exemples com quedaran dites dades, i com es calcularan els paràmetres a partir d'una xarxa de 3 sensors:

$$\boldsymbol{R} = \begin{bmatrix} \sigma_1^2 & r_{12} & r_{13} \\ r_{21} & \sigma_2^2 & r_{23} \\ r_{31} & r_{32} & \sigma_3^2 \end{bmatrix}$$

Càlcul DSC per al sensor 2:

$$\boldsymbol{r_2} = r_{21} \qquad \boldsymbol{R'} = \sigma_1^2$$

$$\boldsymbol{w_2} = \boldsymbol{R'^{-1}} \cdot \boldsymbol{r_2} = (\sigma_1^2)^{-1} \cdot r_{21} \qquad \boldsymbol{\hat{y}_2} = \sum_{i=1}^{3} \boldsymbol{w_n^T} \cdot \boldsymbol{x_n} = \boldsymbol{w_1}^T \cdot \boldsymbol{x_1} + \boldsymbol{w_2}^T \cdot \boldsymbol{x_2}$$

Càlcul DSC per al sensor 3:

$$\boldsymbol{r_2} = [r_{21} r_{23}] \qquad \boldsymbol{r_3} = [r_{31}\, r_{32}] \qquad \boldsymbol{R'} = \begin{bmatrix} \sigma_1^2 & r_{12} \\ r_{21} & \sigma_2^2 \end{bmatrix}$$

$$\boldsymbol{w_2} = \boldsymbol{R'^{-1}} \cdot \boldsymbol{r_2} = \boldsymbol{R'} = \begin{bmatrix} \sigma_1^2 & r_{12} \\ r_{21} & \sigma_2^2 \end{bmatrix}^{-1} \cdot [r_{21} r_{23}]$$

$$\boldsymbol{w_3} = \boldsymbol{R'^{-1}} \cdot \boldsymbol{r_3} = \begin{bmatrix} \sigma_1^2 & r_{12} \\ r_{21} & \sigma_2^2 \end{bmatrix}^{-1} \cdot [r_{31}\, r_{32}]$$

$$\boldsymbol{\hat{y}_3} = \sum_{i=1}^{3} \boldsymbol{w_n^T} \cdot \boldsymbol{x_n} = \boldsymbol{w_1}^T \cdot \boldsymbol{x_1} + \boldsymbol{w_2}^T \cdot \boldsymbol{x_2} + \boldsymbol{w_3}^T \cdot \boldsymbol{x_3}$$

El primer sensor no es codificarà, ja que serà el referent que es té per a treure la redundància existent entre aquest i els altres.

4.3.4. L'algoritme en Matlab®

En aquest apartat es veurà com s'implementa el DSC per Matlab®. L'algoritme en pseudocodi és el següent:

Algoritme DSC per Matlab®

Es té una matriu de mostres dels sensors mesurades durant un temps N (fase *Training*) anomenada V(MxN).
Es calcula la matriu de correlacions R:
for n=1:M
  for m=1:M
  R(n,m)=1/N*(V(n,1:N)*V(m,1:N)');
  end
end
Calculem l'*ibit* de cada sensor a partir del segon.
for i=2:M
  σ=R(i-1,i-1); %potencia del senyal o autocorrelació
  $\hat{r}$=R(1:i-1,i); %vector de correlacions del sensor que em de reduir el senyal
  $\hat{R}$=R(1:i-1,1:i-1); %nova matriu de correlacions
  w(i-1,:)=($\hat{R}^{-1}$*$\hat{r}$); %vector de pesos del filtre de Wiener
  Pot_{err} = σ -($\hat{r}^{H}$*$\hat{R}^{-1}$); %Potencia d'error
  Δ= 1; %espai entre dues mostres reals
  P_e = 1e-1; %probabilitat d'error
  Ibit_t= abs(0.5*$\log_2$(Pot_{err}* P_e*Δ^2))+1); %ibit del sensor vinent
  ibit(i) = ceil(ibit_t); %es redondeja al valor més próxim i s'emmagatzema al vector ibit
end

Es pot observar que primerament es fa el càlcul de la matriu de correlacions a partir de les mostres capturades durant la fase *Training* i més tard es calcula en cada iteració el valor del *ibit,* el qual s'envia més tard a cada sensor.

5. Algoritme d'energia eficient

En aquest apartat de la memòria s'explica en què consisteix l'algoritme emprat i es descriuen les parts d'aquest per tal d'entendre el funcionament del *Distributed Source Coding* DSC, explicat en l'apartat anterior.

Cal recordar que es tenen dos màquines (*hardware*) treballant sincronitzades, el *PC* mitjançant el software Matlab®, i els sensors, programats en un dialecte del C anomenat NesC. La comunicació serà entre el *PC* i el *Fusion Center* o mota base. La mota base s'encarregarà de canviar les rutines d'enviament dels sensors per a les diferents etapes.

Per tant tot seguit es descriuen els diagrames de blocs de cada una de les parts del sistema (*PC, Fusion Center, Sensing Nodes*). Més tard es descriuran els codis que formen els programes de cadascuna de les plataformes amb més cura.

5.1. Diagrames de blocs

5.1.2. PC

En primer terme es parla del programa en Matlab®, a la Figura 16 es veu el seu diagrama de blocs. El programa està dividit en quatre etapes, les dues primeres pertanyen al apartat de *Training,* d'on s'extreuen el nombre de motes que hi ha al sistema i la matriu de correlacions existent entre motes a partir dels valors de les mostres. Les dues darreres etapes són el *Distributed Source Coding* (DSC) i la seva descodificació (DSD), esmentats en l'apartat 3.

Tot seguit es descriuen aquestes quatre etapes:

- **NombreMotes:** en aquesta etapa la finalitat principal és determinar quants sensors té la nostra xarxa. El procés tracta que durant un temps determinat (el qual es pot variar), es recullen mostres i s'analitza de quin sensor provenen. Com que cada sensor ha de tenir un identificador únic, es va comptant de quants sensors rebem i quins són.

Aquesta etapa s'activa sempre que es trobi un canvi en la nostra xarxa, és a dir, quan s'estigui en un altre etapa i es detecti un identificador no conegut. El fet d'afegir un sensor nou variarà la matriu de correlacions i per tant la codificació de les mostres.

- **PeriodeProva:** bàsicament aquesta etapa tracta d'emmagatzemar mostres sense codificar dels sensors per tal de trobar la matriu de correlacions que les relaciona. Es recullen mostres durant 2M instants de temps, on M és el número de sensors que hi ha a la xarxa, trobat durant l'etapa anterior.

Per tant, si el número de sensors de la xarxa varia, també s'ha de tornar a fer una crida a aquesta etapa. Amb el pas del temps la matriu de correlacions s'ha d'actualitzar ja que per diferents motius (climàtics, variacions en la xarxa, etc.) la correlació entre sensors variarà i la matriu pot conduir al error. Per tant es farà una crida a les denominades etapes *Training* (NombreMotes i PeriodeProva) periòdicament per mantenir la matriu actualitzada.

Un cop emmagatzemades les mostres, podem calcular amb aquestes la matriu de correlacions.

- **DSC:** l'etapa de *Distributed Source Coding* és la que determinarà com codificar les mostres dels sensors. Tal com s'ha explicat en l'apartat 4.Distributed Source Coding d'aquesta memòria, aquesta etapa fonamentalment es dedica a calcular el *ibit* de cada sensor, és a dir, la quantitat de bits mínima necessària que ha d'enviar cada sensor per a poder descodificar la mostra satisfactòriament.

Un cop acabades les etapes de *Training* es fa aquesta etapa de nou per a actualitzar l'*ibit* segons la nova matriu de correlacions.

- **DSD i MONITORITZACIÓ:** finalment tenim l'etapa *Distributed Source Decoding,* on es descodifiquen les mostres rebudes ja codificades per l'*ibit,* i l'etapa de monitorització d'on s'extreuen els resultats que volem comprovar (error, compressió, etc.).

Segons es pot deduir, les quatre etapes es poden separar en dues que es van repetint infinitament durant el procés de recol·lecció de mostres. Aquestes són l'etapa *Training* i l'etapa *Distributed Source Coding*. Es representen en un eix temporal en la Figura 17.

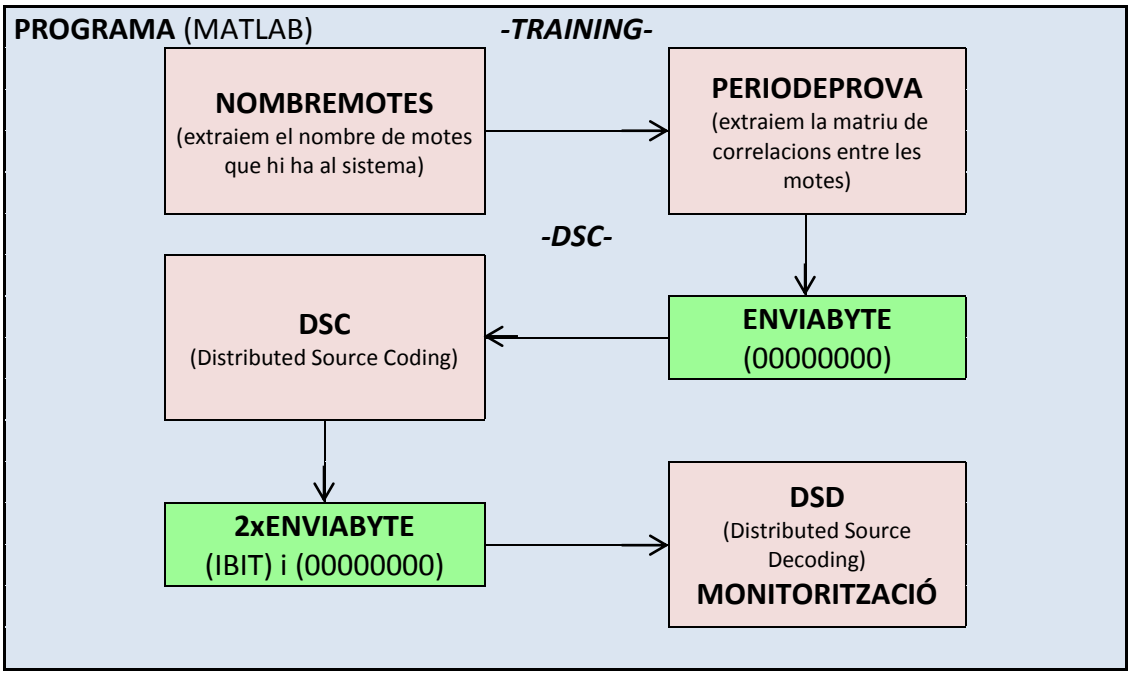


Figura 16. Diagrama de blocs del programa en Matlab®

Es pot veure que el temps que estem en l'etapa *DSC* no és constant, això és degut a que durant aquesta etapa pot haver-hi una interrupció a causa de la inserció d'un nou sensor o la detecció d'un error massa gran que faci que s'hagi de tornar a l'etapa de *Training* per tornar a fer els càlculs previs. En el nostre cas s'ignora l'aparició d'un nou sensor i aquest no tindrà efecte fins la següent etapa *Training*.

Training	*DSC*	*Training*	*DSC*	*Training*	t

Figura 17. Representació de les etapes *Training* i *DSC* amb el pas del temps

Per acabar, es pot observar una etapa entre el *Training* i el *DSC* i una altra entre el *DSC* i el *DSD*.

L'etapa EnviaByte indica que s'envia un byte (el que surt entre parèntesis) a través del port sèrie al *Fusion Center*. Com es veurà en la descripció dels programes dels sensors, el byte tot zeros indicarà un canvi de rutina en l'enviament, és a dir, serà un *flag* per alertar que es passa de la rutina *Training* a la rutina *DSC*.

La segona etapa EnviaByte simplement és l'enviament del *ibit* i l'identificador del sensor a qui pertoca al *Fusion Center*, el qual s'encarregarà d'enviar-ho al sensor pertinent i un cop fet, s'enviarà el bit tot zeros per tornar a la rutina d'enviament inicial mitjançant aquest cop l'*ibit*.

5.1.2. Sensor Node

En aquest apartat es descriu com estan programats els sensors de la xarxa. A la Figura 18 es pot veure el diagrama de blocs d'aquest programa. Bàsicament el programa es basa en els esdeveniments de rebre i enviar paquets.

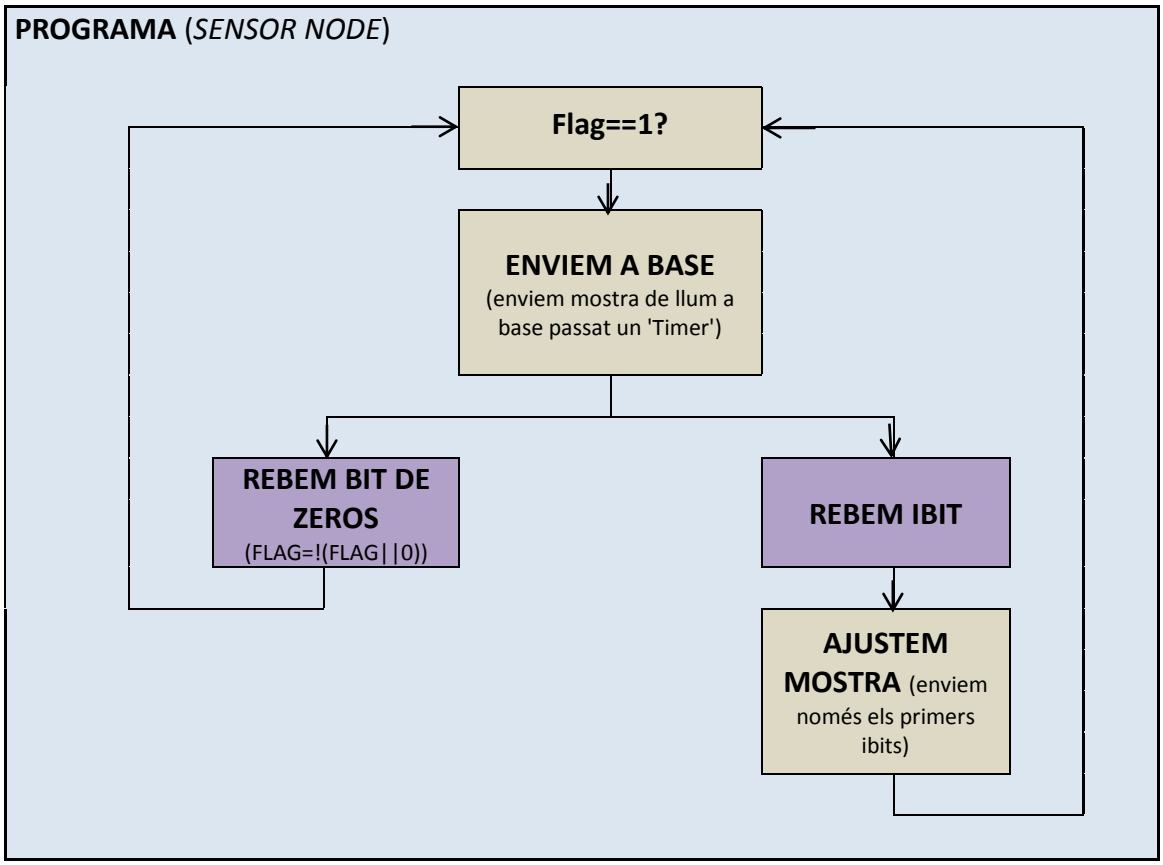


Figura 18. Diagrama de Blocs del programa del *Sensor Node*

Al encendre el sensor, aquest calcularà la llum que rep i l'enviarà al *Fusion Center*, llavors al cap d'un temps determinat (variable segons el nombre de motes a la xarxa), tornarà a enviar una altra mostra.

El *Sensor Node* està programat per diferenciar entre dos tipus de paquets, aquest detall s'explica més endavant, ara només els diferenciarem entre un paquet X, el qual és del tipus que envia el propi *Sensor Node* amb espai per les dades de llum i d'una constant, i un paquet Y, el qual conté menys informació i serveix per alertar al sensor del canvi de rutina d'enviament.

Per tant rebrem l'*ibit* en l'espai de la constant del paquet X i el *flag* en l'estructura de paquet Y.

Vist des del punt de vista del programa en Matlab®, durant l'etapa de *Training* el *Sensor Node* va enviant mostres de llum repetidament al *Fusion Center* i un cop a l'etapa *DSC,* enviat el *flag* de canvi de rutina i rebut el *ibit*, envia la mostra codificada de la mateixa manera que abans.

Algoritme del *Sensor Node*

1.Etapa *Training*
if (flag==1) **//indica que estem en rutina d'enviament**
Envia mostra de llum cada T segons.
end
2.Etapa *DSC*
Es rep la màscara de l'*ibit* des del *Fusion Center.*
AND lògica entre la mostra de llum i la màscara de l'*ibit.*
if (flag==1)
    Envia la mostra de llum codificada cada T segons.
end

5.1.3. *Fusion Center*

Pel programa del *Fusion Center* es distingeixen dues rutines, una quan es rep un paquet des d'un *Sensor Node* i l'altre quan es rep des del *PC*. Les dues es poden observar en la Figura 19.

Paquet del *Sensor Node*: quan es rep un paquet d'un dels sensors, el *Fusion Center* l'encapsularà i l'enviarà al *PC*.

Paquet del *PC*: com s'ha vist en l'explicació del programa en Matlab®, només es rep un paquet del *PC* per dues raons, el canvi de rutina entre l'etapa de *Training* i l'etapa *DSC* i la recepció del *ibit* d'algun dels sensors. Per tant, caldrà identificar quina és la raó de rebre aquest paquet i fer les accions pertinents.

- Es rep el byte tot zeros: aquest byte indica un canvi de rutina, s'ha escollit aquest byte perquè és impossible que l'*ibit* sigui tot zeros, ja que indicaria que dos sensors tenen màxima redundància, per tant, que mesuren el mateix sempre, i això és pràcticament impossible.

El canvi de rutina consisteix en enviar un paquet Y amb el camp del *ibit* a 0 al sensor per a què aquest torni posi a zero el *flag* d'enviament, això vol dir que el *PC* ja ha entrat en l'etapa *DSC* i, per tant, està calculant els *ibits* dels sensors de la xarxa.

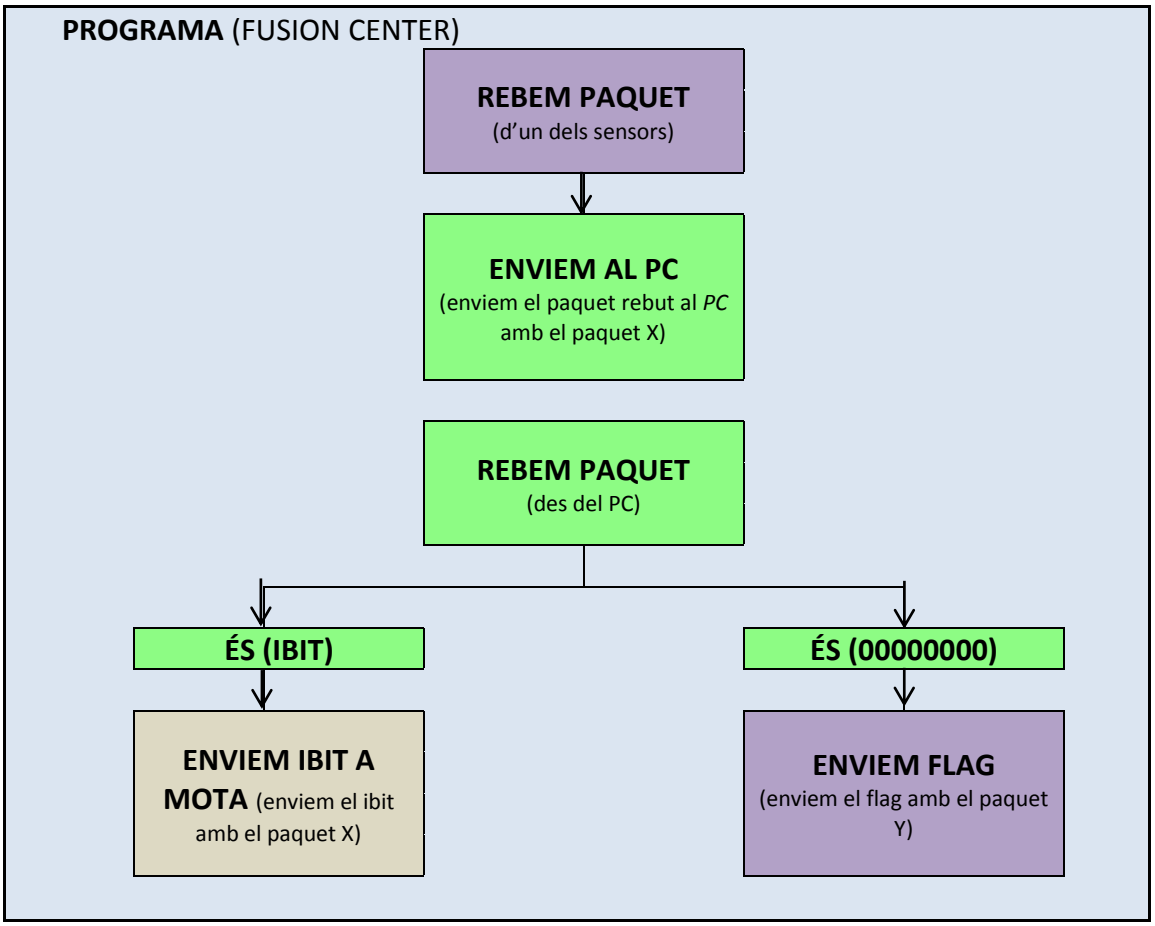


Figura 19. Diagrama de blocs del programa *Fusion Center*

- Es rep l'*ibit*: un cop el *PC* envia l'*ibit* i l'identificador del sensor, el *Fusion Center* envia aquest al sensor pertinent en un paquet X i espera. Un cop enviats tots els *ibits* torna a enviar els *flags* per a que els sensors enviïn les mostres codificades i s'entra en l'etapa *DSD* del programa en Matlab®.

Algoritme del *Fusion Center*

1.Es rep paquet del *Sensor Node*
Encapsulem i enviem mostra al *PC*
2.Es rep paquet del *PC*
if (ibit==0)
    Enviem paquet Y amb l'*ibit* al sensor corresponent
else
    Enviem paquet X amb l'*ibit* al sensor corresponent
end

5.2. Programa

Tot seguit s'expliquen més profundament els codis utilitzats en cada plataforma (*PC, Sensor Nodes* i *Fusion Center*) mitjançant un pseudocodi.

Primerament es comenta el codi emprat en Matlab®, que és el codi utilitzat en el *PC*.

5.2.1. PC

Aquest programa és el més complex de les tres parts, ja que és el que té més fases i el que ha de comunicar a les altres parts quan s'ha de canviar d'etapa.

Com s'ha pogut veure al diagrama de blocs, es pot dividir el programa en dues grans etapes (*Training* i *DSC*), però dins de cada etapa hi ha subetapes que s'han convertit en funcions per a facilitar la comprensió del codi.

Tot seguit s'expliquen aquestes funcions en ordre seqüencial (en l'ordre que s'executen amb el pas del temps).

- **Programa:** és el cos principal del algoritme on es troben les dues principals etapes.

Algoritme de la funció Programa

```
Un cop s'inicia el programa entrem en un bucle infinit on es repeteixen les dues etapes principals.
while (b!=0)
   [M,V,N]=TrainingPhase;

   [ibit,w,wc,R]=DSCphase(M,V,N);
S'entra a la fase de descodificació i monitorització, agafem mostres durant I instants i repetim el bucle.
   for i=1:I
      [error_LWS,error_total,error_CE,error_totalc]=Monitoritzacio(ibit,w,wc,M);
   end
end
```

- ***Training Phase:*** Aquesta etapa engloba el compte dels sensors que hi ha a la xarxa i el recull de mostres per extreure més tard la matriu de correlacions.

Algoritme de la funció TrainingPhase

```
L'algoritme extreu el nombre de motes (M), el Training Phase Time (N) i la matriu de mostres (V).
M = NombreMotes;
N=2*M;
V = PeriodeProva (M,N);
Finalment s'envia el flag 0 per canviar la rutina d'enviament
EnvByte(0,0);
```

Tot seguit es veuran els algoritmes NombreMotes i PeriodeProva de forma breu ja que ja han estat explicats en l'apartat anterior.

- **NombreMotes:** És la funció que compta els sensors que hi ha a la xarxa.

Algoritme de la funció NombreMotes

Quan s'inicia el programa donem un valor a A segons mida esperada de la xarxa
for N=1:3*A
Es rep el paquet pel port sèrie (e), la posició 9 del vector és l'identificador del sensor, així tenim un control dels sensors que envien dades.

```
l = fread(s);
e = ConvPacket(l);
num_mota = e(9);
   if (num_mota != Motes)
      Motes(K) = num_mota;
      K=K++;
      M = length(Motes);
   end
end
```

Finalment retornem el valor M que és el nombre de motes que hi ha a la xarxa.

En aquest algoritme es troba la funció ConvPacket(), aquesta funció s'explica més endavant però bàsicament el que fa és extreure bytes que s'afegeixen al paquet per qüestions de coincidència amb els d'obertura i tancament d'aquest que poden fer rebre erròniament la mostra de llum o el identificador del *Sensor Node*.

- **PeriodeProva:** En aquesta etapa s'aniran emmagatzemant mostres durant un temps N.

Algoritme de la funció PeriodeProva

Es rep el paquet pel port sèrie (e) i es guarden les posicions 9 (id) , 15 i 16 (mostra de llum).
Es recullen mostres tants instants com creiem necessari, en aquest cas durant 2*M
for (N=1:2*(M^2))
  for (n=1:M)
Es recullen les mostres en ordre, així s'evita la presència d'un nou sensor fins la pròxima etapa *Training*

```
      while (num_mota!=n)
         l = fread(s);
         e = ConvPacket(l);
         num_mota = e(9);
      end
  end
```

Es converteix els dos bytes a binari per ordenar-los, i es tornen a decimal per emmagatzemar la dada.

```
   eb = dec2bin(e,8);
   f = [eb(16,1:8)];
   g = [eb(15,1:8)];
   E = [f g];
   V(num_mota,C) = bin2dec(E);
```

Es controla que la dada no valgui 0 per no tenir un error en el càlcul del *ibit*

```
   if V(n,N)==0
      V(n,N)=1;
   end
```

Cal destacar que la mostra de llum arriba amb el byte menys significatiu en la posició del més significatiu a causa del sistema operatiu que s'està utilitzant (Windows), per tant cal reordenar la mostra abans de guardar el valor en decimal.

- **DSCPhase:** En aquesta fase es calcula el *ibit* de cada sensor i s'envia. La part del DSC s'explica en l'apartat anterior, per tant tot seguit explicarem la funció d'enviament.

- **EnvByte:** Aquesta funció necessita dues entrades, l'*ibit* i l'identificador del sensor. Amb l'*ibit* es crea una màscara que serà la que s'enviarà al sensor per codificar la mostra. Si l'*ibit* és igual a 0, el que s'ha d'enviar són els dos bytes a 0 per avisar del canvi de rutina d'enviament.

Algoritme de la funció EnvByte

```
if (ibit==0)
   ibit_mask1=0;
   ibit_mask2=0;
else
   ibit_mask=[zeros(1,16-ibit) ones(1,ibit)];
   ibit_mask1=bi2de(ibit_mask(1,1:8));
   ibit_mask2=bi2de(ibit_mask(1,9:16));
end
paquet=[66;255;255;02;125;03;ibit_mask2;ibit_mask1;num_mota;];
[crc1,crc2] = CRC(paquet);
msg=[126 paquet' crc1 crc2 126];
```

Es comprova que cap byte no sigui igual a 126 o 125 per aplicar l'*escaped byte.*

```
for i=2:length(msg)-1
   if (msg(i)==125)
      S'afegeix un byte amb valor 0x7D XOR 0x20
   elseif (msg(i)==126)
      S'afegeix un byte amb valor 0x7E XOR 0x20
   end
end
```

S'envia el paquet pel port sèrie.

Es pot observar la complexitat del paquet que enviem en aquesta funció. Tot seguit s'explica amb més profunditat cada byte.

A la Figura 20 s'observa el paquet Y, ja anomenat anteriorment i que serveix per la comunicació *PC-Fusion Center* i per l'enviament del *flag* de canvi de rutina del *Fusion Center* als *Sensor Nodes*. Per altra banda, a la Figura 21 es veu l'estructura del paquet X que s'utilitza per a enviar les mostres de llum dels *Sensor Nodes* al *Fusion Center* i per enviar els *ibits* després mitjançant els dos bytes reservats per la constant.

Si s'observa l'encapsulat en ambdós paquets, es veu la estructura ja comentada al apartat on es parla dels paquets en TinyOS. Per tant ara es parlarà dels bytes no vistos en l'apartat comentat i es definiran.

		sens1	
1		126	Framing byte
2		66	packet type
3		125	First address byte
4		94	Escaped byte
5		0	second address byte
6		2	AM msg type
7		125	GroupID
8		93	Escaped byte
9		2	Data length
10	Payload	0	ibit
11		0	
12		1	node_id
13		93	CRC
14		112	
15		126	Framing Byte

Figura 20. Paquet Y

		sens1	sens2	
1		126	126	Framing byte
2		66	66	packet type
3		125	125	First address byte
4		94	94	Escaped byte
5		0	0	second address byte
6		1	1	AM msg type
7		125	125	GroupID
8		93	93	Escaped byte
9		12	12	Data length
10	Header	132	132	Board_id
11		2	2	packet_id
12		1	2	node_id
13		0	0	rsvd
14	Payload	0	0	vref
15		0	0	
16		0	0	therm.
17		0	0	
18		192	225	light
19		1	2	
20		132	132	const.
21		0	0	
22		250	138	CRC
23		4	155	
24		126	126	Framing Byte

Figura 21. Paquet X

- ***Framing Byte:*** Aquest byte indica l'inici i el final del paquet enviat. Per defecte el seu valor és 0x7E en hexadecimal (126 en decimal).

- ***Packet Type:*** No confondre amb *AM msg type* el qual ens indica amb quina interfície es podrà rebre el paquet. Aquest byte indica si el paquet utilitza ACK[5], és a dir, reconeixement de recepció. Els bytes que poden anar en aquest camp i el seu significat s'expliquen en la Taula 1:

Etiqueta del Byte	Byte	Descripció
P_PACKET_NO_ACK	0x42 (66)	No hi ha ACK
P_PACKET_ACK	0x41 (65)	Hi ha ACK, inclou un byte de prefix, el receptor ha de posar aquest prefix a la resposta per saber que ha arribat bé.
P_ACK	0x40 (64)	La resposta al P_PACKET_ACK, ha d'incloure el prefix rebut.
P_UNKNOWN	0xFF (255)	Tipus de paquet desconegut

Taula 1. Tipus de *Packet Type Byte*

- ***Address Bytes:*** Aquests bytes s'han anomenat al apartat de TinyOS però convé fer una lectura més extensa dels valors que pot assolir per a les diferents adreces possibles, les quals es poden observar a la

Etiqueta dels Bytes	Byte	Descripció
Broadcast Address	0xFFFF (65535)	Enviament a tots els nodes de la xarxa possibles.
UART Address	0x007E (126)	Enviament a través del port sèrie.
Node Address	Node_id	Enviament a un node en particular.

Taula 2. Tipus de *Address Bytes*

Tot seguit explicarem com calcular els dos bytes corresponents al CRC i el seu significat.

- ***CRC:*** El CRC o Control de Redundància Cíclica és un sistema de detecció d'errors basat en un polinomi generador (el mateix en l'emissor i el receptor). La forma d'extreure el CRC radica en multiplicar la paraula a codificar per l'ordre del polinomi generador, després es divideix el resultat pel polinomi generador i s'aconsegueix un residu. Finalment, la paraula a transmetre serà la paraula inicial seguida del residu, que serà el CRC, i tindrà una allargada igual al ordre del polinomi generador.

[5] Missatge que s'envia per confirmar que el paquet ha arribat.

En el nostre cas el polinomi generador és 0x1021 ($x^{16} + x^{12} + x^5 + 1$), l'últim bit ($x^{16}$) s'afegeix per a que el polinomi sigui d'ordre 16 i per tant tinguem un CRC de dos bytes.

S'ha de ressaltar que al càlcul del CRC no hi entra els *Framing Byte*, per tant, el càlcul englobarà des del *Packet Byte* fins l'últim byte del *Payload*.

La funció CRC necessita com a variable d'entrada el missatge en decimal i fa el càlcul dels dos bytes tal com s'ha explicat i els torna com a sortida.

- ***Escaped Byte:*** Byte que s'ha d'afegir quan un dels bytes dins del paquet és igual al *Framing Byte* (126) o *Framing Byte*-1 (125). Per a calcular el valor d'aquest byte s'ha de fer una X-OR entre el byte repetit i el valor 0x20 en hexadecimal (32 en decimal) i afegir el byte resultant després del repetit. Si el repetit és igual a 126, s'haurà de posar un 125.

Aquest *Escaped Byte* afecta a tots els bytes dins del paquet, inclòs el CRC.

- ***Monitorització:*** En aquesta etapa s'engloba la descodificació de les mostres rebudes i la monitorització de les diferents variables que es volen controlar, com l'error del filtre de Wiener i l'error total, la compressió de les mostres i l'error d'altres sistemes d'estimació que es poden utilitzar.

Aquesta etapa serà la protagonista en el següent apartat on s'explicarà extensament, comentant els resultats i els sistemes monitoritzats. Per tant, en aquest apartat tan sols s'explica la funció de descodificació o *DSD*.

- ***DSD:*** La funció *DSD* o *Distributed Source Decoding* ens descodifica la mostra codificada pel *DSC*. S'utilitza el sistema vist a la Figura 15 per trobar la posició de la nostra mostra codificada.

Un cop trobada la posició, es busca en un altre vector on es trobin els valors reals i agafar el valor que hi ha a la posició escollida. Com que normalment el valor quedarà entre dues posicions, s'utilitza el valor estimat *y* per a saber quin és el més proper.

Algoritme de la funció DSD

```
pos = 1:2^16;
c1=1:2^16;
for jj = 0:ibit
   ii=16;
   if x_cod_bin(ii-jj)==0
```

```
      pos = pos(1:2:end);
   else
      pos = pos(2:2:end);
   end
end
[nothing, pos_dec] = sort(abs(y-c1(pos)));
x_dec = c1(pos(pos_dec(1));
```

Altres Funcions

S'utilitzen altres funcions que no són fonamentals però que faciliten molt el procediment de l'algoritme:

- ***Restart:*** s'introdueix el nombre de sensors i "reseteja" l'*ibit* de tots aquests per a tornar a la fase de *Training* mitjançant la funció EnvByte**.**
- ***Plots*****:** Funció que dibuixa els plots de la monitorització.
- ***ConvPacket:*** Un cop rebut el paquet, el transforma per obviar els valor alterats per l'*escaped byte*, d'aquesta manera es preveuen els possibles errors que poden sorgir al haver-hi aquesta modificació en el missatge.

Algoritme de la funció ConvPacket

```
j=1;
for i=2:length(msg)-1
      if(msg(i)==125)
        msg_conv(j)=bitxor(msg(i+1),32);
        msg=[msg(1:i) msg(i+2:end) 0];
      else
        msg_conv(j)=msg(i);
      end
      j=j++;
 end
```

5.2.2. Sensor Node

Els *Sensor Nodes* estan programats seguint el diagrama de blocs observat anteriorment. El programa té el format que s'utilitza en TinyOS però en aquest cas s'ordenarà el més seqüencialment possible per a facilitar la comprensió.

Algoritme dels *Sensor Nodes*

L'algoritme es basa en el *Timer*, quan aquest s'activa i el *flag* és 1, s'activa la recollida de la mostra de llum.

```
event result_t Timer.fired(){
   if (flag==TRUE){
```

```
      call PhotoControl.start();
      call Light.getData();
   }
return SUCCESS;
 }
```

Un cop es té la mostra de llum, s'empaqueta amb dependència al valor del *flag2*, si aquest és 0 voldrà dir que no es codifica la mostra i enviem la dada sense modificar, si el *flag2* és 1, s'ha de modificar la mostra, per tant fem una AND lògica amb la màscara del *ibit*. Finalment s'envia.

```
async event result_t Light.dataReady(uint16_t data) {
        atomic llum=data;
        if (flag2==0){
           atomic pack->xData.datap1.light = llum;
        }
        else{
           atomic pack->xData.datap1.light = llum&ibit;
        }
        post SendData();
return SUCCESS;
 }
```

Es tenen dues interfícies de recepció de missatges, una pel canvi de rutina que canviarà el valor del *flag* un cop rebem, i l'altra per la recepció de l'*ibit*, que ens canviarà el *flag2* quan aquest sigui igual a 16 (màscara 0xFFFF).

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m){
        YDataMsg *payload;

        payload = (YDataMsg*)(m->data);
        F = payload->yData.datap2.ibit;
        if (F == 0) {
                atomic flag=!(flag||0);
        }
return m;
}
event TOS_MsgPtr ReceiveIbit.receive(TOS_MsgPtr msg){
        XDataMsg *payload;

        payload = (XDataMsg*)(msg->data);
        atomic ibit = payload->xData.datap1.constant;

        if (ibit==0xFFFF){
           atomic flag2=0;
        }else{
           atomic flag2=1;
        }
return msg;
}
```

5.2.3. Fusion Center

Tot seguit es comenta l'algoritme programat en el *Fusion Center* que com el dels *Sensor Nodes* correspondrà amb el diagrama de blocs abans esmenat.

Algoritme del Fusion Center

EL Fusion Center no fa res fins que no rep un paquet, en aquest cas d'algun Sensor Node, llavors empaqueta els valors importants i els reenvia pel port sèrie.

```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr msg){
        XDataMsg *payload;

        payload = (XDataMsg*)(msg->data);
        llum = payload->xData.datap1.light;
        constant = payload->xData.datap1.constant;
        port = payload->xSensorHeader.node_id;
        atomic pack->xData.datap1.light = llum;
        atomic pack->xData.datap1.constant=constant;
        atomic pack->xSensorHeader.node_id=port;

        post SendData();//envia dada al port sèrie
return msg;
}
void task SendData(){
        atomic sending_packet = TRUE;
        if (call UARTMsg.send(TOS_UART_ADDR,sizeof(XDataMsg),&msg_buffer) !=
SUCCESS)
                sending_packet = FALSE;
return;
}
```

També es pot rebre pel port sèrie un paquet, en aquest cas comprovarem el que es rep i es farà la acció corresponent, si és un 0 s'envia un flag a tots els nodes per canviar la rutina d'enviament i si és diferent de 0 s'envia el valor del ibit al sensor que li pertoca.

```
event TOS_MsgPtr ReceiveUART.receive(TOS_MsgPtr m){
        YDataMsg *payload;

        payload = (YDataMsg*)(m->data);
        ibit = payload->yData.datap2.ibit;
        port = payload->yData.datap2.node_id;

        atomic pack->xData.datap1.constant=ibit;
        if (ibit == 0) {
        post SendFlag();
        }else{
        post IbitMsg();
```

```
        }
return m;
}
void task SendFlag(){
atomic sending_packet = TRUE;
        if (call SendMsg.send(TOS_BCAST_ADDR,sizeof(YDataMsg),&msg_buffer2) !=
SUCCESS)
            sending_packet = FALSE;
return;
}
void task IbitMsg(){
        atomic sending_packet = TRUE;
        if (call UARTMsg.send(port,sizeof(XDataMsg),&msg_buffer) != SUCCESS)
          sending_packet = FALSE;
return;
}
```

En aquest algoritme es poden veure les tres maneres d'enviar que es comenten en l'apartat de TinyOS explicat anteriorment, la *Broadcast*, la *UART* i la expressa a un node.

Cal comentar que els algoritmes presentats són resums i pseudocodis (en el cas dels del *PC*) i que en cap cas compilarien[6].

Finalment és necessari fer una explicació de com donar format als paquets en els dos programes i que complementi la Figura 20 i la Figura 21.

El format del paquet s'inclou en un arxiu anomenat sensorboardApp.h en el qual s'indica l'estructura del o dels paquets i algunes constants globals com els paràmetres que s'afegeixen a les interfícies paramètriques. Seguidament es veurà el format del Paquet X que és el més complex.

Definició del Paquet X a sensorboardApp.h

```
typedef struct XSensorHeader{
  uint8_t  board_id;
  uint8_t  packet_id; // 2
  uint8_t  node_id;
  uint8_t  rsvd;
}__attribute__ ((packed)) XSensorHeader;

typedef struct PData1 {
  uint16_t vref;
```

[6] Els codis del Fusion Center i els Sensor Nodes es troben al Annex A i els de l'algoritme al Annex B.

```
  uint16_t thermistor;
  uint16_t light;
  uint16_t constant;
} __attribute__ ((packed)) PData1;

typedef struct XDataMsg {
  XSensorHeader xSensorHeader;
  union {
  PData1    datap1;
  }xData;
} __attribute__ ((packed)) XDataMsg;

enum {
  AM_XSXMSG = 0,
  AM_XSYMSG = 1,
  AM_XSXUART = 2,
};
```

Com es pot observar, es creen dues estructures que més tard formaran el paquet sencer a enviar. Aquestes estructures són el *Header* on hi ha la informació del sensor i el *Payload* on hi ha els valors de llum, temperatura i alguna constant que es vulgui afegir.

Aquesta manera de declarar l'estructura d'un paquet comporta un problema per a la finalitat del *ibit*, ja que no es podrà retallar la mostra de llum a causa de que està declarada com un enter de 16 bits. Per aquesta raó s'utilitza l'emmascarament per simular l'efecte de la compressió del *ibit*. Per tant es seguiran enviant 16 bits però els no retallats pel valor del *ibit* s'enviaran a 0.

Per últim s'enumeren tres constants que són les que indiquen l'*AM packet type* el qual ens indicarà quin tipus de paquet enviarà o podrà rebre cada interfície.

6. Resultats

6.1. Monitorització

Tot seguit s'exposen diverses gràfiques per a comparar i extreure resultats sobre l'error del filtre de Wiener, l'error total del algoritme i la compressió afegida per aquest. També es fa un estudi de les etapes *Training* i de recollida de mostres.

Per a dur a terme aquestes monitoritzacions, s'intenta utilitzar un escenari comú. Aquest es mostra a la Figura 22.

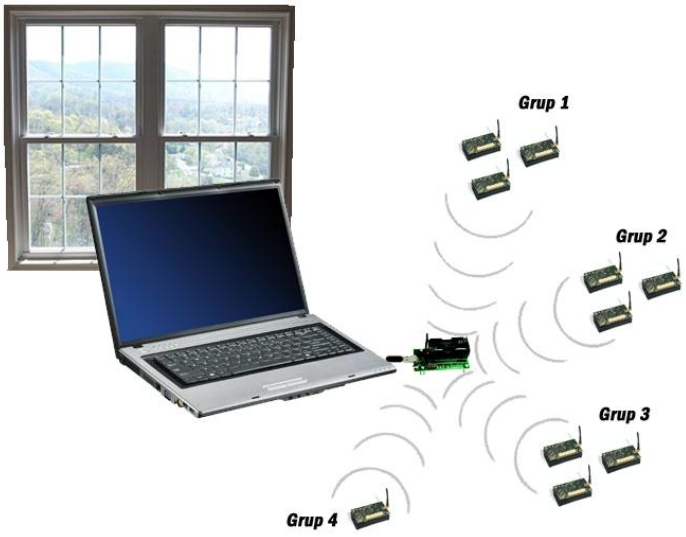


Figura 22. Escenari amb 10 sensors i llum ambient

Com s'observa, si la xarxa és de 10 sensors, es formen 4 grups de sensors.

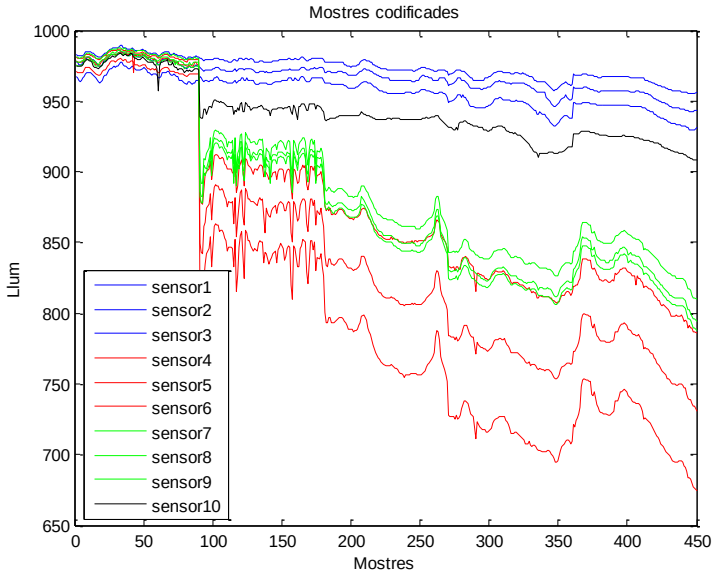


Figura 23. Distribució de mostres codificades que envien els 10 sensors

Un exemple de la distribució de llum que reben es veu a la Figura 23, on durant els primers instants els sensors estan junts i després es separen per així veure la diferència lumínica.

En alguns casos, la variabilitat que introdueix la llum ambient no interessa per extreure alguns resultats, per això el que es farà és utilitzar una llum artificial (una làmpada) per aconseguir que aquesta llum sigui constant

També en alguns escenaris es redueix el nombre de sensors de la xarxa per observar els canvis que pot provocar això als errors monitoritzats.

6.1.1. Estudi del error del filtre de Wiener respecte l'error total del sistema

Primerament s'observarà l'error introduït pel filtre de Wiener fent la resta entre el valor real de la mostra i el valor estimat pel filtre:

$$\hat{y}_{Wiener} = \boldsymbol{w}^H \cdot \boldsymbol{x_{cod}}$$

$$e_{Wiener} = |x_{real} - \hat{y}_{Wiener}|$$

Aquest valor es compara amb l'error total del sistema, el qual s'extreu de la resta entre el valor real i el valor descodificat:

$$e_{total} = |x_{real} - x_{desc}|$$

Per a cada instant, es fa la mitja dels errors calculats de cada sensor. Aquest valor pot semblar molt gran a primera vista i per això es representa l'error relatiu, el qual es calcula de la següent manera:

$$e_{relatiu} = \frac{e}{\sqrt{\sigma^2_{mitj}}}$$

On *e* és l'error Wiener o total i σ^2_{mitj}és la mitja de les potències de senyal, per tant, la mitja de la diagonal de la matriu de correlacions **R**.

Per tots els següents escenaris, la etapa *Training* és de 2M instants, on M és el nombre de sensors a la xarxa i l'interval entre instants és de 20 segons.

#Escenari	#Sensors	Training Time [instants]	DSC Time [instants]	Iteracions	T. Totals [h]	Llum
1.1[Figura 24]	10	20	45	32	8	Ambient
1.2[Figura 25]	5	10	90	3	1.30	Ambient
1.3[Figura 26]	10	20	45	2	1.30	Ambient
1.4[Figura 27]	5	10	90	3	1.30	Artificial

Taula 3. Escenaris E.Wiener vs. E.Total

1r escenari: recollida de mostres de llum de 10 sensors durant 8 hores amb etapes de recollida de mostres de 45 instants de temps amb intervals de 20 segons. En total unes 32 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. La línia gruixuda a la Figura 24 ens determina la mitja d'error durant aquesta durada.

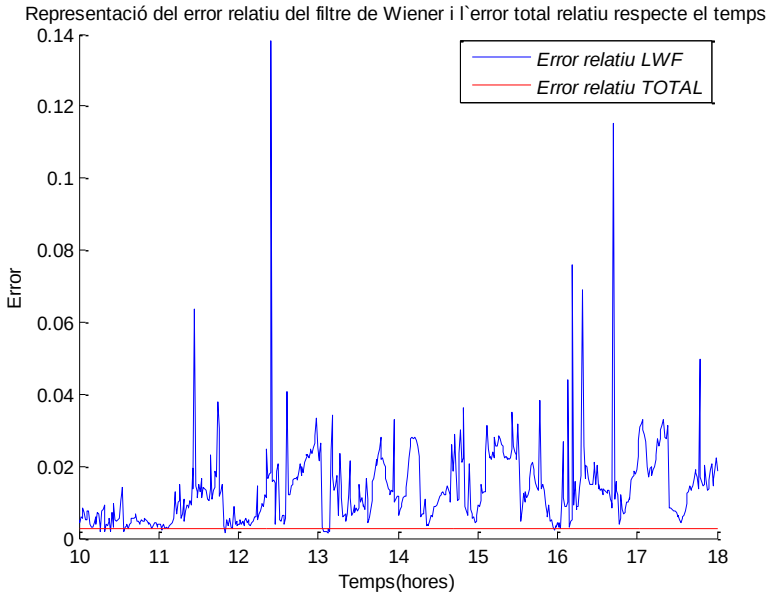


Figura 24. Error relatiu del filtre de Wiener vs. Error relatiu total del escenari 1.1

2n escenari: recollida de mostres de llum de 5 sensors durant 1.30 hores amb etapes de recollida de mostres 90 instants de temps amb intervals de 20 segons. En total unes 3 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. En aquest cas observem a la Figura 25 que l'error mitjà és el doble que l'anterior, aquest resultat es pot explicar per l'augment d'instants en els quals estem recollint mostres, ja que al augmentar aquest temps, augmenta la probabilitat de que la correlació entre sensors variï i per tant ens introdueix més error. En aquest cas la llum mesurada es la llum ambient, per tant la variació d'aquesta en 90 instants de temps ens produirà un error superior a quan es mesuraven 45 instants.

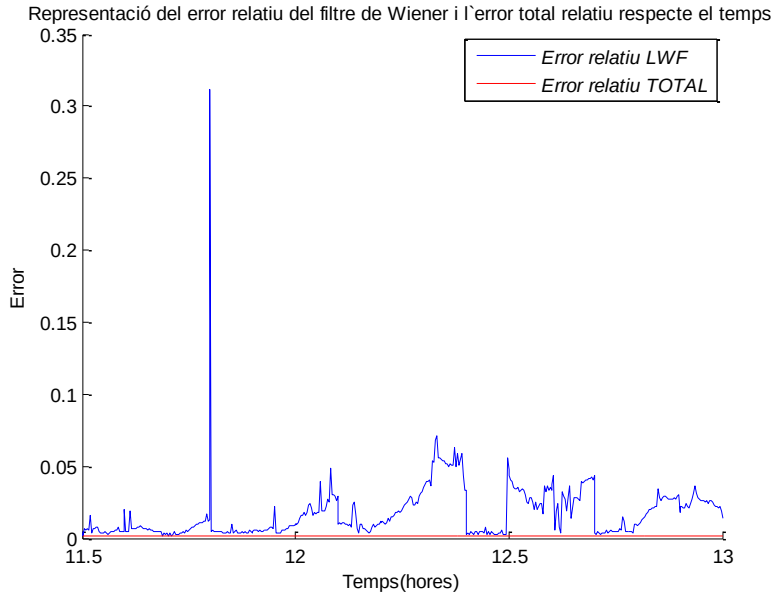


Figura 25. Error relatiu del filtre de Wiener vs. Error relatiu total del escenari 1.2

3r escenari: recollida de mostres de llum de 10 sensors durant 1.40 hores amb etapes de recollida de mostres de 120 instants de temps amb intervals de 20 segons. En total unes 2 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. Com s'observa a la Figura 26 l'error torna a créixer si s'augmenta més el temps de recollida. A més un augment del valor de les mostres durant la primera iteració fa que l'error sigui molt gran i es corregeix al inici de la segona etapa, com es pot comprovar amb la discontinuïtat observada a la Figura 26.

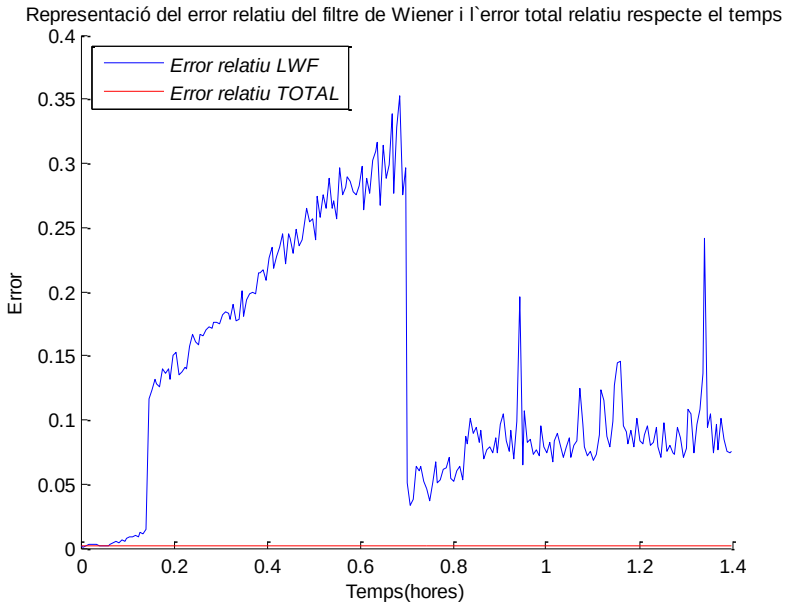


Figura 26. Error relatiu del filtre de Wiener vs. Error relatiu total del escenari 1.3

4t escenari: recollida de mostres de llum artificial (constant al llarg del temps) de 5 sensors durant poc menys de hora i mitja en etapes de recollida de mostres de 90 instants de temps amb intervals de 20 segons. En total unes 3 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. En aquest cas el valor relatiu del error, com observem a la Figura 27, disminueix un ordre respecte el primer escenari tot hi doblar els instants de recollida de mostres. Això és degut a que les dades de llum no varien, són gairebé constants, i per tant no es té l'error afegit de que la matriu de correlacions quedi obsoleta. Per tant es pot augmentar el temps de recollida que si el valor es gairebé constant, l'error serà el mateix.

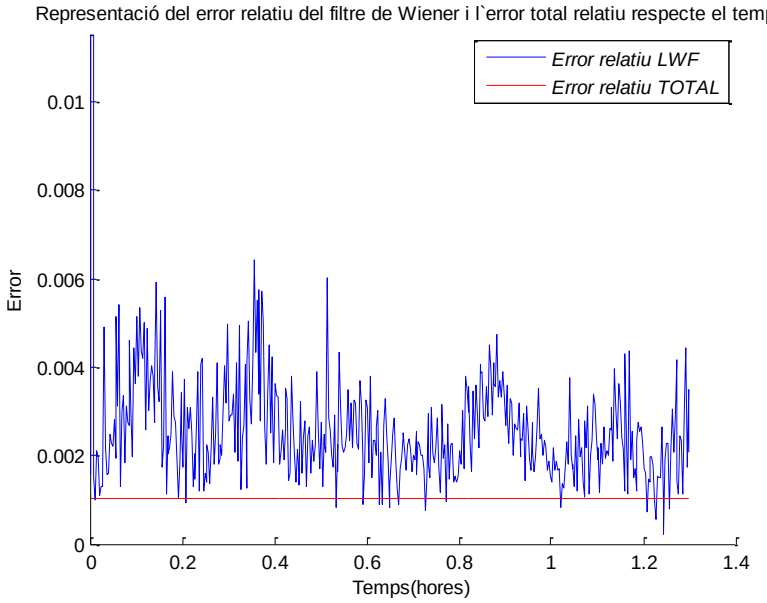


Figura 27. Error relatiu del filtre de Wiener vs. Error relatiu total del escenari 1.4

Vistos els quatre escenaris s'observa que sempre el valor total d'error tot i no ser zero, és sempre molt baix, per tant tenim una descodificació quasi perfecta. A partir de les quatre figures es pot arribar a la conclusió que com major sigui el temps de recollida de mostres major serà l'error per mostres variables en el temps i que amb un temps de *Training* òptim es tenen uns valors estimats molt propers als reals, els quals ens poden indicar quan és necessari tornar a fer una mesura de la matriu de correlacions. Per tant, el següent a observar seran els temps de *Training* i de recollida de mostres.

6.1.2. Determinació del temps *Training*

La determinació del temps que emprarem en l'etapa *Training* és important ja que aquesta etapa consumirà gran part de l'energia del sensor, tenint en compte que durant aquesta etapa s'envia la mostra sencera i no codificada.

Tot seguit es comparen 4 escenaris on s'ha augmentat el *Training Time* per observar els resultats obtinguts. Els intervals entre instants segueixen sent de 20 segons.

#Escenari	#Sensors	Training Time [instants]	DSC Time [instants]	Iteracions	T. Totals [h]	Llum
2.1[Figura 28]	10	20	45	16	4	Ambient
2.2[Figura 30]	10	40	100	3	1.30	Ambient
2.3[Figura 32]	10	60	100	8	4	Ambient
2.4[Figura 34]	10	80	120	5	4	Ambient

Taula 4. Escenaris determinació del temps *Training*

1r escenari: Mostres recollides de 10 sensors des de les 14.00 hores fins les 18.00 hores. A la Figura 28 es veu l'error relatiu del filtre de Wiener, el qual aumenta considerablement a partir de les quatre i mitja. Si s'observa la Figura 29 on es representen les mostres de llum capturades, a aquesta hora hi ha una disminució considerable de la llum detectada pels sensors 1 i 10. Aquest canvi brusc ha provocat un augment del error de l'estimació, però l'error total no s'ha vist afectat.

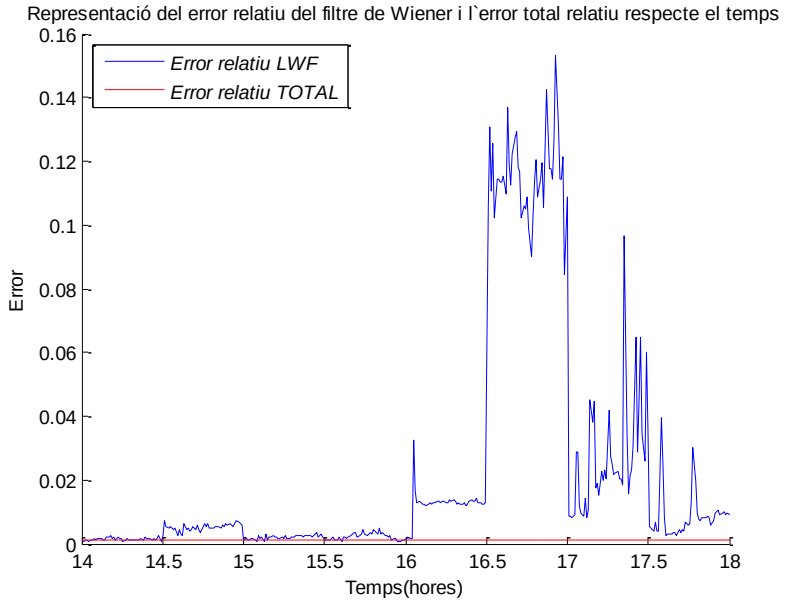


Figura 28. Error relatiu amb un Training Time de 2*M

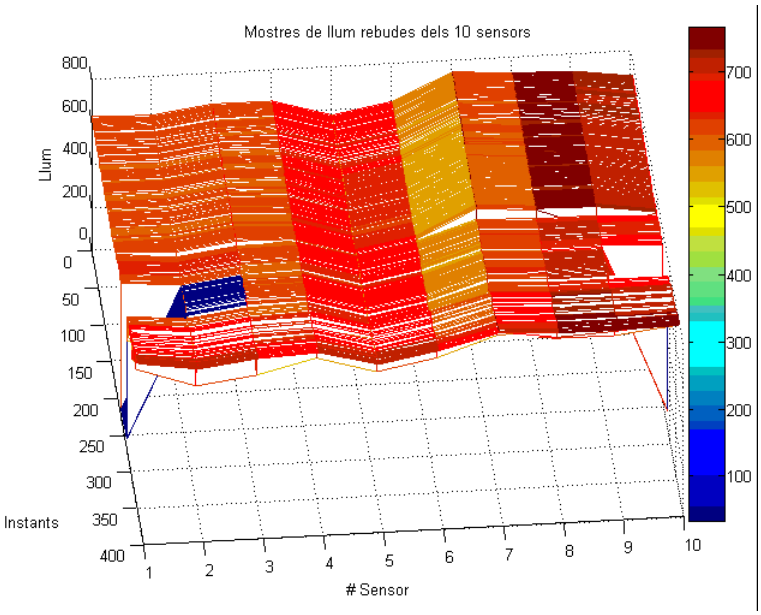


Figura 29. Mostres de llum rebudes a l'escenari 2.1

2n escenari: Mostres recollides de 10 sensors amb llum ambient al final del dia durant 100 instants de temps amb un període *Training* de 40 instants (4*M). Es pot veure a la Figura 30 com l'error relatiu és petit al principi de cada iteració però creix a mida que passa el temps, això ens indica que la matriu de correlacions comença a quedar-se obsoleta. En l'última etapa, les mostres són més constants, com es pot veure a la Figura 31, i per aquesta raó l'error gairebé no pateix grans variacions.

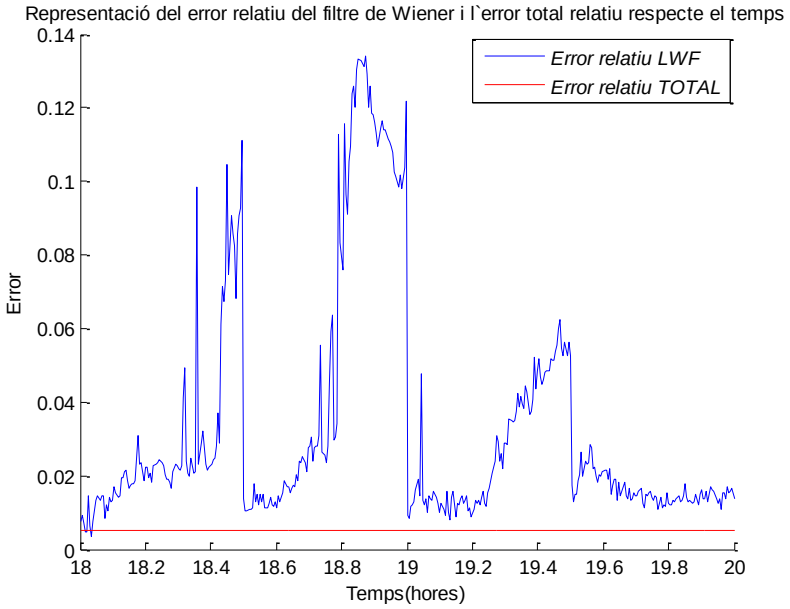


Figura 30. Error relatiu amb un Training Time de 4*M

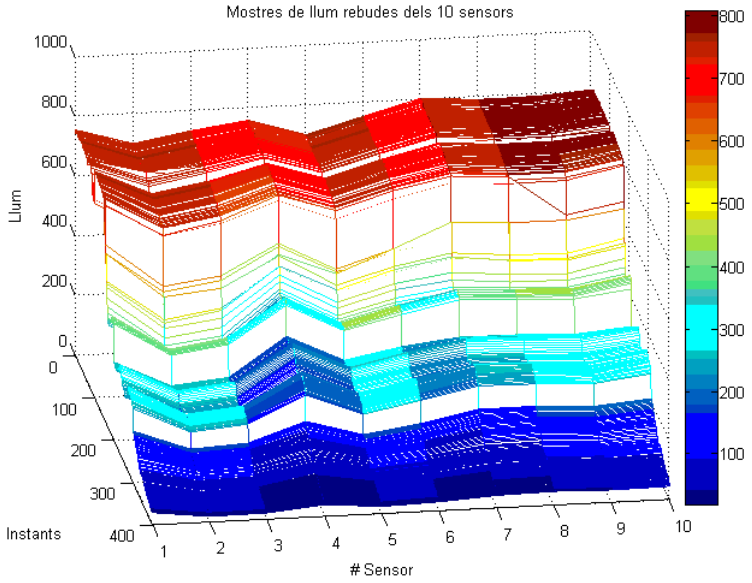


Figura 31. Mostres de llum rebudes a l'escenari 2.2

3r escenari: Mostres recollides de 10 sensors amb llum ambient durant 100 instants de temps amb un període *Training* de 60 instants (6*M). En aquest cas, es pot observar que l'error en algunes iteracions es manté constant, però en cap cas es millora l'error obtingut en el primer escenari, pel que tenim un resultat semblant però gastant el triple d'energia durant l'etapa *Training*.

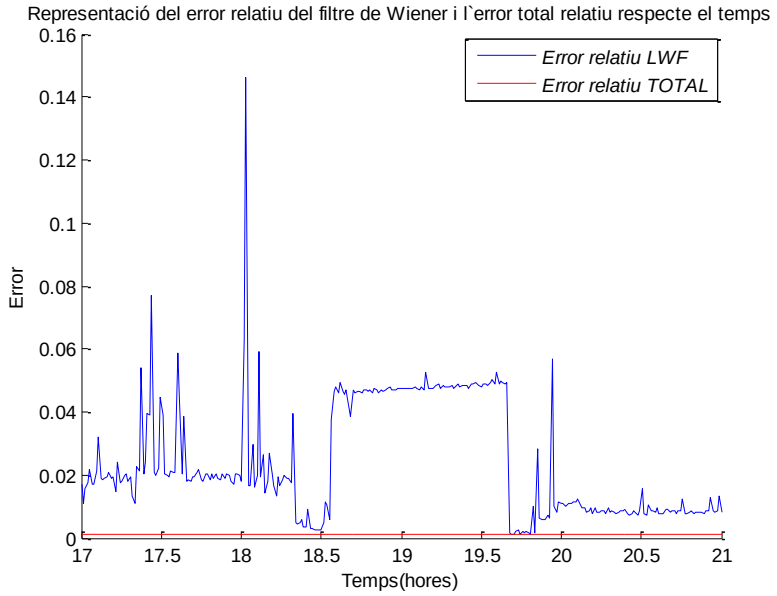


Figura 32. Error relatiu amb un Training Time de 6*M

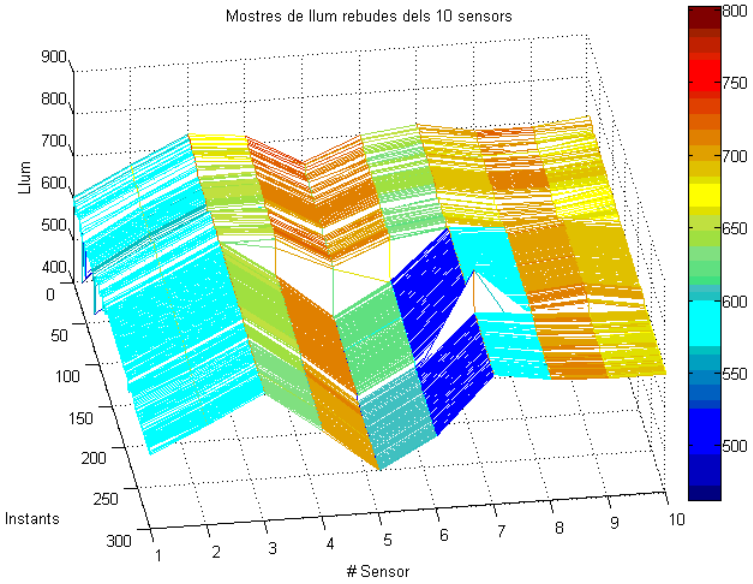


Figura 33. Mostres de llum rebudes a l'escenari 2.3

4t escenari: Mostres recollides de 10 sensors amb llum ambient decreixent (tarda-nit) durant 120 instants de temps amb un període *Training* de 80 instants (8*M). Finalment aquest cas confirma tot el que s'ha suposat. Per mostres de llum petites i força constants l'error del filtre de Wiener es manté petit, però quan hi ha un cert canvi en la llum, l'error augmenta considerablement ja que l'etapa de recollida de mostres es fa més gran per compensar que l'etapa de *Training* sigui més llarga. Les tres iteracions es veuen clarament en la Figura 34 i les variacions de la llum recollida a la Figura 35.

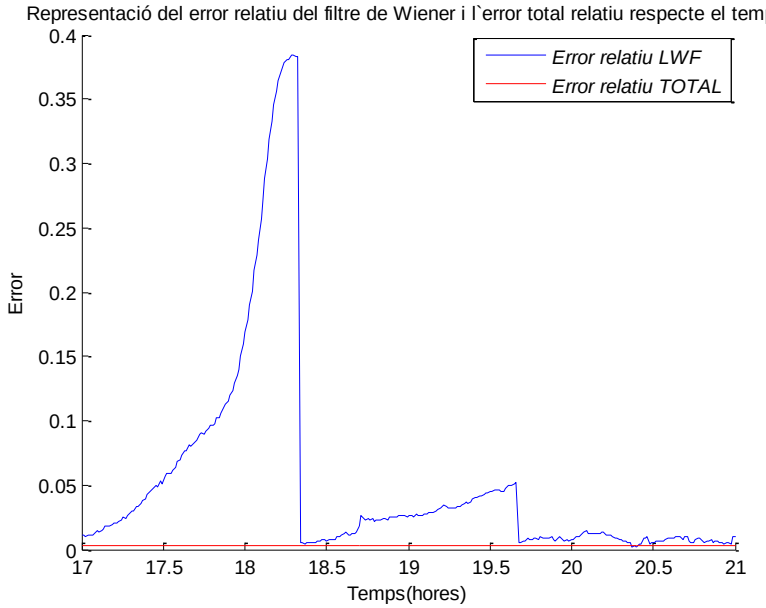


Figura 34. Error relatiu amb un Training Time de 8*M

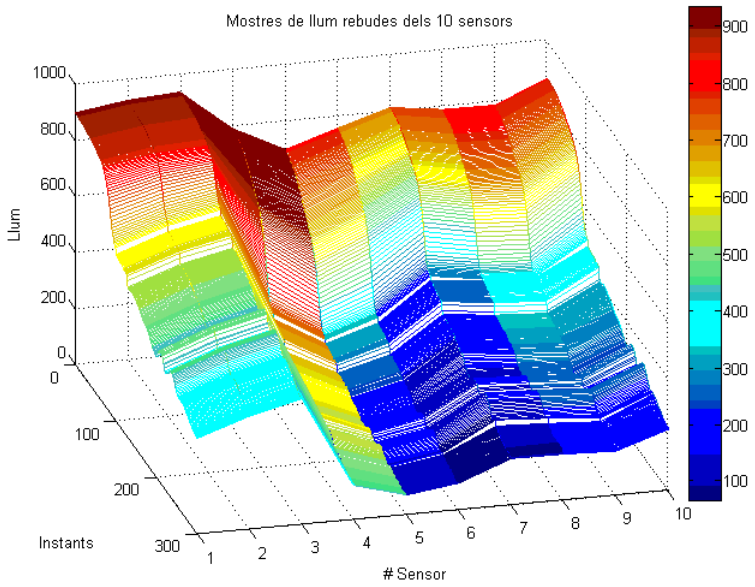


Figura 35. Mostres de llum rebudes a l'escenari 2.4

Per tant augmentar l'etapa *Training* no aporta cap millora al nostre sistema. La matriu formada amb 2M instants és prou precisa ja que els valors estan bastant correlats i fer una etapa més llarga només augmenta el cost energètic del sistema.

6.1.3. Determinació del temps de recollida de mostres codificades

El temps de recollida de mostres és molt important com ja s'ha comentat anteriorment, ja que tot depèn dels valors recollits en l'etapa de *Training* i les vegades que s'actualitzin aquests valors dependrà directament del temps que estem recollint dades. Per tant és important observar la variació del error a mida que es varia aquest temps i intentar buscar una solució òptima.

Per tots els següents escenaris, la etapa *Training* és de 2M instants, on M és el nombre de sensors a la xarxa i l'interval entre instants és de 20 segons.

#Escenari	#Sensors	Training Time [instants]	DSC Time [instants]	Iteracions	T. Totals [h]	Llum
3.1[Figura 36]	10	20	45	8	2	Ambient
3.2[Figura 38]	10	20	90	4	2	Ambient
3.3[Figura 40]	10	20	120	2	2	Ambient
3.4[Figura 42]	10	20	120	2	2	Artificial

Taula 5. Escenaris determinació temps de recollida de mostres

1r escenari: recollida de mostres de llum de 10 sensors durant prop de 2 hores en etapes de recollida de mostres de 45 instants de temps amb intervals de 20 segons. En total unes 8 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. A la Figura

36 es pot observar un error relatiu baix dins dels que s'han observat en apartats anteriors.

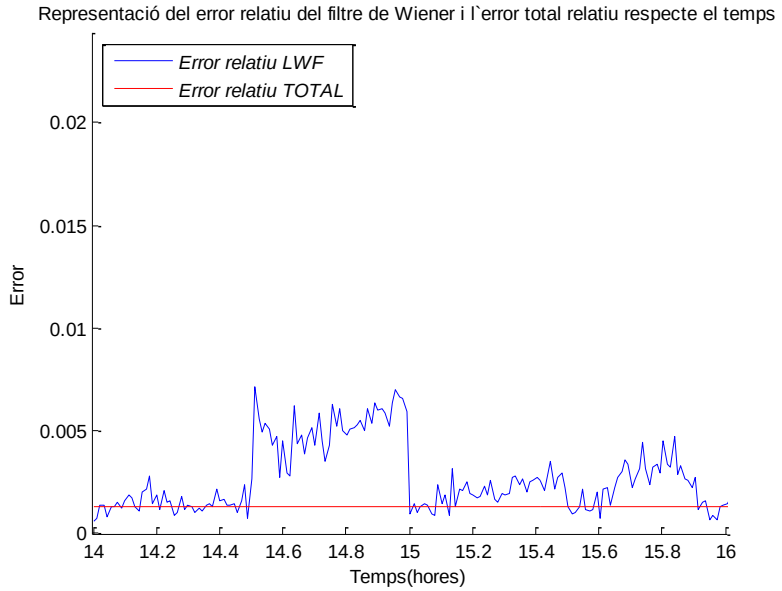


Figura 36. Error relatiu amb 45 instants de recollida de mostres

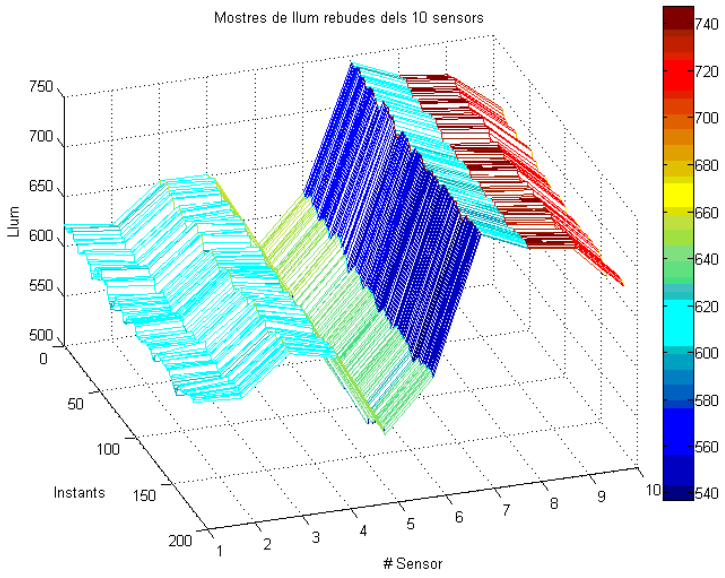


Figura 37. Mostres de llum rebudes a l'escenari 3.1

2n escenari: recollida de mostres de llum de 10 sensors de les 18.00 hores a les 20.00 hores en etapes de 90 instants de temps amb intervals de 20 segons. En total unes 4 iteracions de l'etapa *Training* i l'etapa de recollida de mostres. Observant la Figura 38 i laFigura 39 es pot veure com al principi quan tots els sensors mesuren la mateixa llum l'error del filtre és molt petit, però quan comença a haver una variació augmenta. Tot

hi això, els marges on varia l'error són molt petits i en cap cas importants, ja que són com a màxim d'un 1%.

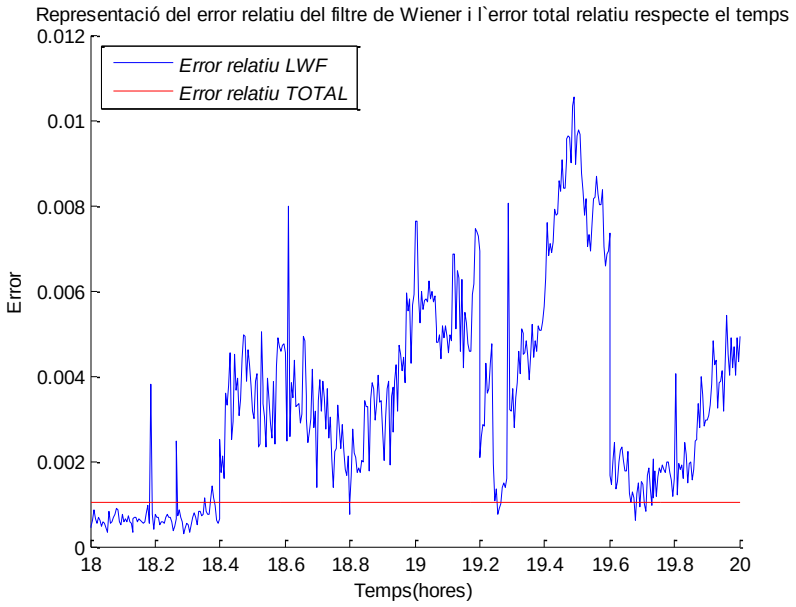


Figura 38. Error relatiu amb 90 instants de recollida de mostres

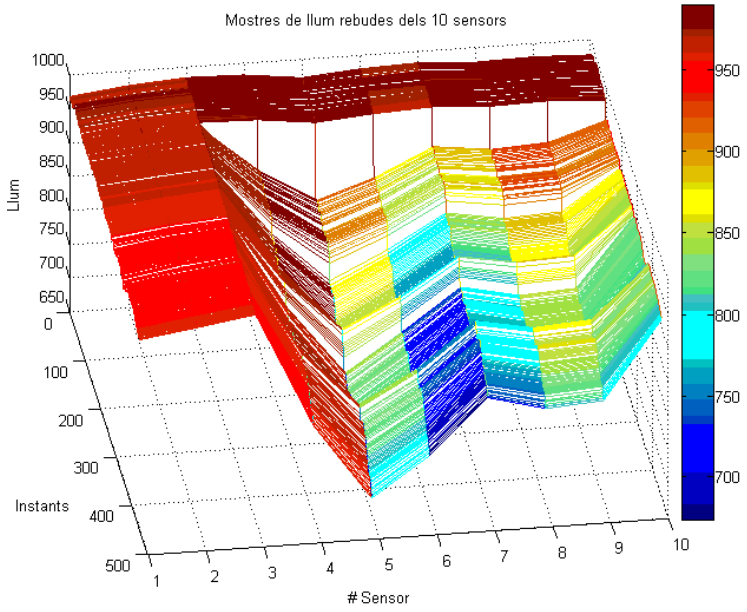


Figura 39. Mostres de llum rebudes a l'escenari 3.2

3r escenari: recollida de mostres de llum de 10 sensors de les 18.00 hores a les 20.00 hores en etapes de 120 instants de temps amb intervals de 20 segons. En aquest cas es veu clarament on acaba la primera iteració i comença la següent a la Figura 40 (el mateix cas que a la Figura 27) ja que l'error acumulat és gran. Al haver una gran variació de llum (veure la Figura 41) durant l'etapa de recollida de mostres, la matriu de correlació es queda obsoleta i és necessari l'actualització d'aquesta.

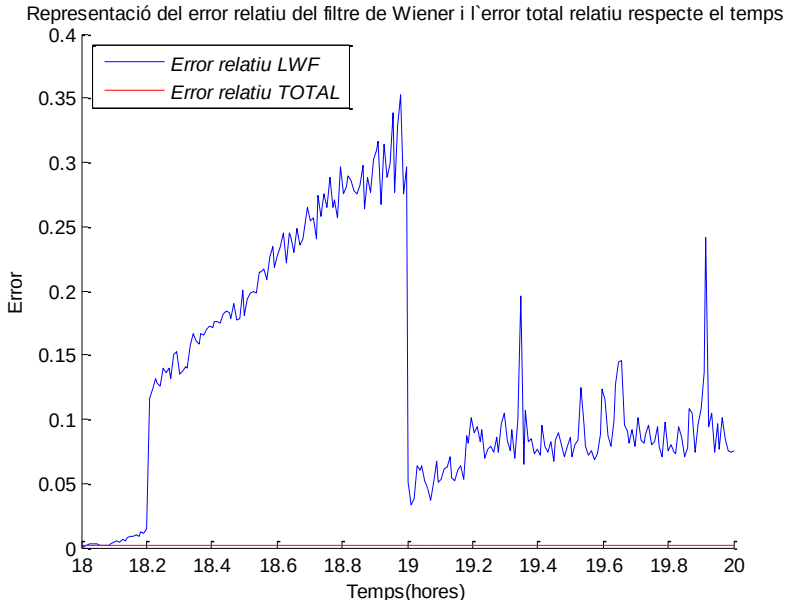


Figura 40. Error relatiu amb 120 instants de recollida de mostres

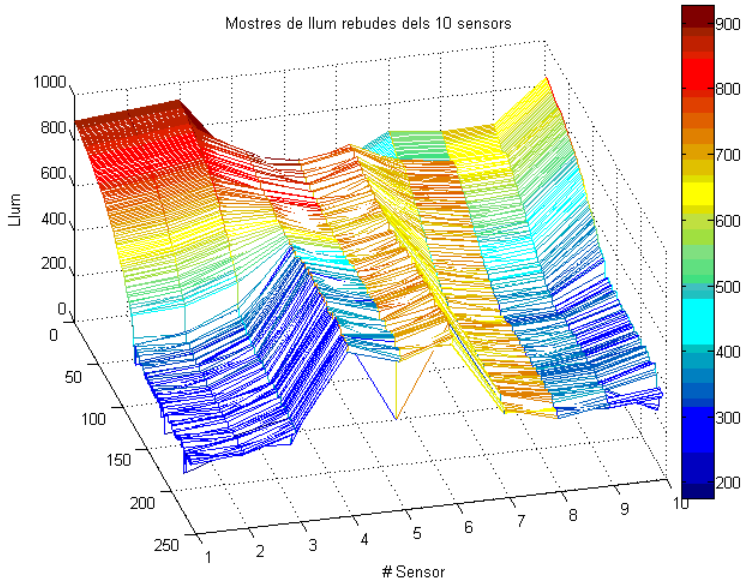


Figura 41. Mostres de llum rebudes a l'escenari 3.3

4t escenari: recollida de mostres de llum de 10 sensors durant 2 hores amb llum artificial en etapes de recollida de mostres 120 instants de temps amb intervals de 20 segons. Els resultats es troben a la Figura 42 i la Figura 43. L'error del filtre de Wiener torna a ser molt petit per aquest cas (no arriba al 0.7%) ja que les mostres de llum són constants en el temps.

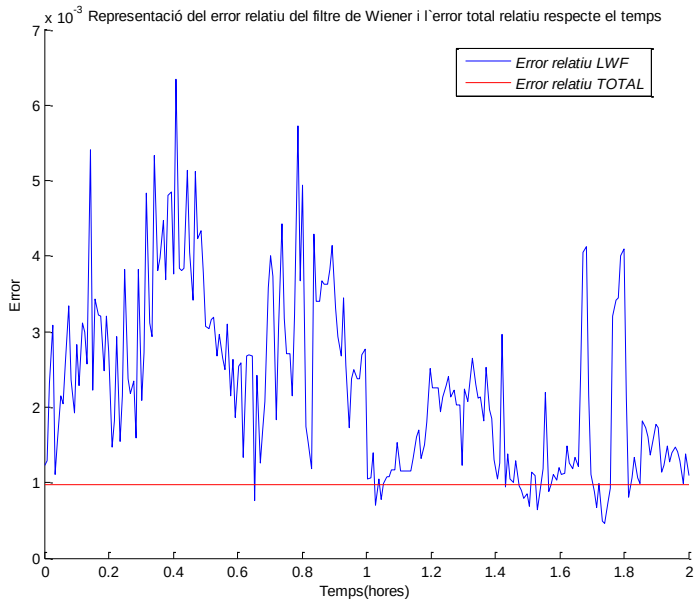


Figura 42. Error relatiu amb 120 instants de recollida de mostres

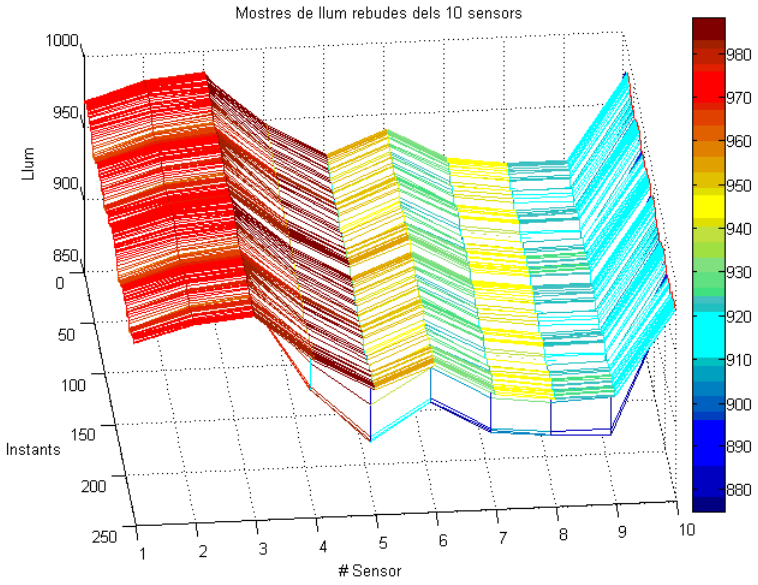


Figura 43. Mostres de llum rebudes a l'escenari 3.4

Com a resultat de l'observació de les anteriors gràfiques es pot dir que el temps de recollida de mostres dependrà directament de la variació dels valors recollits. Si aquestes mostres pateixen una variació greu, incrementarà fortament l'error però si les dades es mantenen en valors força constants, es podran produir iteracions molt llargues entre la recollida i el període de *Training*.

Per tant, hi ha dues opcions per a determinar cada quant s'ha de produir un període de *Training.* La més senzilla de veure és realitzar un cert estudi de a quines hores del dia es produeixen més canvis en els valors controlats i incrementar les iteracions en

aquest temps. Una alternativa més complicada però molt més efectiva és monitoritzar l'error del filtre de Wiener, que com ja s'ha vist, es bastant petit mentre la matriu de correlacions es vagi actualitzant i d'aquesta manera quan l'error fos superior a un cert nivell es cridaria un altre cop l'etapa *Training*. El problema d'aquesta alternativa és que per a calcular aquest error es necessita una mostra real i, per tant, algun dels sensors que no fos el primer hauria d'enviar la seva dada sense codificar amb el conseqüent augment de consum energètic.

Com ja s'ha comentat, es vol tenir un temps de *Training* petit en comparació a l'etapa de recollida de mostres, així doncs si és necessari un estalvi d'energia gran, s'han d'adequar les iteracions *Training* al màxim i depenent de les característiques del escenari es podrà utilitzar una de les dues alternatives.

6.1.4. Variació del *ibit* amb el temps

El valor més important a observar és el *ibit* ja que indica fins a quin punt es codifica la mostra i per tant quant s'estalviarà en energia en cada enviament.

Cal recordar que la mostra de llum s'envia amb 2bytes (16 bits) i que el ibit ens indica quin es el menor nombre de bits que es poden enviar de la mostra al codificar-la.

Com s'observa a la Figura 44 on es veu la variació del *ibit* per a cada sensor i de fons la variació de les mostres de llum en funció del temps, el *ibit* varia directament amb el valor de llum mesurada durant l'etapa *Training*.

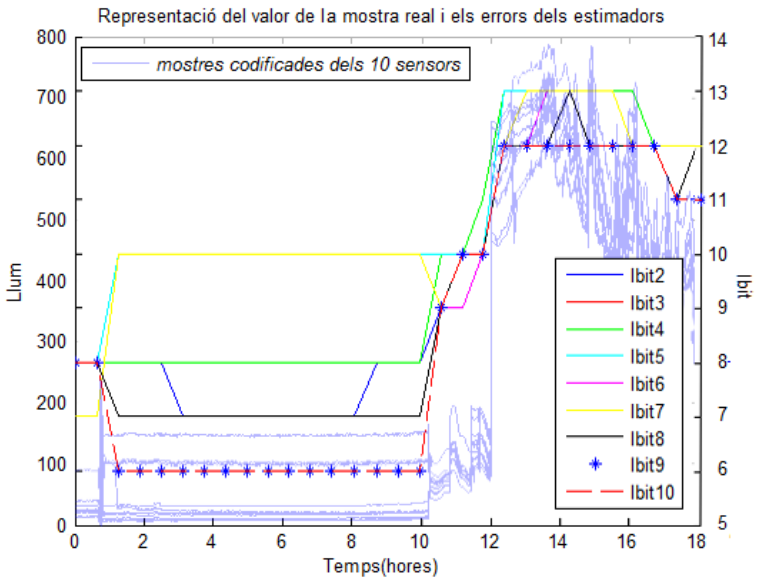


Figura 44. Comparació del *ibit* de cada sensor per a cada iteració amb els valors de llum capturats de 00h-18h

La mateixa comprovació es pot fer a la Figura 45 on en aquest cas es mira la variació del *ibit* des del migdia fins a la nit d'un dia diferent al de la Figura 44. Es torna a observar el mateix comportament del *ibit*, on tindrem una compressió major de la mostra enviada com menor sigui el valor de la llum rebuda.

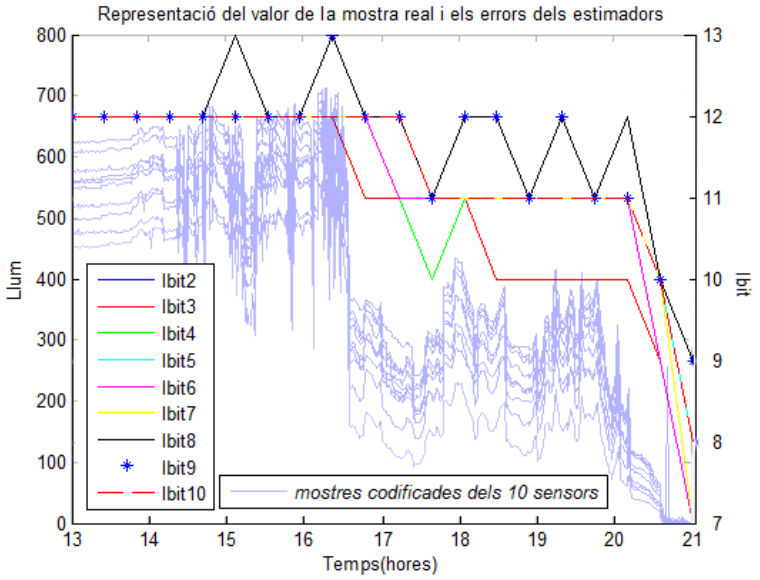


Figura 45. Comparació del *ibit* de cada sensor per a cada iteració amb els valors de llum capturats de 13h-21h

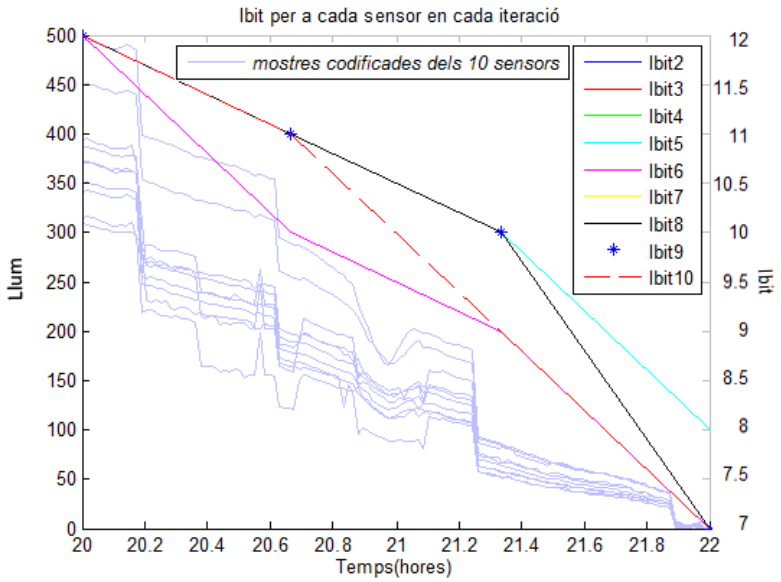


Figura 46. Comparació del *ibit* de cada sensor per a cada iteració amb els valors de llum capturats de 20-22h

Per últim es pot observar en la Figura 46 les hores de major variació de la llum (al vespre) on en dues hores es passa a reduir cada mostra cinc bits.

Per tant, durant les hores amb més llum tindrem una compressió d'un 18.75% (3bits/16bits) ja que l'*ibit* normal en aquests casos és de 13 i a l'alba, el capvespre i el

vespre té una compressió de fins un 50% (9bits/16bits) de la mostra enviada, el que és un estalvi gran d'energia.

7. Conclusions

Explicada la teoria, comentat l'algoritme i observats els resultats ara s'han d'extreure les conclusions del projecte.

Si la finalitat del projecte era la implementació d'una codificació distribuïda de font en una xarxa real, es pot dir que l'objectiu s'ha acomplert satisfactòriament. Durant la creació del algoritme que permetés tenir una comunicació eficient entre la xarxa de sensors i el *PC* s'han assolit molts coneixements sobre la plataforma MoteWorks™ i del sistema operatiu que aquesta utilitza, TinyOS, així com del dialecte de programació del C, NesC, com del programa Matlab®.

Tot això ha desembocat en un algoritme de comunicació entre la xarxa de sensors real i el *PC* que permet no només l'intercanvi d'informació sinó també el processat en temps real de la informació rebuda i la manipulació de les dades enviades pels sensors des del *PC*.

Finalment amb la monitorització d'aquest algoritme podem afirmar que es produeix un estalvi de l'energia prou eficient per a no tenir errors al descodificar les mostres i a més que els resultats es poden millorar adaptant l'etapa *Training* i l'etapa de recollida de mostres codificades al tipus de xarxa que tenim, com també es pot variar la probabilitat d'error per aconseguir un *ibit* menor.

8. Línies Futures

En primer lloc cal aconseguir programar els sensors per a que puguin variar l'estructura del paquet en temps real i no haver d'enviar en divisions de dos bytes o un byte. Aconseguit això la idea del *ibit* es podrà adaptar a la perfecció aconseguint estalvis d'energia molt grans.

També s'ha de recordar que la finalitat d'aquest projecte és implementar una xarxa de sensors real i comprovar el funcionament d'una codificació distribuïda de font en aquesta xarxa. Per tant, el següent pas serà aconseguir fer un sistema robust que pugui funcionar durant un període il·limitat, ignorant o solucionant els errors que vagi trobant.

Un altre fet a tenir en compte és que encara que en aquest cas s'hagi treballat en mostres de llum, l'algoritme funcionarà amb qualsevol paràmetre que mesurin els sensors de la WSN, per tant aquest algoritme es podrà implementar en qualsevol xarxa de sensors per a l'estalvi d'energia.

Com ja s'ha comentat en l'apartat dels resultats, s'ha de idear una manera de tenir un control sobre l'error del filtre de Wiener, ja que aquest indicarà quan cal tornar a actualitzar la matriu de correlacions que es formen a l'etapa *Training*.

9.Referències

[1]. Hayes, Monson H. 7.Optimum Filters. “Statistical Digital Signal Processing and Modeling” s.l. John Wiley & Sons, pp. 335-339, 1996.

[2] J. Chou, D. Petrovic, K. Ramchandran. “A Distributed and Adaptive Signal Processing Approach to Reducing Energy Consumption in Sensor Networks” IEEE Infocom 2003.

[3]Ph. Levis i D. Gay. “TinyOS Programming” s.l.Cambridge

Pàgines Web:

[4].http//www.measurementdevices.com/index.php?name=News&file=article&sid=2068

[5].http//fiji.eecs.harvard.edu/CodeBlue

[6].http//web.mit.edu/newsoffice/2008/trees-0923.html

[7].http//www.datablog.net/2007/04/citysense_an_ur.html

[8].http//www.eecs.berkeley.edu/IPRO/Summary/Old.summaries/03abstracts/polastre.1.html

Annex A

Codi del *Fusion Center* i dels *Sensor Nodes*

Fusion Center

MyApp.nc

```
includes sensorboardApp;

configuration MyApp {
}
implementation {
  components Main, MyAppM, TimerC, LedsC, Photo, GenericComm as Comm;

  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> MyAppM.StdControl;
  Main.StdControl -> Comm.Control;

  MyAppM.Timer -> TimerC.Timer[unique("Timer")];
  MyAppM.Leds -> LedsC.Leds;

  MyAppM.SendMsg -> Comm.SendMsg[AM_XSXMSG];
  MyAppM.UARTMsg -> Comm.SendMsg[AM_XSYMSG];
  MyAppM.ReceiveMsg -> Comm.ReceiveMsg[AM_XSXMSG];
  MyAppM.ReceiveUART -> Comm.ReceiveMsg[AM_XSXUART];
}
```

MyAppM.nc

```
includes sensorboardApp;

module MyAppM {
provides {
    interface StdControl;
    }
  uses {
    interface Timer;
    interface Leds;
    interface ReceiveMsg;
    interface ReceiveMsg as ReceiveUART;
    interface SendMsg;
    interface SendMsg as UARTMsg;
    }
}
implementation {
  bool sending_packet = FALSE;
  uint16_t ibit;
  uint8_t port;
  int node_id;
  int flag=1;
  TOS_Msg msg_buffer;
  TOS_Msg msg_buffer2;
  XDataMsg *pack;
  YDataMsg *pac;

  command result_t StdControl.init() {
    call Leds.init();
    atomic {
      pack = (XDataMsg *)&(msg_buffer.data);
```

```
      pack->xSensorHeader.board_id = SENSOR_BOARD_ID;
      pack->xSensorHeader.packet_id = 1;
      pack->xSensorHeader.node_id = TOS_LOCAL_ADDRESS;
      pack->xSensorHeader.rsvd = 0;
      }
    return SUCCESS;
  }
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
  event result_t Timer.fired()
  {
    return SUCCESS;
  }
/*------------------------Tasques d'enviament-----------------------------*/
//Tasca d'enviament del paquet al PC
void task SendData(){
    atomic sending_packet = TRUE;
    if (call UARTMsg.send(TOS_UART_ADDR,sizeof(XDataMsg),&msg_buffer) != SUCCESS)
            sending_packet = FALSE;
    return;
  }
//Tasca que envia el Flag al Sensor Node
void task SendFlag(){
    atomic sending_packet = TRUE;
    if (call SendMsg.send(TOS_BCAST_ADDR,sizeof(YDataMsg),&msg_buffer2) != SUCCESS)
            sending_packet = FALSE;
    return;
}
//Tasca que envia l'Ibit a la mota
void task IbitMsg(){
    atomic sending_packet = TRUE;
    if (call UARTMsg.send(port,sizeof(XDataMsg),&msg_buffer) != SUCCESS)
             sending_packet = FALSE;
    return;
}
/*-----------------------------------------------------------------------------*/
/*-----------------------------SENDONE-----------------------------------------*/
   //Enviament del paquet satisfactori
event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success) {
    sending_packet = FALSE;
    return SUCCESS;
}
event result_t UARTMsg.sendDone(TOS_MsgPtr msg, result_t success) {
     sending_packet = FALSE;
     return SUCCESS;
}
 /*-----------------------------------------------------------------------------*/
 /*-----------------------------RECEIVE-----------------------------------------*/
 //Rebem des del port USB (PC)
event TOS_MsgPtr ReceiveUART.receive(TOS_MsgPtr m){
    YDataMsg *payload;
```

```
    payload = (YDataMsg*)(m->data);
    ibit = payload->yData.datap2.ibit;
    port = payload->yData.datap2.node_id;

    atomic pack->xData.datap1.constant=ibit;
    if (ibit == 0) {
        post SendFlag();
        call Leds.greenToggle();
    }
    else{
       call Leds.redToggle();
       post IbitMsg();
    }
    return m;
}
//Rebem des dels Sensor Nodes
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr msg){
    XDataMsg *payload;
    uint16_t llum;
    uint16_t constant;
    uint8_t port;
    call Leds.yellowToggle();

    payload = (XDataMsg*)(msg->data);
    llum = payload->xData.datap1.light;
    constant = payload->xData.datap1.constant;
    port = payload->xSensorHeader.node_id;
    atomic pack->xData.datap1.light = llum;
    atomic pack->xData.datap1.constant=constant;
    atomic pack->xSensorHeader.node_id=port;

    return msg;
    }
}
```

Sensor Nodes

MyApp.nc

```
includes sensorboardApp;

configuration MyApp {
}
implementation {
components Main, MyAppM, TimerC, LedsC, Photo, GenericComm as Comm;

  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> MyAppM.StdControl;
  Main.StdControl -> Comm.Control;

  MyAppM.Timer -> TimerC.Timer[unique("Timer")];
  MyAppM.Leds -> LedsC.Leds;
  MyAppM.PhotoControl -> Photo.PhotoStdControl;
  MyAppM.Light -> Photo.ExternalPhotoADC;

  MyAppM.SendMsg -> Comm.SendMsg[AM_XSXMSG];
  MyAppM.ReceiveMsg -> Comm.ReceiveMsg[AM_XSXMSG];
  MyAppM.ReceiveIbit -> Comm.ReceiveMsg[AM_XSYMSG];
}
```

MyAppM.nc

```
includes sensorboardApp;

module MyAppM {
    provides {
        interface StdControl;
    }
    uses {
        interface Timer;
        interface Leds;
        interface StdControl as PhotoControl;
        interface ADC as Light;
        interface ReceiveMsg;
        interface ReceiveMsg as ReceiveIbit;
        interface SendMsg;
     }
}

implementation {
  bool sending_packet = FALSE;
  uint16_t ibit;
  uint16_t F;
  int flag=1;
  int flag2=0;
  uint16_t ibit_mask=0xFFFF;
  uint16_t llum;
  TOS_Msg msg_buffer;
  XDataMsg *pack;
  TOS_Msg msg_buffer2;
  YDataMsg *pac;
```

```
  command result_t StdControl.init() {
      call Leds.init();
      call PhotoControl.init();
      atomic {
        pack = (XDataMsg *)&(msg_buffer.data);
        pack->xSensorHeader.board_id = SENSOR_BOARD_ID;
        pack->xSensorHeader.packet_id = 3;
        pack->xSensorHeader.node_id = TOS_LOCAL_ADDRESS;
        pack->xSensorHeader.rsvd = 0;
        }
  return SUCCESS;
  }
  command result_t StdControl.start() {
      call Timer.start(TIMER_REPEAT, 25000);
      call PhotoControl.start();
      call Light.getData();
  return SUCCESS;
  }
command result_t StdControl.stop() {
    }
/*------------------------Tasques d'enviament-----------------------------*/
void task SendData()
{
  call PhotoControl.stop();
  atomic sending_packet = TRUE;
      // Enviem a la base -> node_id=0
      if (call SendMsg.send(0,sizeof(XDataMsg),&msg_buffer) != SUCCESS)
         sending_packet = FALSE;
      return;
}
async event result_t Light.dataReady(uint16_t data) {
    atomic llum=data;
    if (flag2==0){
            atomic pack->xData.datap1.light = llum;
            atomic pack->xData.datap1.constant = llum;
    }
    else{
            atomic pack->xData.datap1.light = llum&ibit;
            atomic pack->xData.datap1.constant = llum;
    }
    post SendData();
    return SUCCESS;
}
event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success) {
    call Leds.greenToggle();
    atomic sending_packet = FALSE;
    return SUCCESS;
}
//Rebem Flag de la base
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr m){
     YDataMsg *payload;
     payload = (YDataMsg*)(m->data);
     F = payload->yData.datap2.ibit;
     if (F == 0) { //si rebem un Ibit=0, canviem de rutina d'enviament
         atomic flag=!(flag||0); //Fem una NOR per cambiar el FLAG com un interruptor
     }
```

```
    return m;
}
//Rebem Ibit de la base
event TOS_MsgPtr ReceiveIbit.receive(TOS_MsgPtr msg){
    XDataMsg *payload;
    payload = (XDataMsg*)(msg->data);
    atomic ibit = payload->xData.datap1.constant;
    if (ibit==0xFFFF){
        atomic flag2=0;
    }else{
        atomic flag2=1;
    }
    call Leds.yellowToggle();
return msg;
}
/*-----------------------------------------------------------------------*/
/*-----------------------------TIMER-------------------------------------*/
event result_t Timer.fired(){
   if (flag==1){
        call PhotoControl.start();
        call Light.getData();
   }
return SUCCESS;
}
/*-----------------------------------------------------------------------*/
}
```

Sensorboardapp.h

```
TOSH_ALIAS_PIN(PHOTO_CTL, INT1);
TOSH_ALIAS_PIN(TEMP_CTL, INT2);

enum {
  TOSH_ACTUAL_PHOTO_PORT = 1,
  TOSH_ACTUAL_TEMP_PORT = 1,
};

enum {
  TOS_ADC_PHOTO_PORT = 1,
  TOS_ADC_TEMP_PORT = 2,
};

#define FEATURE_SOUNDER    1

// Define SOUND_STATE_CHANGE one of two ways:
//     One time sound at test init   ==>  FALSE
//     Continuous beeping throughout ==>  !sound_state
#define SOUND_STATE_CHANGE  FALSE
//#define SOUND_STATE_CHANGE  !sound_state
// crossbow sensor board id
//#define  SENSOR_BOARD_ID 0x88            //XTYPE_XTUTORIAL=0x88
#define MTS310
#ifndef MTS310
#define  SENSOR_BOARD_ID 0x83            //MTS300 sensor board id
#else
#define  SENSOR_BOARD_ID 0x84            //MTS310 sensor board id
#endif
```

```
typedef struct XSensorHeader{
  uint8_t  board_id;
  uint8_t  packet_id; // 2
  uint8_t  node_id;
  uint8_t  rsvd;
}__attribute__ ((packed)) XSensorHeader;

typedef struct YSensorHeader{
  /*uint8_t  board_id;
  uint8_t  packet_id; // 2
  uint8_t  node_id;
  uint8_t  rsvd;*/
}__attribute__ ((packed)) YSensorHeader;

typedef struct PData1 {
  uint16_t vref;
  uint16_t thermistor;
  uint16_t light;
  uint16_t constant;
} __attribute__ ((packed)) PData1;

typedef struct PData2 {
  uint16_t ibit;
  uint8_t node_id;
  } __attribute__ ((packed)) PData2;

typedef struct XDataMsg {
  XSensorHeader xSensorHeader;
  union {
  PData1    datap1;
  }xData;
} __attribute__ ((packed)) XDataMsg;

typedef struct YDataMsg {
  YSensorHeader ySensorHeader;
  union {
  PData2    datap2;
  }yData;
} __attribute__ ((packed)) YDataMsg;

enum {
  AM_XSXMSG = 0,
  AM_XSYMSG = 1,
  AM_XSXUART = 2,
};
```

Annex B

Algoritme en Matlab

Funcions utilitzades per a l'algoritme al *PC*

Algoritme Matlab

Programa.m

```
%Programa dividit en 3 fases:
%Training Phase
%DSCPhase
%Monitorització
clear all;
clc;
b=0;
J=0;
A=10; %Tamany estimat de la xarxa de sensors (Per al càlcul del nombre de sensors real)
fprintf('Comença el programa\n')
while (b~=20)%número d'iteracions del programa (Training+DSC)
   %Training Phase
   fprintf('Iniciem la iteració %d\n',b)
   [M,V,N,A]=TrainingPhase(A);
   %DSCPhase
   [ibit,w,R]=DSCphase(M,V,N);

   ibit_t(b+1,:)=ibit;%s'emmagatzema el vector d'ibits per la monitorització
   % Monitorització
   I=45; %mirem l'error durant 45 instants de temps
   for i=1:I
      fprintf('Monitoritzem l`instant %d\n',i)
      [error_LWF,error_total,x_cod,x_real]=Monitoritzacio(ibit,w,M);  %Decodificacio  (DSD)  i  Recollida
(proves practiques)
      e_LWF(J+i,:)=error_LWF;
      e_total(J+i,:)=error_total;
      x_cod_t(J+i,:)=x_cod;
      x_real_t(J+i,:)=x_real;
      for j=1:M
         hold on
         plot (1:N,V(j,:));
      end
   end
   b=b+1;
   J=b*I;
   Restart(10);
end
for n=1:J
   eLWFm(n)=mean (e_LWF(n,:));
   etotalm(n)=mean (e_total(n,:));
end
Plots(J,eLWFm,etotalm,M,x_real_t,ibit_t,b,R)
save('10-30iter_llumambient_N=2M_4.mat')
```

TrainingPhase.m

```
%Funció utilitzada per rebre dades dels sensors mda100cb. Dividit, en 2
%apartats
%   *nombreMotes: funció que analitza els missatges rebuts per a saber quantes
%   motes hi ha connectades a la base i envien les seves dades. Seguidament
%   torna a programa el nº de motes que hi ha connectades.
%   *PeriodeProva: funció on durant un període determinat es reben les
%   dades sense comprimir per a treure, després, una matriu de correlacions
```

```
%   estimada. Torna a programa la matriu amb les mostres en cada instant
%   per sensor.
function [M,V,N,A] = TrainingPhase(A)
   fprintf('Som a TrainingPhase\n')
   M = NombreMotes(A);
   fprintf('La xarxa està composta de %d sensors\n',M)
   N=2*M;%TRAINING TIME
   V = PeriodeProva (M,N);
   EnvByte(0,0); %enviem el flag 0's perquè les motes deixin d'enviar
end
```

NombreMotes.m

```
%Funció que analitza els missatges rebuts per a saber quantes
%motes hi ha connectades a la base i envien les seves dades. Funciona
%rebent la dada, i comprovant la id del sensor que la envia, s'emmagatzema
%aquesta id i es va comprovant si ja ha enviat o no. Es rebran mostres fins
%a 3 vegades la mida esperada de la xarxa(A).
%Seguidament torna a programa el nº de motes que hi ha connectades.

function M = NombreMotes (A)
fprintf('Som a NombreMotes\n')
Motes = 0;
K=1;
for N=1:3*A
   s=serial ('com8'); % creació del port
   set(s,'Baudrate',57600); % velocitat de transmissió a 57600 Baudios
   set(s,'StopBits',1); % es llegeix cada 1 bit
   set(s,'DataBits',8); % la dada és de 8 bits
   set(s,'Parity','none'); % sense bit de paritat
   set(s,'Terminator','CR/LF');% "c" caràcter amb el que finalitza l'enviament
   set(s,'OutputBufferSize',1); % "1" és el número de bytes a enviar
   set(s,'InputBufferSize' ,24); % "24" és el número de bytes a rebre
   set(s,'Timeout',30); % 30 segons d'espera fins a rebre alguna dada

   fopen(s); %s'obre el port sèrie
   l = fread(s); %paquet rebut
   e = ConvPacket(l); %conversió del paquet per evitar errors
   fclose(s); %es tanca el port sèrie
   delete(s);

   num_mota = e(9); %guardem el id de la mota
      if num_mota ~= Motes %Entra mostra d'un sensor inèdit
         Motes(K) = num_mota; %guardem la id en el vector de motes
         K=K+1;
         M = length(Motes); %el nº de motes es igual a la longitud del vector Motes
         N=1; %si entra una id nova, posem N a 1 per tornar a fer 6 cicles
         % Dintre del for no funciona N=1!!
      end
end
```

PeriodeProva.m

```
%Funció on durant un TRAINING TIME es reben les dades sense comprimir i es
%guarden en una matriu de mostres per a treure, després, una matriu de
%correlacions estimada. Torna a programa la matriu amb les mostres en cada
```

```
%instant per sensor.

function V = PeriodeProva(M,N)
fprintf('Som a PeriodeProva\n')
fprintf('Recollirem mostres durant %d instants\n',N)
e=0;
i=0;
Motes = 0;
k=0;
for N=1:2*M
   fprintf('Recollim l`instant %d\n',N)
   for n=1:M
      num_mota=0;
      d=15;
      while num_mota~=n
         i=i+1;
         s=serial ('com8'); % creació del port
         set(s,'Baudrate',57600); % velocitat de transmissió a 57600 Baudios
         set(s,'StopBits',1); % es llegeix cada 1 bit
         set(s,'DataBits',8); % la dada és de 8 bits
         set(s,'Parity','none'); % sense bit de paritat
         set(s,'Terminator','CR/LF');% "c" caràcter amb el que finalitza l'enviament
         set(s,'OutputBufferSize',1); % "1" és el número de bytes a enviar
         set(s,'InputBufferSize' ,24); % "24" és el número de bytes a rebre
         set(s,'Timeout',30); % 30 segons d'espera fins a rebre alguna dada

         fopen(s); %s'obre el port sèrie
         l = fread(s);
         e = ConvPacket(l); %conversió del paquet per evitar errors
         fclose(s); %es tanca el port sèrie
         delete(s);
         num_mota = e(9);
         %control per saber quan una mota ha deixat d'enviar
         k=k+1;
         if k>12
            fprintf('No rebem senyal de la mota %d\n',n)
         end
      end
      k=0;
      eb = dec2bin(e,8); %passem a binari el paquet
      g = [eb(d,1:8)]; %segon subpaquet uint8 de la mostra
      f = [eb(d+1,1:8)]; %primer subpaquet uint8 de la mostra
      E = [f g]; %ajuntem els dos paquets de 8 bits per tenir el de 2Bytes
      V(n,N) = bin2dec(E); %guardem la mostra en decimal a la matriu de mostres
      %Per evitar després errors amb divisions per zero
      if V(n,N)==0
         V(n,N)=1;
      end
   end
end
```

DSCPhase.m

```
%  *DSC: Distributed Source Coding, amb la matriu de correlacions i el
%  número de sensors traurem el valor del ibit, el valor de compressió per
```

```
%   a la següent mota a rebre. Després canviem la rutina d'enviament.
function [ibit,w,R] = DSCphase (M,V,N)
   fprintf('Som a DSCphase\n')
   [ibit,w,R]=DSC(M,V,N); %càlcul del ibit i enviament a les motes corresponents
   EnvByte(0,0); %s'avisa a les motes que poden tornar a enviar
end
```

DSC.m

```
function [ibit,w,R] = DSC(M,V,N)
fprintf('Som a DSC\n')
%--------------Càlcul de la matriu de correlacions------------
for n=1:M
   for m=1:M
      R(n,m)=1/(N)*(V(n,1:N)*V(m,1:N)');
   end
end
%------------------Distributed Source Coding---------------------------
w=zeros(M-1);
for i=2:M
   sigma=R(i-1,i-1); %potència del senyal o autocorrelació
   r=R(1:i-1,i); %vector de correlacions del sensor que em de reduir el senyal
   R_=R(1:i-1,1:i-1); %nova matriu de correlacions
 %--------------Càlcul del vector de pesos------------
   w(i-1,:)=[(inv(R_)*r)' zeros(1,M-i)];
 %--------------Càlcul potencia error-----------------
   Poterr = sigma-(r'*inv(R_));
   dist= 1; %en funció del sensor
   Pe = 1e-1; %aquest valor pot ser modificat
   ibit_t = abs(0.5*log2(Poterr(1)/(Pe*dist^2))+1); %ibit que enviarem a la mota vinent per comprimir la
dada que ens enviarà
   ibit(i) = ceil(ibit_t); %ceil redondeja al valor més pròxim
   fprintf('La mota %d té un ibit=%d\n',i,ibit(i))
   EnvByte(ibit(i),i); %s'envia l'ibit a la mota pertinent
end
```

EnvByte.m

```
function EnvByte(ibit,num_mota)
   if ibit==0
      fprintf('Canviem rutina enviament\n')
      ibit_mask1=0;
      ibit_mask2=0;
   else %quan és 0 es per canviar la rutina d'enviament
      fprintf('Enviament del ibit\n')
      %es fa la màscara a partir del ibit
      ibit_mask=[zeros(1,16-ibit) ones(1,ibit)];
      ibit_mask1=bi2de(fliplr(ibit_mask(1,1:8)));
      ibit_mask2=bi2de(fliplr(ibit_mask(1,9:16)));
   end
   %BROADCAST
   paquet=[66;255;255;02;125;03;ibit_mask2;ibit_mask1;num_mota;];
%paquet=[66;126;00;02;125;03;ibit_mask2;ibit_mask1;num_mota;];%UART
   [crc1,crc2] = CRC(paquet); %Es calcula els 2bytes de CRC
   msg=[126 paquet' crc1 crc2 126];%s'afegeix CRC i Framing Byte
%Comprovem que cap byte no sigui igual a 126 o 125 per aplicar l'escaped byte
   for i=2:length(msg)-1 %es passa per alt els Framing Bytes
```

```
        if (msg(i)==125)
            escaped_byte=bitxor(msg(i),32);
            msg_conv =[msg(1:i) escaped_byte];
            msg=[msg_conv msg(i+1:end)];
        elseif (msg(i)==126)
            escaped_byte=bitxor(msg(i),32);
            msg(i)=msg(i)-1;
            msg_conv =[msg(1:i) escaped_byte];
            msg=[msg_conv msg(i+1:end)];
        end
    end
    msg=msg';

    s=serial ('com5'); % creació del port
    set(s,'Baudrate',57600); % velocitat de transmissió a 57600 Baudios
    set(s,'StopBits',1); % es llegeix cada 1 bit
    set(s,'DataBits',8); % la dada és de 8 bits
    set(s,'Parity','none'); % sense bit de paritat
    set(s,'Terminator','CR/LF');% "c" caràcter amb el que finalitza l'enviament
    set(s,'OutputBufferSize',length(msg)); % "length(msg)" és el número de bytes a enviar
    set(s,'InputBufferSize' ,24); % "24" és el número de bytes a rebre
    set(s,'Timeout',30); % 30 segons d'espera fins a rebre alguna dada

    fopen(s); %s'obre el port sèrie
    fwrite(s,msg); %enviem el byte [ibit num_mota]
    fclose(s); %es tanca el port sèrie
    delete(s);
end
```

CRC.m

```
%Funció que calcula el CRC de 16bits del paquet introduit
function [crc1,crc2] = CRC (msg)
msg=msg';
msg = de2bi(msg,8);
for i=1:size(msg,1)
    msg(i,:)=fliplr(msg(i,:));
end
C = reshape(msg.',[],1);
msg=C';

r=zeros(1,16);      % Remainder register initialization
for c3=1:length(msg)

    s1=bitxor(msg(c3),r(1));    % XOR between r0 and the message bit
    s2=bitxor(s1,r(12));        % XOR r11
    s3=bitxor(s1,r(5));         % XOR r4

    r=[r(2:16) s1];             % Left shift of r, and r15 update

    r(11)=s2;                   % r10 update
    r(4)=s3;                    % r3 update
end
msg_FCS=[msg r];                % Message + FCS field
r_c2=sprintf('%d',r);
```

```
crc_hex=dec2hex(bin2dec(r_c2));
crc2=bi2de(fliplr(r(1,1:8)));
crc1=bi2de(fliplr(r(1,9:end)));
end
```

Monitorització.m

```
%Funció que serveix per recollir les mostres dels sensors i extreure els
%resultats que creiem òptims per l'anàlisi d'aquests
function [error_LWF,error_total,x_cod,x_real] = Monitoritzacio (ibit,w,M)
%Monitoritzem el error entre mostres per extreure un periode d'ibit
for n=1:M
   num_mota=0;
   d=15; %posició en el paquet del primer byte de la mostra del sensor
   while num_mota~=n
      s=serial ('com5'); % creació del port
      set(s,'Baudrate',57600); % velocitat de transmissió a 57600 Baudios
      set(s,'StopBits',1); % es llegeix cada 1 bit
      set(s,'DataBits',8); % la dada és de 8 bits
      set(s,'Parity','none'); % sense bit de paritat
      set(s,'Terminator','CR/LF');% "c" caràcter amb el que finalitza l'enviament
      set(s,'OutputBufferSize',1); % "1" és el número de bytes a enviar
      set(s,'InputBufferSize' ,24); % "24" és el número de bytes a rebre
      set(s,'Timeout',30); % 30 segons d'espera fins a rebre alguna dada
      fopen(s); %s'obre el port sèrie
      l = fread(s);
      e = ConvPacket(l); %conversió del paquet per evitar errors
      fclose(s); %es tanca el port sèrie
      delete(s);
      num_mota=e(9);
   end
   fprintf('Monitorització sensor %d\n',num_mota)
   eb = dec2bin(e,8); %passem a binari el paquet
   %Mostra codificada en binari
   bin2=fliplr(de2bi(e(d),8));
   bin1=fliplr(de2bi(e(d+1),8));
   x_cod_bin =[bin1 bin2];
   %Mostra real
   x_real(num_mota) = bin2dec([[eb(18,1:8)] [eb(17,1:8)]]); %ajuntem els dos paquets de 8 bits per
tenir el de 2Bytes
   %Mostra codificada en decimal
   x_cod(num_mota) = bin2dec([[eb(d+1,1:8)] [eb(d,1:8)]]);
   if num_mota==1 %la mota 1 no está codificada
      y = x_cod(num_mota);
      x_dec=x_cod(num_mota);
      x_real(num_mota) = x_cod(num_mota);
   else
      y = w(num_mota-1,1:num_mota-1)*x_cod(1:num_mota-1)';%estimació Wiener
      [x_dec]=DSD(ibit(num_mota),x_cod_bin,y); %mostra descodificada
   end
   error_LWF(n)= abs(x_real(num_mota)-y); %error del filtre de Wiener
   error_total(n)=abs(x_real(num_mota)-x_dec); %error total
end
end
```

DSD.m

```
%DSD: Distributed Source Decoding. Funció que descodifica les mostres
%codificades en funció del ibit i de la estimació
function[x_dec] = DSD (ibit,x_cod_bin,y)
   pos = 1:2^16;
   c1=1:2^16;
   ii=16;
   for jj = 0:ibit
      ii=16;
      if x_cod_bin(ii-jj)==0
         pos = pos(1:2:end);
      else
         pos = pos(2:2:end);
      end
   end
   [nothing, pos_dec] = sort(abs(y-c1(pos)));
   x_dec = c1(pos(pos_dec(1)));
end
   %ibit->nombre de bits, no ibit_mask
   %x_cod->mostra q rebem ja codificada
   %y->mostra estimada y=w*x
   %c1->vector de mostres reals, en aquest cas c1=pos
```

ConvPacket.m

```
% Funció que recorre el paquet i extreu els escaped byte per no cometre
% errors en la lectura dels paquets
function [msg_conv]=ConvPacket(msg)
conv=32; %0x20 en hexadecimal
j=1;
msg_conv=0;
%msg =
msg=msg';
   for i=2:length(msg)-1

      if(msg(i)==125) %si es igual a 25 vol dir que s'ha utilitzat escapedbyte
         msg_conv(j)=bitxor(msg(i+1),conv);
         msg=[msg(1:i) msg(i+2:end) 0];
      else
         msg_conv(j)=msg(i);
      end
      j=j+1;
   end
end
```

Restart.m

```
%Funció per tornar deixar d'enviar mostres codificades amb els sensors
function Restart(M)
for i=2:M
   EnvByte(16,i);
end
```

Plots.m

```
%Funció per a la representació dels resultats desitjats, en aquest cas:
%Error Wiener vs. Error total
%Error relatiu Wiener vs. Error relatiu Total
```

```
%Representació en tres dimensions de les mostres capturades
%Representació del ibit per a 10 sensors
function Plots(I,eLWFm,etotalm,M,x_real_t,ibit_t,b,R)
   x=linspace(0,2,length(eLWFm)); %eix horari
   %Error Wiener vs. Error total
   figure()
   hold on;

   plot(x,eLWFm,'-b');
   plot(x,etotalm,'-r');

   Title('Representació del error mitjà del filtre de Wiener i l`error total mitjà respecte el temps');
   %xlabel('Temps(minuts)');
   xlabel('Temps(hores)');
   ylabel('Error');
   legend('\it{Error} LWF','\it{Error} TOTAL',2)

   %Càlcul del error relatiu
   for i=1:M
   S(i)=R(i,i);
   end
   sigma_mitj=mean(S);
   RMSE=eLWFm/sqrt(sigma_mitj);
   RMSEt=etotalm/sqrt(sigma_mitj);

   %Error relatiu Wiener vs. Error relatiu Total
   figure()
   hold on;

   plot(x,RMSE,'-b');
   plot(x,RMSEt,'-r');

   Title('Representació del error relatiu del filtre de Wiener i l`error total relatiu respecte el temps');
   %xlabel('Temps(minuts)');
   xlabel('Temps(hores)');
   ylabel('Error');
   legend('\it{Error relatiu} LWF','\it{Error relatiu} TOTAL',2);

   %representació en tres dimensions de les mostres capturades
   figure()
   mesh(x_real_t)
   Title('Mostres de llum rebudes dels 10 sensors');
   xlabel('# Sensor');
   ylabel('Instants');
   zlabel('Llum')

   %Representació del ibit per a 10 sensors
   figure()
   hold on
   x1=linspace(0,18,b);
   plot(x1,ibit_t(:,2))
   plot(x,ibit_t(:,3),'r')
   plot(x,ibit_t(:,4),'g')
   plot(x,ibit_t(:,5),'c')
   plot(x,ibit_t(:,6),'m')
   plot(x,ibit_t(:,7),'y')
```

```
    plot(x,ibit_t(:,8),'k')
    plot(x,ibit_t(:,9),'b*')
    plot(x,ibit_t(:,10),'--r')
    Title('Ibit per a cada sensor en cada iteració');
    %xlabel('Temps(minuts)');
    xlabel('Temps(hores)');
    ylabel('Ibit');
    legend('Ibit2','Ibit3','Ibit4','Ibit5','Ibit6','Ibit7','Ibit8','Ibit9','Ibit10',9);
end
```

Annex C

Datasheets mòduls MoteWorks™

IRIS XM2110

Crossbow

IRIS

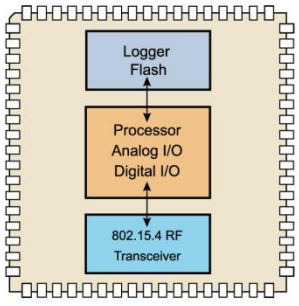
WIRELESS MEASUREMENT SYSTEM

- 2.4 GHz IEEE 802.15.4, Tiny Wireless Measurement System
- Designed Specifically for Deeply Embedded Sensor Networks
- 250 kbps, High Data Rate Radio
- Wireless Communications with Every Node as Router Capability
- Expansion Connector for Light, Temperature, RH, Barometric Pressure, Acceleration/Seismic, Acoustic, Magnetic and other Crossbow Sensor Boards

Applications

- Indoor Building Monitoring and Security
- Acoustic, Video, Vibration and Other High Speed Sensor Data
- Large Scale Sensor Networks (1000+ Points)

ZigBee™Alliance



XM2110CA Block Diagram

IRIS

The IRIS is a 2.4 GHz Mote module used for enabling low-power, wireless sensor networks. The IRIS Mote features several new capabilities that enhance the overall functionality of Crossbow's wireless sensor networking products.

Product features include:

- Up to three times improved radio range and twice the program memory over previous MICA Motes
- Outdoor line-of-sight tests have yeilded ranges as far as 500 meters between nodes without amplification
- IEEE 802.15.4 compliant RF transceiver
- 2.4 to 2.48 GHz, a globally compatible ISM band
- Direct sequence spread spectrum radio which is resistant to RF interference and provides inherent data security
- 250 kbps data rate
- Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
- Plug and play with Crossbow's sensor boards, data acquisition boards, gateways, and software

MoteWorks™ enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks. MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-the-air-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and configuration.

Processor & Radio Platform

The XM2110CA is based on the Atmel ATmega1281. The ATmega1281 is a low-power microcontroller which runs MoteWorks from its internal flash memory. A single processor board (XM2110) can be configured to run your sensor application/processing and the network/radio communications stack simultaneously. The IRIS 51-pin expansion connector supports Analog Inputs, Digital I/O, I2C, SPI and UART interfaces. These interfaces make it easy to connect to a wide variety of external peripherals.

Sensor Boards

Crossbow offers a variety of sensor and data acquisition boards for the IRIS Mote. All of these boards connect to the IRIS via the standard 51-pin expansion connector. Custom sensor and data acquisition boards are also available. Please contact Crossbow for additional information.

Document Part Number: 6020-0124-01 Rev A

Phone: 408.965.3300 • Fax: 408.324.4840 • E-mail: info@xbow.com • Web: www.xbow.com

Processor/Radio Board	XM2110CA	Remarks
Processor Performance		
Program Flash Memory	128K bytes	
Measurement (Serial) Flash	512K bytes	> 100,000 Measurements
RAM	8K bytes	
Configuration EEPROM	4K bytes	
Serial Communications	UART	0-3V transmission levels
Analog to Digital Converter	10 bit ADC	8 channel, 0-3V input
Other Interfaces	Digital I/O,I2C,SPI	
Current Draw	8 mA	Active mode
	8 µA	Sleep mode (total)
RF Transceiver		
Frequency band[1]	2405 MHz to 2480 MHz	ISM band, programmable in 1 MHz steps
Transmit (TX) data rate	250 kbps	
RF power	3 dBm (typ)	
Receive Sensitivity	-101 dBm (typ)	
Adjacent channel rejection	36 dB	+ 5 MHz channel spacing
	34 dB	- 5 MHz channel spacing
Outdoor Range	> 300 m	1/4 wave dipole antenna, LOS
Indoor Range	> 50 m	1/4 wave dipole antenna, LOS
Current Draw	16 mA	Receive mode
	10 mA	TX, -17 dBm
	13 mA	TX, -3 dBm
	17 mA	TX, 3 dBm
Electromechanical		
Battery	2X AA batteries	Attached pack
External Power	2.7 V - 3.3 V	Molex connector provided
User Interface	3 LEDs	Red, green and yellow
Size (in)	2.25 x 1.25 x 0.25	Excluding battery pack
(mm)	58 x 32 x 7	Excluding battery pack
Weight (oz)	0.7	Excluding batteries
(grams)	18	Excluding batteries
Expansion Connector	51-pin	All major I/O signals

Notes

[1]5 MHz steps for compliance with IEEE 802.15.4/D18-2003.
Specifications subject to change without notice

IRIS Mote (bottom view)

MIB520CA Mote Interface Board

Base Stations

A base station allows the aggregation of sensor network data onto a PC or other computer platform. Any IRIS Mote can function as a base station when it is connected to a standard PC interface or gateway board. The MIB510 or MIB520 provides a serial/USB interface for both programming and data communications. Crossbow also offers a stand-alone gateway solution, the MIB600 for TCP/IP-based Ethernet networks.

Ordering Information

Model	Description
XM2110CA	2.4 GHz IRIS OEM Reference Board

Document Part Number: 6020-0124-01 Rev A

Crossbow Technology, Inc. • 4145 North First Street • San Jose, California 95134-2109

Fusion Center: MIB520CB

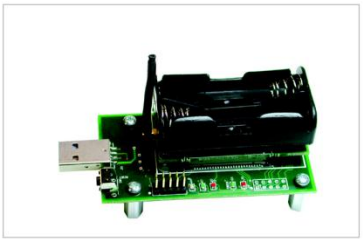
MIB520

USB INTERFACE BOARD

- Base Station for Wireless Sensor Networks
- USB Port Programming for IRIS/MICAz/MICA2 Hardware Platforms
- Supports JTAG code debugging
- USB Bus Power

Applications

- USB Interface
- Testbed Deployments
- In-System Programming



MIB520CB with attached Mote

MIB520CB

The MIB520CB provides USB connectivity to the IRIS and MICA family of Motes for communication and in-system programming. Any IRIS/MICAz/MICA2 node can function as a base station when mated to the MIB520CB USB interface board. In addition to data transfer, the MIB520CB also provides a USB programming interface.

The MIB520CB offers two separate ports: one dedicated to in-system Mote programming and a second for data communication over USB.The MIB520CB has an on-board processor that programs Mote Processor Radio Boards. USB Bus power eliminates the need for an external power source.

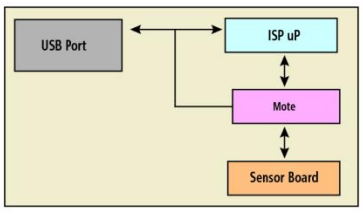
Specifications

USB Interface
- Baud Rate: 57.6 K
- Male to Female USB cable (included with unit)

Mote Interface
- Connectors:
  - 51-pin
- Indicators:
  - Mote LED's: Red Green, Yellow

Programming Interface
- Indicators:
  - LEDs - Power Ok (Green), Programming in Progress (Red)
- Switch to reset the programming processor and Mote.

Jtag Interface
- Connector: 10-pin male header POWER
- USB Bus powered



MIB520CB Block Diagram

Ordering Information

Model	Description
MIB520CB	USB PC Interface Board

Document Part Number: 6020-0091-04 Rev A

Phone: 408.964.9700 | Fax: 408.324.4841 | E-mail: infoca@memsic.com | www.memsic.com

Sensor: MDA100CB

6 MDA100CA/ MDA100CB

MD100CA and MDA100CB have the same content in this chapter except for some minor changes.

The MDA100 series sensor boards have a precision thermistor, a light sensor/photocell, and general prototyping area. The prototyping area supports connection to all eight channels of the Mote's analog to digital converter (ADC0–7), both USART serial ports and the I2C digital communications bus. The prototyping area also has 45 unconnected holes that are used for breadboard of circuitry.

6.1.1 Thermistor

The thermistor, (YSI 44006, http://www.ysi.com) sensor is a highly accurate and highly stable sensor element. With proper calibration, an accuracy of 0.2 °C can be achieved. The thermistor's resistance varies with temperature. (See Table 6-1 and the resistance vs. temperature graph in Figure 6-3) This curve, although non-linear, is very repeatable. The sensor is connected to the analog-digital converter channel number 1 (ADC1) thru a basic resistor divider circuit. In order to use the thermistor, the sensor must be enabled by setting digital control line INT2 high. See the Figure 6-1 below.

Table 6-1. *Thermistor Specifications*

Type	YSI 44006
Time Constant	10 seconds, still air
Base Resistance	10 kΩ at 25 °C
Repeatability	0.2 °C

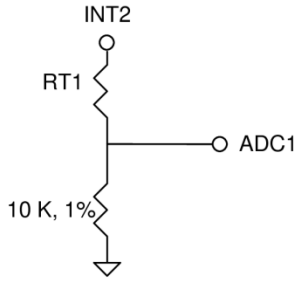


Figure 6-1(a). *Schematic of the Thermistor on MDA100CA*

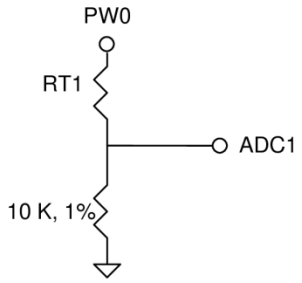


Figure 6-2(b). *Schematic of the Thermistor on MDA100CB*

Table 6-2. *Resistance vs. Temperature, ADC1 Reading*

Temperature (°C)	Resistance (Ohms)	ADC5 Reading (% of VCC)
-40	239,800	4%
-20	78,910	11%
0	29,940	25%
25	10,000	50%
40	5592	64%
60	2760	78%
70	1990	83%

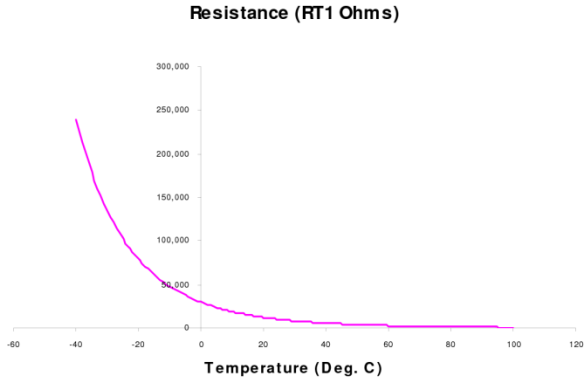


Figure 6-3. *Resistance vs. Temperature Graph*

6.2 Conversion to Engineering Units

The Mote's ADC output can be converted to Kelvin using the following approximation over 0 to 50 °C:

$$1/T(K) = a + b \times \ln(R_{thr}) + c \times [\ln(R_{thr})]^3$$

where:

$R_{thr} = R1(ADC\_FS-ADC)/ADC$

a = 0.001010024

b = 0.000242127

c = 0.000000146

R1 = 10 kΩ

ADC_FS = 1023, and

ADC = output value from Mote's ADC measurement.

6.3 Light Sensor

The light sensor is a simple CdSe photocell. The maximum sensitivity of the photocell is at the light wavelength of 690 nm. Typical on resistance, while exposed to light, is 2 kΩ. Typical off resistance, while under dark conditions, is 520 kΩ. In order to use the light sensor, digital control signal PW1 must be turned on. The output of the sensor is connected to the analog-digital converter channel 1 (ADC1). When there is light, the nominal circuit output is near VCC or full-scale, and when it is dark the nominal output is near GND or zero. Power is controlled to the light sensor by setting signal INT2.

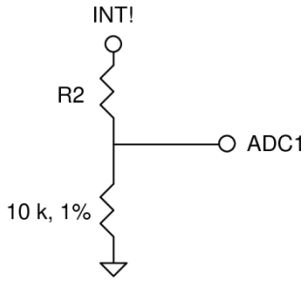


Figure 6-4. *Schematic of the light sensor*

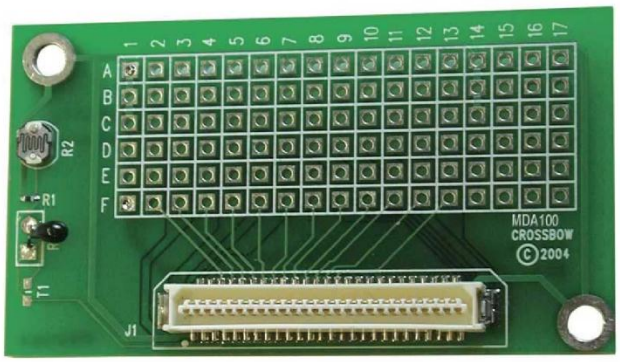
6.4 Prototyping Area

The prototyping area is a series of solder holes and connection points for connecting other sensors and devices to the Mote. The prototyping area layout is shown in the diagram and tables below.

Table 6-3. *Connection Table for MDA100. Use the photo (top view) below the table to locate the pins.*

	A	B	C	D	E	F
1	GND	GND	GND	VCC	VCC	VCC
2	OPEN	OPEN	USART1_CK	INT3	ADC2	PW0
3	OPEN	OPEN	UART0_RX	INT2†	ADC1†	PW1†
4	OPEN	OPEN	UART0_TX	INT1	ADC0†	PW2
5	OPEN	OPEN	SPI_SCK	INT0	THERM_PWR	PW3
6	OPEN	OPEN	USART1_RX	BAT_MON	THRU1	PW4
7	OPEN	OPEN	USART1_TX	LED3	THRU2	PW5
8	OPEN	OPEN	I2C_CLK	LED2	THRU3	PW6
9	OPEN	OPEN	I2C_DATA	LED1	RSTN	ADC7
10	OPEN	OPEN	PWM0	RD	PWM1B	ADC6
11	OPEN	OPEN	PWM1A	WR	OPEN	ADC5
12	OPEN	OPEN	AC+	ALE	OPEN	ADC4
13	OPEN	OPEN	AC-	PW7	OPEN	ADC3
14	GND	GND	GND	VCC	VCC	VCC
15	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
16	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN
17	OPEN	OPEN	OPEN	OPEN	OPEN	OPEN

†Shared functionality



WARNING: Never connect signals that are greater than VCC (3V typical) or less than 0 V to any of the holes that connect to the Mote Processor Radio board. It is okay to connect different voltages to the non-connected holes. However, be careful. If a voltage out of the range of 0 to Vcc should reach the Mote Processor Radio Board damage will occur.

Resum:

Les xarxes de sensors sense fils són cada cop més utilitzades en el món en el que vivim. Un dels problemes més importants d'aquestes és el consum energètic dels sensors, mancant l'autonomia d'aquests sistemes.

Es sap que les mesures dels sensors d'aquestes xarxes en molts casos són correlades entre sí. Aquest fet ha inspirat el sistema de codificació Distributed Source Coding, el qual utilitza la correlació existent entre els elements de la xarxa per aconseguir una reducció de bits en l'enviament de les mostres d'aquests

En aquest treball es provarà el sistema de codificació esmentat en una xarxa real de sensors. Per aquest fi, es programaran els sensors i un algoritme en Matlab® que s'encarregui tant de l'emmagatzematge de les mostres com del càlcul i l'enviament del valor de bits a reduir en cada sensor. Aquest valor s'anomenarà ibit.

Resumen:

Las redes de sensores inalámbricas son cada vez más útiles en el mundo en el que vivimos. Uno de los problemas más importantes de éstas es el consumo energético de los sensores, mancando la autonomía de estos sistemas.

Se sabe que las muestras recogidas por los sensores de estas redes en muchos casos están correladas entre sí. Este hecho ha inspirado el sistema de codificación Distributed Source Coding, el cual utiliza la correlación existente entre los elementos de la red para conseguir una reducción de bits en el envío de las muestras de éstos.

En este proyecto se probará el sistema de codificación nombrado en una red real de sensores. Para este fin, se programaran los sensores y un algoritmo en Matlab® que se encargará tanto del almacenamiento de las muestras como del cálculo y el envío del valor de bits a reducir de cada sensor. A este valor se le nombrará ibit.

Summary:

Wireless Sensor Networks (WSN) are increasingly useful in the world which we live. One of the most important problems of these is the energy consumption of sensors, undermining the autonomy of these systems.

It is known that samples collected by sensors in these networks often are correlated with each other. This has inspired the Distributed Source Coding system, which uses the correlation between the network elements to achieve a reduction of bits in the transmission of samples.

This project will test the coding system named in a real WSN. To this aim, the sensors will be programmed and an algorithm will be done with Matlab® that will handle the storage of samples and both the calculation and send the value of bits to reduce the samples from each sensor. This value will be named ibit.