Implementació d'un sistema de compressió de dades en el dispositiu mota Cou24 sota TinyOS.

Damià Arasa

Enginyeria Tècnica en Informàtica de Sistemes

Consultor: Jordi Bécares Ferrés

10/06/2011

Dedicatòria i agraïments

En el moment de posar punt i final als meus estudis a la UOC, deixo constància del meu agraïment a tots aquells que durant el llarg període transcorregut m'han animat i ajudat a prosseguir.

Per a Lídia

Resum

Les limitacions en la capacitat d'emmagatzemament dels dispositius sensors sense fils o *motes* és consubstancial al seu disseny, però pot convertir-se en un problema en determinades situacions. Aquest projecte de fi de carrera ha esta motivat per l'interès en superar aquestes limitacions en una situació concreta: una mota dedicada a prendre mostres amb alguns dels seus sensors que passa un període llarg de temps sense poder comunicar amb la base.

Per aconseguir aquest objectiu s'ha dissenyat i implementat un sistema que comprimeix les mostres en una mota només quan l'espai d'emmagatzemament està pròxim a exhaurir-se. Aquest procés de compressió pot repetir-se indefinidament i cada vegada que actua elimina només les mostres menys significatives, tot preservant la freqüència del mostreig.

A continuació es descriu l'algorisme que s'ha dissenyat, respectuós amb les capacitats de processament de la mota, la implementació que s'ha fet en nesC per a una mota Cou24 amb TinyOS, el programari complementari que s'ha desenvolupat en Java per a PC, i el resultat de les proves que s'han fet de tot aquest conjunt.

Índex de continguts

1. Introducció

1.1 Justificació

Els dispositius sensors sense fils o motes tenen la limitació de disposar d'una capacitat d'emmagatzemament reduïda. En el cas de les motes Cou24 la memòria RAM és de 8 KB, i no tot aquest espai pot utilitzar-se per emmagatzemar dades. En una situació normal això no té perquè suposar un problema, ja que les motes estan dissenyades per a formar part d'una xarxa i per tant poden transmetre de manera immediata les dades que recullen. Però si aquesta comunicació s'interromp durant un període de temps prolongat es corre el risc de que es perdin dades.

L'aplicació que s'ha implementat en aquest TFC pretén pal·liar la gravetat d'aquesta e pèrdua de dades. La solució desenvolupada consisteix en un sistema de compressió que, quan és necessari, genera nou espai de memòria lliure esborrat de les dades menys significatives.

De retruc això obre la possibilitat d'utilitzar les motes com a dispositius "stand-alone" en casos on la precisió no sigui molt important. En aquest es escenari la capacitat de comunicació de les motes només s'utilitzaria per una recollides esporàdiques de dades, tenint en qualsevol cas la seguretat de que allargar el temps entre recollides successives de dades només comportaria una pèrdua de detall.

1.2 Descripció del projecte

L'objectiu del projecte és dissenyar i implementar un algorisme que doti les motes d'una capacitat "il·limitada" d'emmagatzemament de mostres, aplicant una compressió variable que a partir del moment en que la memòria s'ompli vagi eliminant selectivament les mostres menys significatives. Per tant, a mesura que passi el temps no desapareixeran ni les mostres més antigues ni els valors més extrems, i sempre serà possible una reconstrucció de com ha evolucionat la magnitud estudiada, tot i que s'aniran perden detalls. Si ho representem en forma de gràfica, el pas del temps farà que es perdin les oscil·lacions petites tipus "dents de serra", però s'aniran mantenint els màxims, els mínims, els punts d'inflexió corresponents a les oscil•lacions més àmplies, i també el pendent aproximat. Dit d'una altra manera, l'efecte de pas del temps provocarà només la reducció de l'escala de la gràfica.

1.3 Objectius

a) Definició

1. Dissenyar un algoritme de compressió d'un flux de dades en el dispositiu.
2. Minimitzar la pèrdua d'informació provocada per la compressió.
3. Permetre reconstruir l'evolució del flux de dades en el període emmagatzemat.
4. Baix temps d'execució.
5. Dissenyar diferents nivells de compressió.
6. Adaptar dinàmicament el nivell de compressió al nombre de dades.
7. Capacitat de compressió il·limitada.

b) Descripció

1. L'objectiu més general és comprimir un flux de dades provinent del mostreig dels valors llegits en algun dels sensors del dispositiu. No se suposa que aquests valors seran periòdics. La finalitat pràctica és que el dispositiu pugui emmagatzemant dades en períodes en que no hi hagi comunicació, tot i assumint una certa pèrdua d'informació.

2. Només s'han de comprimir dades quan sigui necessari, i sempre amb el mínim nivell de compressió que permeti obtenir una certa quantitat d'espai addicional per a emmagatzemar noves dades durant un cert període.

3. La compressió ha de fer-se de tal manera que es vagin eliminant detalls però es mantingui l'evolució general del flux de dades, és a dir, com a mínim els valors màxims i mínims i els pendents. Això implica una certa anàlisi del flux, i l'evitació del efecte *aliasing*.

4. Es pretén que les rutines de compressió no interrompin el mostreig de les dades, i per tant que la seva execució sigui ràpida i/o pugui fragmentar-se en rutines petites de ràpida execució.

5. S'han de poder comprimir dades ja comprimides, bé aplicant un algoritme diferent, bé aplicant el mateix amb una parametrització diferent. Un requeriment important que imposa aquest objectiu és que un nivell de compressió no "desfiguri" les dades fins al punt que el nivell següent no pugui reconèixer-les i per tant no tingui criteri per a complir l'objectiu 2 (minimitzar la pèrdua d'informació).

6. Ha d'haver un mecanisme de supervisió de la comprensió que decideixi quin és el nivell de compressió a aplicar, i quan es fa necessari passar al següent.

7. L'algorisme o nivell de màxima compressió ha de poder aplicar-se indefinidament, i tenir sempre èxit: aconseguir sempre incrementar l'espai disponible. La pèrdua d'informació serà cada vegada més gran, però sempre d'acord amb l'objectiu 3: sempre s'ha de poder reconstruir el flux de dades, a una certa escala.

1.4 Enfocament i metodologia

Una vegada fixats els objectius, la primera tasca ha consistit en fer una recerca d'algoritmes de compressió susceptibles de ser utilitzat per a la finalitat proposada. Donades les especificitats de l'entorn (sistema encastat en un dispositiu amb limitacions de memòria i potència de processament) i l'exigència dels requeriments (en especial la capacitat de compressió il·limitada) no ha estat possible adaptar algun dels algoritmes disponibles, i per tant la tasca més crítica ha estat el disseny d'un algoritme propi.

L'enfocament que s'ha utilitzat en el disseny d'aquest l'algoritme ha estat el de interpretar el flux de mostres com una gràfica basada en la seva representació cartesiana. A partir d'aquí el problema s'ha convertit en el de representar una gràfica a diferents escales minimitzant la pèrdua de detall.

Una vegada trobat un procediment matemàtic idoni per a dur a terme la compressió, la resta de la feina a consistit a implementar-lo, per a la qual cosa la metodologia ha estat imposada per les característiques del sistema operatiu TinyOS i del llenguatge nesC.

1.5 Planificació

Inicialment el projecte s'havia descompost en les tasques següents:

1. **Anàlisi prèvia del projecte, recerca i documentació sobre algorismes de compressió i compactació**. El resultat havia de permetre prendre les decisions de disseny més importants, com ara el format en que s'emmagatzemarien les dades, quin o quins algorismes de compressió s'utilitzarien, i si caldria utilitzar algorismes de compactació de memòria.

2. **Creació d'un esquelet de les aplicacions**. Per al dispositiu: captura de dades, emmagatzemament i transmissió. Per al PC: recepció i arxiu.
3. **Creació de la resta dels mòduls**. En el dispositiu: compressió. En el PC: descompressió.
4. **Proves, comparació de resultats i refinaments**.

La seva distribució temporal era:

Tasca		Inici	Fi
1	Recerca, anàlisi i documentació	25/03/11	7/04/11
2	Crear esquelet d'aplicacions	25/03/11	26/04/11
3	Crear mòduls específics	26/04/11	18/05/11
4	Proves i refinaments	18/05/11	31/05/11

La causa més important de la desviació sobre aquesta planificació inicial ha estat motivada pel la constatació de que el sistema previst per permetre comparar les dades comprimides amb les no comprimides no era adequat. Per això ha estat necessari refer part de l'estructura o esquelet de l'aplicació de la mota, d'acord amb un suggeriment del consultor. S'ha afegit la tasca:

Generalitzar lectura de mostres i cues. Permetre que l'aplicació utilitzi 3 cues per a emmagatzemar les dades (tantes com sensors té la mota Cou24) i permetre associar cada cua amb un sensor, no necessàriament diferent.

La decisió de dur a terme aquest replantejament s'ha pres una vegada ja s'havien creat alguns dels mòduls específics, en particular el de compressió de dades, per la qual cosa s'ha hagut de modificar part de la feina feta en aquesta tasca. També ha comportat una reducció en el temps dedicat a la tasca final de proves i refinaments.

Finalment la temporització de les tasques ha estat:

Tasca		Inici	Fi
1	Recerca, anàlisi i documentació	25/03/11	7/04/11
2	Crear esquelet d'aplicacions	25/03/11	26/04/11
3	Crear mòduls específics	26/04/11	15/05/11
4	Generalitzar lectura de mostres i cues	15/05/11	22/05/11
5	Proves i refinaments	22/05/11	31/05/11

1.6 Recursos

Els recursos utilitzats han estat els següents:

- Maquinari:
    - Un PC.
    - Una parella de motes.
- Programari
    - Una màquina virtual amb el S.O. Ubuntu 10.04 i el TinyOS preinstal.lat.
    - Llenguatge de programació nesC. Compilador i llibreries incloses en el TinyOS.
    - Entorn de programació Eclipse amb el plugin Yeti per a facilitar el desenvolupament en nescC per a TinyOS.
    - Utilitat *meshprog* per a descarregar el programa a la mota.
    - Utilitat *SerialForwarder*, que forma part de l'entorn TinyOS, per a comunicar la mota amb el PC via sèrie a través d'un port USB.
    - Programa *BaseStation* en una de les motes, per a fer de pont comunicant via sèrie amb el PC i via ràdio amb l'altra mota.

1.7 Productes obtinguts

El resultat del desenvolupament del projecte ha esta un programa per a la mota, anomenat **ComprMostres**, i 3 aplicacions complementàries en Java: **ConfigSessio**, **RecullMostres** i **DescomprMostres**.

El programa **ComprMostres** llegeix 1, 2 o els 3 sensors de la mota a una freqüència determinada i envia les lectures a través de la ràdio. Mentre no rep confirmació dels enviaments, guarda les lectures de cada sensor en una cua. Quan la cua està gairebé plena, engega un procés de compressió de les mostres que genera nou espai lliure i alhora minimitza la pèrdua d'informació. El procés pot repetir-se indefinidament, incrementant-se el nivell de compressió a cada nova execució.

L'aplicació **ConfigSessio** permet configurar el funcionament de la mota a través de paràmetres de la línia de comandes (es detallen en l'annex I). Es comunica amb la mota mitjançant la utilitat **SerialForwarder** i una **BaseStation**, i reintenta l'enviament del missatge de configuració fins que rep confirmació per part de la mota.

L'aplicació **RecullMostres** comunica amb la mota amb el mateix procediment que l'aplicació anterior. La seva funcionalitat consisteix en estar a l'espera de rebre missatges d'enviament de mostres i emmagatzemar-los en un fitxer. Els paràmetres de configuració es detallen també en l'annex I.

L'aplicació **DescomprMostres** rep com a entrada un fitxer de mostres i genera com a sortida un fitxer per a cada mota i cua en el que s'han interpolat les mostres eliminades en el procés de compressió. Val a dir que l'algoritme de compressió exigeix una codificació de les dades que les altera fins i tot en el cas que no s'hagin comprimit. Això comporta que necessàriament s'haurà d'utilitzar l'aplicació **DescomprMostres** per a recuperar els valors originals, hagi hagut compressió o no.

1.8 Descripció de la resta de la memòria

La resta d'aquest document conté la següent informació:

- L'apartat 2 descriu breument els antecedents de sistemes de compressió del dades semblants al que s'han desenvolupant en aquest projecte.
- Els apartats 3 i 4 contenen el nucli del treball realitzat. En el 3 es dóna una explicació general del disseny i s'argumenten les decisions més importants. En el 4 s'expliquen en detall els algorismes utilitzats.
- En l'apartat 5 s'expliquen les proves que s'han dut a terme i els resultats que se n'han pogut extreure.
- L'apartat 6 analitza la viabilitat tècnica del projecte.
- L'apartat 7 recull les conclusions, incloent-hi una proposta d3e millores i una autoavaluació.
- L'apartat 8 conté un glossari d'alguns termes específics del projecte.
- L'apartat 9 inclou la bibliografia.
- Finalment als annexos s'inclou un manual d'usuari, una explicació dels procediments de compilació i execució, i alguns exemples d'ús.

2. Antecedents

2.1 Estat de l'art

2.1.1 Implementacions de sistemes de compressió en la mota

Existeixen com a mínim dos projectes de compressió de dades sota TinyOS. En cap dels dos casos els objectius coincideixen plenament amb els del present projecte, però tots dos tenen un gran interès.

El projecte **SenZip** proporciona un servei de compressió distribuït per xarxa compatible amb el CTP (*Collection Tree Protocol*), d'acord amb el *TinyOs proposa1 123*. És, per tant, un esquema de compressió per xarxa, que té en compte la distribució espacial dels sensors. Cada element de la xarxa és alhora un node de comunicació i un sensor de dades. La diferència principal amb el present projecte és que SenZip no pretén una compressió progressiva, sinó únicament que cada sensor enviï els paquets de dades comprimits. L'objectiu és principalment un estalvi d'energia, estimant que l'increment de l'energia necessari consumida per la mota per a comprimir les dades serà menor que la reducció d'energia aconseguida a causa d'escurçar els períodes d'utilització de la ràdio, ja que els paquets comprimits es transmetran en menys temps. Utilitza algorismes de compressió adequats per a senyals contínues, tals com el 2D wawelet o el DPCM, que seran analitzats en l'apartat següent. El cost computacional d'aquests algorismes els fa adequats per a comprimir paquets petits en una mota, però no per a comprimir una gran quantitat de dades, raó per la qual han estat descartats per al present projecte.

L'altre cas interessant es descriu en el document *Implementation of Data Compression and FFT on TinyOS* (Ning Xu). Implementa dos algorismes de compressió. El primer és la codificació Lempel-Ziv (LZ77), que és un algorisme sense pèrdua semblant a la codificació Huffmann. No resulta aplicable en el present projecte pels motius que s'explicaran en l'apartat següent. El segon és la transformada ràpida de Fourier (FFT), adequada per a la compressió de senyals, en la seva variant més ràpida, l'algorisme de Sorensen. Les conclusions de l'autor demostren la no conveniència d'aquest tipus d'algorismes per a plataformes tan austeres com la mota:

"The FFT component is not very useful in timer-driven sampling mode since FFT is so slow that it can not keep up the pace with a reasonable sampling rate (e.g.,16Hz). It is shown that the Mica mote takes more than 30 seconds to complete a 512-point FFT, this fact suggests that we should not run FFT in timer-driven sampling mode" (Ning Xu, 4-5).

2.1.2 Revisió dels algoritmes de compressió aplicables

Podem classificar els algorismes de compressió en dos grans grups: sense pèrdua i amb pèrdua.

Els algorismes sense pèrdua aconsegueixen un nivell determinat de compressió en funció de la naturalesa de les dades i de les característiques del propi algorisme. No hi ha pèrdua d'informació perquè la descompressió reconstrueix les dades en la seva integritat. Els més usats són els basats en diccionari, tals com la codificació Huffmann i derivats (CCITT grup 3 i 4), o la més moderna codificació Lempel-Ziv i les seves variants (LZ77, LZ78, LZSS). Els algorismes d'aquests tipus no són utilitzables en el present projecte perquè no permeten una compressió progressiva: no és possible tornar a comprimir les dades ja comprimides. S'ha descartat també la possibilitat d'utilitzar algun d'aquests algorismes com a primer nivell de compressió, és a dir, la primera vegada que és necessari obtenir més memòria s'utilitza un d'aquests algorismes, i a partir de la segona s'utilitza ja un algorisme sense pèrdua. Això presentaria l'avantatge de que retardaria el moment de començar a perdre informació, i per tant, potser en algun cas no seria necessari arriba-hi. Però malauradament l'estratègia és inviable, perquè aquests algorismes converteixen les dades a un format que les torna irreconeixibles sense descomprimir-les prèviament, i per tant no permeten la utilització posterior d'un algorisme amb pèrdua que intenti que aquesta pèrdua sigui la mínima possible.

Per la seva banda els algorismes amb pèrdua donen prioritat a la compressió sobre la integritat de les dades. Alguns d'ells permeten un escalat, és a dir, poden fer-se aplicacions successives variant algun paràmetre i el resultat és una progressiva pèrdua de detalls. Aquesta filosofia coincideix amb els objectius d'aquest projecte, i per tant aquests algorismes han estat els que més s'han analitzat.

Per a poder minimitzar la pèrdua de detalls, els algorismes que estem considerant han de estar adaptats a la naturalesa de les dades a les que s'apliquen (han de poder saber què és un detall i que no). En el nostre cas els que semblen més adequats són els algorismes

per a la compressió de senyals continus. Normalment aquests algoritmes es basen en una transformada: el senyal és primer descompost i després quantificat, eliminant en la quantificació alguns dels components obtinguts. En l'apartat anterior hem vist alguns dels més utilitzats: els basats en la FTT (Fast Fourier Transformation), el 2D Wawelet, i el DPCM (Differential Pulse-Code Modulation).

Ja ha quedat clar que el FTT és excessivament costós en termes computacionals per a ser utilitzar en una mota a causa de l'ús de càlculs en coma flotant. Per contra el 2D Wawelet sí que s'utilitza, com hem vist en projecte SenZip, tot i que el seu cost computacional és també alt. El 2D Wavelet es basa en l'aplicació de la Transformada Wavelet Discreta (DWT) per a obtenir els coeficients necessaris en el procés de quantificació de l'algorisme de compressió. Mentre que la FTT és més adequada per a la compressió de senyals periòdics com audio o vídeo, perquè representa el senyal en ones sinusoïdals, que són periòdiques, la DWT és més adequada per a la compressió d'imatge perquè es tracta de senyals en dues dimensions que no són periòdics, sinó que presenten transicions a causa dels contorns dels objectes. En aquest cas la DWT obté millors resultats, perquè representa el senyal a partir d'elements més simples, les *wavelets*.

El cost computacional del DWT és també alt, tot i que darrerament s'han proposat variants que el fan més i més simple (Lipinski i Yatsymirskyy). En el projecte SenZip es pot utilitzar de manera eficient gràcies a alguns factors específics:

- La compressió es fa sobre paquets de dimensió reduïda.
- La compressió és distribuïda, és a dir, cada node de la xarxa efectua una part de es operacions de l'algorisme.
- Els processadors utilitzats en els casos documentats s'han utilitzat microprocessadors situats en ordre de magnitud superior en comparació amb el del Cou24, com per exemple un StrongARM SA-1100 (Ciancio i Ortega).

Cap d'aquests factors pot ser tingut en compte en el present projecte perquè entre en contradicció amb els seus objectius.

El DPCM s'utilitza principalment per a la codificació i compressió de senyals d'audio. Es basa en guardar no el valor de les mostres sinó el de les seves diferències, amb l'objectiu d'aplicar posteriorment un procediment de compressió sense pèrdua. La idea és que la entropia de les diferències entres senyals és molt més gran que la entropia entre les mostres originals, de manera que l'algorisme de compressió serà molt més

eficaç si actua sobre les diferències que si ho fa sobre les mostres originals. La seva eficiència depèn molt de la redundància present en els senyals analitzats. Quan aquesta redundància és alta, com és el cas dels senyals d'audio, l'algorisme aconsegueix una gran eficàcia, però amb senyals totalment aleatoris fins i tot pot ser contraproduent.
Com a conclusió s'ha de dir que cap dels algorismes analitzats s'ha considerat adequat per a les característiques específiques de la mota, si bé aquests dos últims, el 2D wavelet i el DPCM ofereixen alternatives que potser els faran aplicables en futures evolucions.

2.2 Estudi de mercat

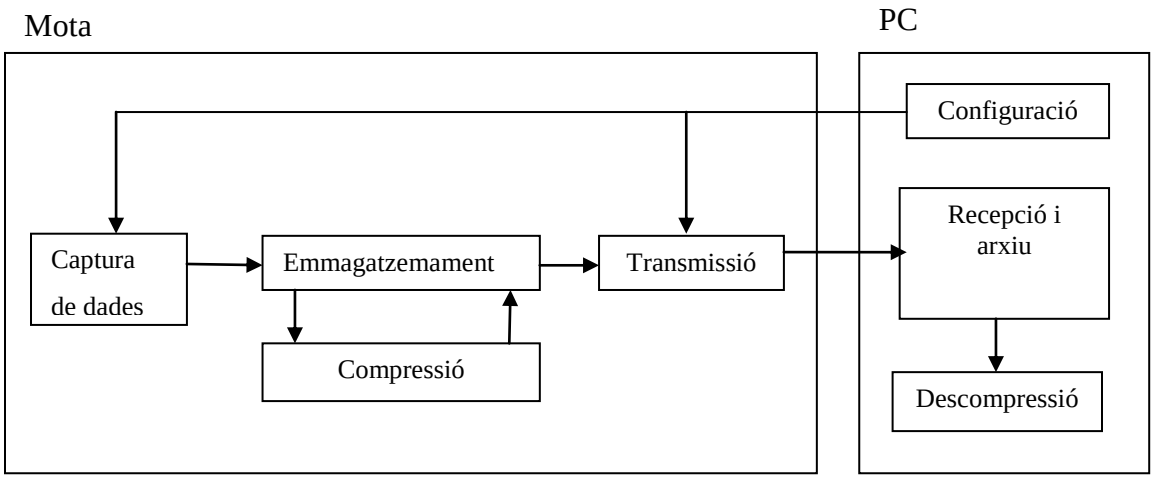
S'ha de dir, en primer lloc, que l'objectiu d'aquest projecte no és el desenvolupament d'un producte final que pugui ser utilitzat directament amb una finalitat específica, sinó més aviat un component que pot ser utilitzat en moltes de les aplicacions habituals de les motes. Podem, en tot cas, restringir el seu l'àmbit d'utilització natural en funció d'algunes de les seves característiques.
La primera restricció ve determinada pel fet que el component desenvolupat només pot aplicar-se a la recol·lecció de mostres a través dels sensors de la mota. Certament aquesta és una de les utilitzacions més característiques de les motes, però no la única.
La segona restricció deriva de l'objectiu principal del sistema de compressió que s'ha dissenyat, que és permetre que la recollida de mostres pugui continuar en absència de comunicacions. Això implica que les aplicacions per a les quals les comunicacions en temps real són imprescindibles, queden també excloses.
En conclusió: el mercat al que va destinat aquest desenvolupament és el de les aplicacions que requereixen mostrejar els sensors a una freqüència fixa i no necessiten comunicacions en temps real.
Però també s'ha de dir que aquest projecte podria obrir camí per a una aplicació de les motes que no és habitual perquè és una mica contradictòria amb la seva naturalesa. Es tracta de motes destinades a capturar dades dels seus sensors en mode autònom (stand-alone). És una aplicació contradictòria amb la naturalesa de la mota perquè una de les característiques essencials d'aquest dispositiu és la capacitat de comunicació, singularment en connexió amb una xarxa sense fils. Però l'abast d'aquesta comunicació sense fils és reduït, i per tant no se'n pot treure profit en ubicacions allunyades d'una infraestructura de comunicacions apropiada. En aquestes circumstàncies la recollida de mostres només pot fer-se de manera esporàdica, i potser no es pot garantir una

determinada periodicitat. Suposem, per exemple, una mota ubicada en un vehicle que només de tant en tant s'apropa a la base de comunicacions, o en un lloc de difícil accés que transcorre molt de temps sense que sigui visitat per algú equipat amb l'instrumental necessari per a fer la recollida de les dades. En aquests casos la memòria de la mota podria arribar a omplir-se al cap d'un temps i deixaria de guardar mostres. El sistema desenvolupat en aquest projecte permetria que, encara que sigui de manera degradada, la mota continuï recollint dades indefinidament.

3. Descripció funcional

3.1 Sistema total

3.1.1 Diagrama de blocs



- **Configuració**: Aplicació que permet configurar els processos d'emmagatzemament i transmissió de la mota.
- **Captura de dades**: Rutina temporitzada que mostreja els valors d'alguns dels sensors a la freqüència configurada. Cada valor llegit s'emmagatzema immediatament.
- **Emmagatzemament**: Els valors llegits es guarden a la memòria del dispositiu utilitzant un sistema que permeti una aplicació fàcil de l'algorisme de compressió i compactació.
- **Transmissió**: Quan hi ha alguna dada emmagatzemada i hi ha connexió amb el PC, la dada s'envia immediatament i s'esborra de la memòria.
- **Compressió**: Monitoritza l'ocupació de l'espai d'emmagatzemament; quan aquest espai baixa d'un cert valor mínim, activa els mecanismes de compressió i compactació.
- **Recepció i arxiu**: Connecta amb una mota, rep les dades que aquesta envia i les guarda en un fitxer.
- **Descompressió**: Processa un fitxer de dades per tal de retornar-les al seu estat original, la qual cosa pot implicar reconstruir-les.

a) Xarxa de comunicacions

Donats els objectius del projecte, la xarxa de comunicacions s'utilitza només com a infraestructura per a transmetre les dades des de la mota al PC.

La implementació realitzada utilitza la comunicació per ràdio per comunicar la mota que recull les dades i una altra mota que fa de pont amb el PC, fent servir el programa **BaseStation**. Aquesta darrera mota comunica amb el PC via sèrie fent servir la utilitat **SerialForwarder**. Però el disseny permet canviar fàcilment aquest esquema, ja que el component que envia les dades des de la mota està desacoblat de la resta de l'aplicació, d'acord amb el sistema propi del TinyOS: s'ha definit una interfície per a enviar les dades i en la configuració del mòdul principal de l'aplicació s'ha fet el *wiring* amb un component que implementa aquesta interfície a través de l'enviament per ràdio. La substitució d'aquest mòdul per un altre que implementi també la interfície però a través d'un altre mitjà seria transparent a la resta de l'aplicació.

3.1.2 Interacció entre els elements

Tal com indica el diagrama de blocs, el PC configura el mode de funcionament de la mota a través d'un missatge i a partir d'aquest moment la mota comença a recollir dades i transmetre-les d'acord amb la configuració rebuda.

Quan el llança l'aplicació de recepció, el PC rep les dades i les guarda en un fitxer. Posteriorment aquest fitxer pot ser processar per a desc0omprimir les dades.

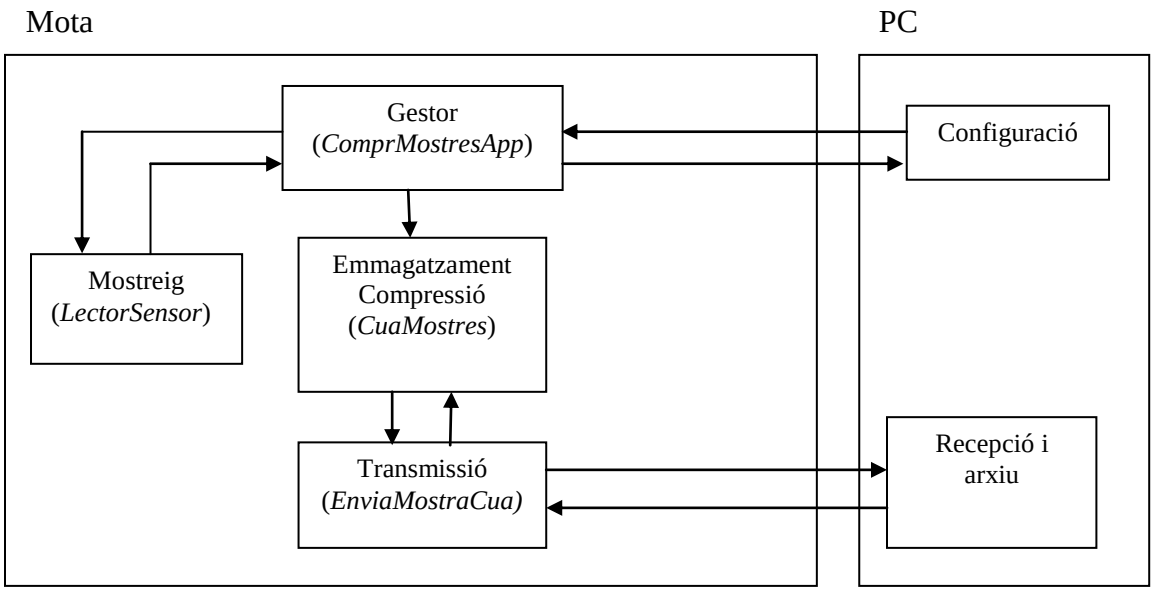
3.2 Interfície d'usuari al PC

Tal com es fa palès al diagrama de blocs del sistema, la interfície d'usuari està formada per 3 aplicacions, que s'han descrit breument a l'apartat 1.7. Donada la simplicitat de la interfície, i per no caure en una reiteració excessiva, només s'assenyala ara en quin cas s'ha d'utilitzar cadascuna d'elles.

La de configuració de la mota, **ConfigSessio**, s'executa puntualment abans de començar una sessió de recollida de mostres. La de recepció, **RecullMostres**, pot estar permanentment executant-se a l'espera de rebre dades. La de descompressió, **DescomprMostres**, s'executa puntualment una vegada finalitzada una sessió de recollida de mostres per tal de descomprimir-les.

3.3 Aplicació de la mota

3.3.1 Diagrama de blocs detallat



A sota del nom de cada bloc s'indica el del component del programa de la mota que l'implementa. Tenint com a referència el diagrama de blocs general, cal fer les següents observacions:

- **Gestor**: S'encarrega de rebre la configuració i enviar la confirmació al PC, de programar el component de lectura dels sensors, de rebre les dades de les lectures i de passar-les al component d'emmagatzemament.
- **Mostreig**: Quan rep l'ordre del gestor, mostreja els sensors amb la freqüència configurada i senyalitza cada lectura al gestor.
- **Emmagatzemament/compressió**: S'han fusionat totes dues funcionalitats en un sol component perquè el procés de compressió necessita un coneixement complet dels detalls del sistema d'emmagatzemament per tal de ser eficient. Aquest component passa les dades, una a una, al procés de transmissió, i no passa la següent fins que no ha rebut el senyal que indica que s'ha enviat l'anterior.
- **Transmissió**: Intenta enviar la dada que li ha passat el procés anterior i reintenta l'enviament fins que no rep la confirmació. Quan això passa ho senyalitza i queda a l'espera de rebre una nova dada per a transmetre.

4. Descripció detallada

4.1 Aspectes més destacables

Els apartats següents aprofundeixen en els aspectes més rellevants de projecte, explicant en detall el disseny i la implementació dels algorismes i les principals decisions de disseny. Aquests aspectes són els següents:

- La compressió de dades és l'objectiu central del projecte, i és, per tant, la part en la que s'han centrat la major part dels esforços. En coherència amb els objectius plantejats l'algorisme de compressió està dissenyat per a comprimir un flux de mostres amb un nivell de compressió progressivament major que pugui incrementar-se indefinidament. Per tant és evident que es tracta d'un sistema de compressió **amb pèrdua**, però que minimitza aquesta pèrdua aprofitant-se del fet que la seqüència de les dades manté una certa lògica (no són aleatòries), a causa del fet que provenen de mostrejar sensors a una freqüència continua.
- Estretament relacionat amb l'algorisme de compressió pròpiament dit hi ha les decisions relatives al sistema d'emmagatzemament de dades i a la seva codificació, que estan fortament condicionades per l'algorisme.
- A causa de que el funcionament bàsic de l'algorisme consisteix a suprimir les mostres que en cada nivell es consideren menys rellevants, s'ha implementat també un sistema de compactació per tal de reaprofitar els espais de memòria que queden lliures. Finalment s'ha arbitrat un mecanisme que decideix en quin moment s'ha d'engegar el procés de compressió i a quin nivell.
- La contra-part de l'algorisme de compressió és el de descompressió, que consisteix bàsicament en la interpolació de les mostres eliminades.
- I per últim també s'ha desenvolupat un mecanisme de comunicació que s'adapti a la resta del sistema i garanteixi la rebuda dels missatges tant per part del la mota com de PC.

4.2 Compressió

4.2.1 Algorisme de compressió

L'objectiu de l'algorisme de compressió és eliminar en cada moment les mostres que menys informació aportin, segons el nivell de compressió que estigui fixat en aquest moment. Per tal de decidir quina és la mostra que menys informació aporta s'ha

considerat la representació gràfica dels valors llegits en funció del temps, i s'ha decidit que si prenem sèries de 3 punts consecutius i els unim format un triangle, la mostra menys informativa serà la que determini el triangle d'àrea menor.

Si tenim la sèrie de 3 mostres (m_1,m_2,m_3) capturades en instants successius, és evident que en el cas trivial que les 3 mostres siguin $m_1=m_2=m_3$ podem eliminar m_2 sense que hi hagi pèrdua d'informació. La sèrie quedaria (m_1,m_{2-3}) i els valors originals podrien reconstruir-se sense pèrdua. La segona mostra hauria de guardar-se amb la informació addicional de que el valor correspon a dos instants successius, i no només a un.

Si les mostres no són iguals però la seva representació gràfica forma una recta, és a dir, el pendent del segment [m_1,m_2] és igual que el pendent del segment [m_2,m_3], igualment podríem eliminar la mostra intermèdia sense pèrdua d'informació. La diferència amb el cas anterior és que ara la reconstrucció de les 3 mostres no seria trivial, sinó que s'hauria d'interpolar m_2 calculant el pendent entre m_1 i m_3 i determinant el valor del punt x per a $y=(m_3-m_1)/2$.

En al resta de casos l'eliminació d'un punt comporta pèrdua d'informació, però si volem minimitzar aquesta pèrdua hauríem de començar per suprimir els punts que s'apartin menys dels dos cassos anteriors. Aquests seran els que s'apartin menys de la recta formada pels dos valors dels extrems. I la manera més fàcil i ràpida de determinar això és calcular l'àrea del triangle format pels 3 punts. Com més petita sigui aquesta àrea més a prop estarà el punt m_2 de la recta formada per m_1,m_3.

D'acord amb aquests principis, l'algorisme de compressió consisteix bàsicament en:

1. Calcular les àrees dels triangles determinats per cada sèrie de 3 punts successius.
2. Eliminar la mostra intermèdia de les sèries que determinin els triangles d'àrea menor.

Quan s'elimina una mostra, però, s'ha de mantenir un comptador del nombre d'unitats de temps que transcorren entre la mostra anterior i la posterior. Si no es fes així es modificaria la freqüència del mostreig. Això s'aconsegueix mitjançant el mètode de codificació que s'explica a l'apartat 4.2.3.

També s'ha dissenyat un algorisme per tal de determinar quin és el valor que fa de llindar per a la compressió (és a dir, el valor de l'àrea per sota de la qual se suprimiran les mostres), i que determina el nivell de compressió a aplicar en cada moment. L'apartat 4.2.5 el descriu.

4.2.2 Emmagatzemament de dades a la mota

S'han considerat dues alternatives: llista encadenada i cua. La primera presenta l'avantatge d'una gran facilitat per a l'eliminació de dades. Per contra suposa un gran cost en l'ús de la memòria, ja que per cada dada s'ha de mantenir un punter a la dada següent. Per la seva banda la cua és el sistema més econòmic des del punt de vista de l'ús de la memòria, però resulta costosa pel que fa a l'eliminació de dades ja que s'ha de compactar per a reaprofitar l'espai alliberat.

La consideració més important a l'hora de prendre la decisió ha estat no entrar en contradicció amb l'objectiu principal del projecte, que és incrementar la capacitat d'emmagatzemament de la mota. Fent servir aquest criteri sembla absurd començar el disseny utilitzant un mètode que d'entrada redueix la capacitat d'emmagatzemament, xocant frontalment amb un dels objectius explícits: que la compressió no es fes més que quan fos estrictament necessari.

S'ha triat, per tant, una cua. L'algorisme que decideix engegar la compressió actua quan aquesta cua ha arribat a un determinat nivell d'ocupació. Quan arranca la compressió, l'algoritme va copiant les dades a mesura que recorre la cua per tal d'omplir els espais que hagin quedat buits.

4.2.3 Mètode de codificació

El convertidor A/D de la mota entrega els valors amb 10 dígits significatius. Donada la dificultat per manipular directament valors de 10 bits s'ha optat per emmagatzemar cada lectura en 16 bits, aprofitant els 10 bits restants per a guardar informació relativa a la compressió. Per a mantenir la claredat i coherència a partir d'ara anomenarem **lectura** al valor de 10 bits que conté una lectura d'un sensor, mentre que utilitzarem el terme **paraula** per referir-nos a cada unitat d'informació de 16 bits que es guarda a la cua, i que pot contenir informació sobre una lectura a més d'altra informació.

Com a resultat de la compressió una lectura pot requerir més d'una paraula per a ser emmagatzemada. Anomenarem **paquet** a l'element que conté tota la informació sobre una lectura. Un paquet contindrà sempre una paraula de tipus **mostra**, que conté la lectura, però en el cas de el nivell de compressió sigui molt alt pot contenir també una paraula de tipus **tempsAdd**, que únicament contindrà informació sobre les unitats de temps que han transcorregut des de la lectura anterior.

D'acord amb aquestes definicions, la codificació es fa de la següent manera.

- Una paraula de tipus **mostra** està formada pels següents camps:

Bits	Nom	Significat	Valor
15	conteLectura	Indica que la paraula és de tipus mostra, i per tant conté una lectura.	1
14-10	temps	Nombre d'unitats de temps que han transcorregut des de la lectura anterior. La primera està implícita, i per tant el rang de valors que pot codificar és de 1 a 2^5 = de 1 a 32.	
9-0	lectura	Valor de la lectura	

- Una paraula de tipus **tempsAdd** està formada pels camps:

Bits	Nom	Significat	Valor
15	conteLectura	Indica que la paraula és de tipus tempsAdd, i per tant no conté una lectura.	0
14-10	incrTemps	Nombre d'unitats de temps addicionals que han transcorregut des de la lectura anterior (a més de les 32 del camp temps de la paraula de tipus mostra del paquet).	

Per tant les paraules de tipus *tempsAdd* només seran necessàries a partir del moment en que les unitats de temps transcorregudes enter una mostra i la següent siguin més grans de 32, desbordant la capacitat del camp temps de la paraula de tipus mostra. Si el camp *incrTemps* també arriba a desbordar-se (el valor que ha de contenir és superior a 2^{15}-1 = 32767) s'afegeix una paraula addicional de tipus mostra, i així successivament.

Aquesta codificació garanteix que qualsevol paraula d'un paquet conté un valor diferent de 0: una paraula de tipus mostra té el bit de més pes a 1, i una paraula de tipus *tempsAdd* conté com a mínim una unitat de temps. Per tant tota paraula que conté el valor 0 és una paraula que ha quedat buida com a resultat de la compressió, i per tant es pot compactar.

La contrapartida és que qualsevol paraula s'ha de descodificar, fins i tot en el cas que únicament contingui una lectura. Per tant qualsevol fitxer recollit s'ha de passar pel procés de descompressió.

4.2.4 Compactació

Tot i que les posicions de memòria corresponents a paquets esborrats s'omplen amb 0 i per tant no es poden confondre amb cap paquet vàlid, la utilització d'una cua comporta la necessitat de consolidar tot l'espai lliure en posicions lògicament posteriors al punter d'escriptura. Recordem que en una cua el punter d'escriptura apunta a la posició immediatament posterior a la darrera utilitzada, i el procés d'afegir una nova dada consisteix en escriure-la en la posició apuntada per aquest punter i incrementar-lo a continuació.

Van contemplar-se dues possibilitats per a la compactació de la memòria que ha quedat alliberada enmig de la cua:

1. Un procés de compactació que s'executés després del procés de compressió, i comprimís tota la cua en una sola passada.
2. Combinar la compactació amb la compressió, de forma que cada vegada se suprimeix un paquet es copiï la següent en les posicions que han quedat lliures.

El cost és el mateix en els dos cassos, perquè s'hauran de copiar el mateix nombre de bytes. S'ha implementat la segona possibilitat, la combinada, perquè simplifica una mica l'algorisme bàsic de compressió, que analitza 3 paquets consecutius i, si és necessari, suprimeix l'intermedi. Si la compactació es fes al final del procés, l'anàlisi s'hauria de fer entre 3 paquets que podrien ser no consecutius. El que s'ha implementat és que immediatament després d'esborrar el paquet intermedi d'un grup de 3 es copia el 3r a la posició que ocupava el 2n, i a continuació es copia el paquet següent a la posició que ocupava el 3r. D'aquesta manera tornem a tenir un grup de 3 paquets consecutius, i al final del procés no quedaran posicions buides enmig de la cua.

El sistema de codificació garanteix que mai serà necessari afegir cap paraula com a conseqüència de la compressió, ja que:

1. La compressió deixa com a mínim una paraula lliure, ja que suprimeix un paquet i aquest paquet ocupa com a mínim una paraula.
2. El paquet modificat com a resultat de la compressió pot créixer com a molt en una paraula, ja que si l'increment en les unitats de temps que conté el paquet provoca desbordament, això provocarà que s'afegeixi una nova paraula i només una.

Per tant el resultat de la compressió pot ser que no quedi lliure cap paraula, o que quedi lliure una o més.

4.2.5 Procediment de decisió

a) Decisió sobre el moment d'arrancar la compressió

El moment d'arrancar el procés de compressió s'ha de determinar tenint en compte que el temps que trigarà la compressió no pot ser major que el temps que resta per a que s'ompli la cua.

Càlcul del temps que resta per a que s'ompli la cua

Aquest temps es pot calcular de forma determinista, ja que la freqüència de mostreig és fixa. Serà per a cada cua igual al producte del període de mostreig (temps entre lectures successives) pel nombre de posicions lliures.

Càlcul del temps que trigarà la compressió de la cua

Aquest temps no es pot calcular de forma determinista, ja que depèn de molts factors variables, però sobre tot del nombre d'operacions de compressió que efectivament es dugui a terme. El que s'ha fet és un càlcul heurístic en base a diverses proves amb cues diferents en les que s'ha mesurat el temps invertit en cada compressió. Aquestes proves han donat valors de entre 1,3 i 1,8 milisegons per cada element de la cua. Per a major seguretat s'ha considerat que el temps de compressió d'una mostra és de 2 milisegons.

Tot i que el valor de 2 milisegons per mostra es pot considerar alt, de fet no imposa restriccions massa importants al algoritme. La limitació que en resulta és la freqüència mínima de mostreig, per sota de la qual no hi ha temps suficient per a dur a terme la compressió. Si prenem com a referència una cua de 1000 elements, caldrien 2x1000=2000 ms per a comprimir-la, i per tant no es podria mostrejar amb períodes per sota de 2 segons, que no és una xifra massa restrictiva per a la major part de les aplicacions.

Cal remarcar, però que poden tractar-se fins a 3 cues simultàniament, i per tant la xifra anterior s'ha de multiplicar pel nombre de cues que s'utilitzin. Per tant en el pitjor dels casos, amb 3 cues de 1000 elements (que sembla el límit raonable per la capacitat de la mota) seria el període mínim de mostreig de 6 segons.

En base als càlculs anteriors, el procés de compressió s'ha d'iniciar com a molt tard quan el producte del període de mostreig en milisegons pel nombre de posicions lliures sigui el doble del nombre d'elements ocupats de la cua.

El càlcul que fa el programa és el següent (funció *fixarPeriode* del component *cuaMostresP*):

1. Calcula el nombre d'elements que pot processar el procés de compressió entre lectures successives, és a dir, en un període de mostreig. Ho fa dividint el període en ms pel nombre de ms que costa processar un element, que hem vist que és 2.
2. Divideix el nombre d'elements de la cua per aquest valor, amb la qual cosa obté el nombre de períodes necessari per a comprimir tota la cua. Per tant el procés de compressió s'haurà d'engegar quan el nombre de posicions lliures de la cua sigui igual a aquest valor, que s'anomena **minimEspaiLliure**.

b) Decisió sobre el nivell de compressió

És necessari assegurar que el procés de compressió alliberarà prou espai per a garantir que quan acabi tornaran a haver-hi prou elements lliures perquè doni temps a dur a terme un nou procés de compressió. Això s'aconsegueix comprimit utilitzant el nivell de compressió adequat. Hem vist que el nivell de compressió és el valor de l'àrea del triangle que es pren com a llindar, de forma que totes les mostres que determinin triangles amb àrea inferior a aquest valor se suprimiran.

Per a decidir el nivell de compressió que s'ha d'aplicar, el programa utilitza el següent algorisme:

1. Cada vegada que s'afegeix un element a la cua, la funció d'inserció (*afegeix*, del component *CuaMostresP*) calcula l'àrea del triangle que determina aquesta mostra i les dues anteriors.
2. El programa manté una taula on es guarda el nombre de triangles que tenen an àrea compresa entre un valor mínim i un valor màxim. En la implementació actual aquesta taula (*afegeix*, del component *CuaMostresP*) té 5 nivells, i en cadascun d'ells es guarden els valors compresos en un interval de 5 unitats. És a dir que d'entrada aquesta taula contindrà en el primer element el nombre de triangles d'àrea entre 0 i 5, en el segon el nombre de triangles d'àrea entre 5 i 10, i així successivament.
3. Quan el nombre de posicions lliures de la cua és menor o igual que *minimEspaiLliure*, s'ha de posar en marxa la compressió. Abans de fer-ho el programa consulta la taula per a determinar el nivell de compressió a aplicar. Aplicarà el nivell per sota del qual hi hagi un nombre de triangles major o igual

a *minimEspaiLliure*, garantint així que quan acabi el procés tornarà a haver suficient espai lliure. Per exemple, si la taula conté els valors:

<5	<10	<15	<20	<25
3	4	7	6	4

I si *minimEspaiLliure* és 10, el nivell de compressió que s'aplicarà serà 15, ja que la suma dels triangles amb àrea menor que 15 és 11. Per tant tenim la certesa de que si se suprimeixen totes les mostres que determinen un triangle d'àrea inferior a 15, acabarà el procés de compressió amb 11 posicions lliures.

Es pot argumentar que la supressió d'un paquet no necessàriament allibera una paraula, com ja hem vist, però hi ha tota una sèrie d'arrodoniments en els càlculs que donen un cert marge de seguretat. Per exemple, els càlculs es fan com si s'hagués de comprimir tota la cua, però de fet això no es fa mai: sempre hi ha *minimEspaiLliure* elements que no es comprimeixen perquè no contenen paquets.

4. La taula de nivells és mòbil i s'inicia en el nivell de compressió vigent en cada moment. Per tant, una vegada s'ha dut a terme el procés de compressió, els nivells ja comprimits s'eliminen i s'afegeixen els consecutius al que era fina ara l'últim. Per exemple, en el cas anterior despès d'efectuar la compressió a nivell 15 la taula quedaria:

<20	<25	<30	<35	<40
6	4	0	0	0

I continuaria actualitzant-se cada vegada que s'afegeix una nova mostra a la cua.

4.3 Descompressió

Bàsicament el procés de descompressió reconstrueix les mostres interpolant els punts suprimits. La implementació es troba en la classe en *app/classesAux/ MaquinaDescomprimir*. En detall, l'algorisme funciona de la següent manera:

1. El primer paquet no està comprimit. Això queda garantit pel disseny de l'algorisme de compressió, però si no fos així (per exemple perquè manquin dades de l'inici) no hi hauria possibilitat de reconstruir-lo sense saber el valor de la lectura anterior. Per tant del primer simplement s'extreu el camp *lectura* i es retorna.

2. A partir del segon paquet, amb cada paraula nova de tipus *mostra* es fa:
    a. Si no hi ha una mostra anterior pendent (només passarà en el segon paquet), simplement s'extreuen el valors dels camps *lectura* i *temps* i es guarden.
    b. Si hi ha una mostra pendent anterior:
        i. Es reconstrueixen els valors entre la lectura anterior i l'actual. Per a fer això primer es calcula el pendent, dividint la diferència entre el valor de la lectura anterior i la actual pel temps acumulat, i després es genera un nombre de mostres igual al d'unitats de temps a base de sumar al valor de la lectura anterior el producte del pendent per la posició (nombre d'unitats de temps transcorregudes fins a la lectura en qüestió).
        ii. Es retorna la llista de lectures generades
        iii. Es guarden els valors dels camps *lectura* i *temps* de la mostra actual.
3. Si el paquet és de tipus *tempsAdd*, simplement s'acumula al temps guardat el valor del camp *tempsIncr*.

4.4 Comunicació

4.4.1 Fiabilitat

Com que la infraestrucura de comunicacions que ofereix TinyOS no ofereix garantia de lliurament dels missatges, s'ha previst un sistema per a assegurar la recepció dels missatges, que consisteix simplement en que per a cada missatge hi ha el corresponent missatge de confirmació, i el dispositiu emissor reintenta l'enviament periòdicament fins que rep la confirmació de la recepció.

L'associació entre el missatge enviat i la confirmació es fa via un número d'identificació que forma part de tots dos missatges. Un missatge enviat es considera rebut quan ha arribat la confirmació corresponent amb el mateix número de missatge

4.4.2 Missatges

Només hi ha dos intercanvis de missatges, cadascun amb la seva confirmació:

Missatge	Sentit
ConfigSessio	PC a mota
AckConfig	Mota a PC
MostraCua	PC a Mota
AckMostra	Mota a PC

En la implementació s'han multiplicat per 3 el nombre de missatges de tipus *MostraCua* (i per tant també els de tipus *AckMostraCua*), a causa de que el mecanisme *ActiveMessage* del TinyOS té un buffer de una unitat per a cada tipus de missatge. Si s'hagués utilitzat un únic missatge amb un camp que indiqués a quina cua corresponia, el buffer hauria estat molt sovint ple i el funcionament s'hauria degradat a causa dels reintents.

4.5 Consideracions sobre la implementació

4.5.1 Modularitat

L'aplicació de la mota s'ha dissenyat seguint les directrius marcades per la filosofia del TinyOS. Per a cada funcionalitat s'ha definit un *interface*, s'ha creat un component que l'implementa i una configuració que els relaciona.

Els components dissenyats tenen un grau d'acoblament molt baix i una interfície senzilla, i per tant seria senzill substituir, per exemple, el mòdul gestor de cues per un altre, o fer que les cues enviessin els missatges amb les mostres de manera diferent.

Aquestes directrius generals no s'han pogut respectar del tot en algun cas que es comenta en els apartats següents.

4.5.2 Gestió de les cues

Com que la mota disposa de 3 sensors, el sistema de compressió ha de disposar de 3 cues. La implementació s'ha fet amb un component genèric, que es crea 3 vegades en la configuració del component principal. Hauria estat més elegant un vector de components, però la filosofia del TinyOS condueix a fer-ho d'aquesta manera, que és més eficient en termes d'execució però més costosa en utilització de la memòria de programa.

El fet de que les cues estiguin implementades en un component genèric obliga a que també sigui genèric el component que envia els missatges amb els paquets. Per tant se'n

creen també 3 en la configuració del component principal. Això té la conseqüència de que s'ha de definir també 3 missatges diferents, un per cada cua, i 3 missatges diferents de confirmació. Certament s'hauria pogut fer amb un sol missatge, però es perdrien algunes de les avantatges que proporciona el sistema de gestió de missatges del TinyOS (i també les que proporciona la utilitat *mig* a l'hora de generar les classes Java per a manegar els missatges des del PC).

S'ha implementat una vinculació dinàmica entre cada cua i el sensor del que emmagatzema les dades. Aquesta característica seria prescindible en una utilització real del programa, però s'ha vist la seva conveniència per a poder fer les proves comparatives entre dades comprimides i sense comprimir: s'ha de poder permetre que cues diferents emmagatzemin les dades del mateix sensor. Això impedeix fer la connexió entre els components *CuaMostresP* i el component *LectorSensor* a través del *wiring*. Fent-ho d'aquesta manera *LectorSensor* enviaria les lectures directament a les cues a través d'un senyal. Tal com s'ha implementat el component *LectorSensor* està connectat amb el component principal de l'aplicació, i és aquest qui fa arribar la dada a la cua o cues que l'emmagatzemen.

4.5.3 Tasques

S'han utilitzat abastament les tasques de TinyOS. En particular això ha permès que el procés de compressió no bloquegi el processador mentre s'executa. Aquest procés s'ha dissenyat com a una *task*, fent que comprimeixi i compacti un paquet cada vegada que s'executa. Això permet, per exemple, que el mostreig pugui continuar a la freqüència programada mentre s'està duent a terme la compressió.

5. Proves i resultats

1.1 Descripció dels jocs de proves

1. Objectiu: Comprovar que els valors guardats en la cua són idèntics a als valors originals abans de la compressió.

   Implementació: Enviament directe de les lectures abans de afegir-les a la cua, i comparació posterior de les dades rebudes de la cua. Aquest enviament es fa utilitzant un missatge genèric de debug.
2. Objectiu: Comprovar que els valors guardats en una cua abans de que arribi a actuar el procés de compressió són idèntics als valors enviats directament a mesura que van afegint-se a la cua.

   Implementació: Configurar dues cues per a que guardin valors del mateix sensor, i una els enviï directament mentre que l'altra els guarda. Quan la 2a està a punt d'omplir-se, es configura perquè enviï les dades, i es comparen les dades rebudes de la 1a cua amb les rebudes de la 2a.
3. Objectiu: Compara valors comprimits amb els originals, amb diferents nivells de compressió.

   Implementació: Configurar sensors per a que guardin valors del mateix sensor, però només la 1a enviï directament. Quan hagi transcorregut un temps t_1 canviar la configuració per a que la 2a també enviï. I a partir d'un temps $t_2 > t_1$ canviar novament la configuració per a que totes 3 enviïn, fins que els valors que arribin siguin els mateixos. A partir del nombre de dades de la primera cua que s'han rebut a t_1 i a t_2 es pot determinar quantes mostres s'han de considerar per a avaluar la compressió de la cua 2 i quantes per a avaluar la compressió de la cua 3.

1.2 Resultats

En les proves 1 i 2 s'ha comprovat que els resultats són els esperats.

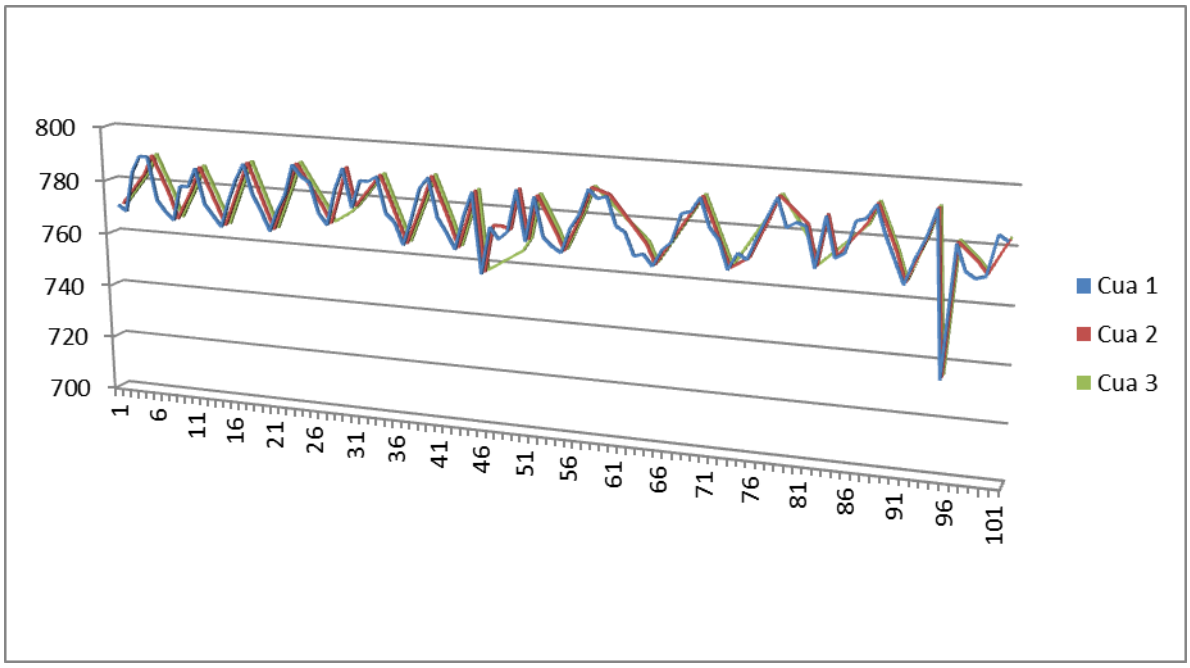
L'objectiu de la prova 3 és avaluar la bondat de l'algorisme de compressió, comparant les mostres comprimides amb diferents nivells de compressió i mesurant en quin grau les comprimides s'aparten de les originals. Aquesta comparació s'ha fet de dues maneres: qualitativament, a través de les representacions gràfiques de les mostres, i

qualitativament, a partir de la comparació aritmètica de les lectures comprimides amb les reals.

Per a la prova 3 s'han fet 4 sèries de preses de mostres, utilitzant sempre el sensor de llum, ja que la llum és la magnitud més fàcil de variar a voluntat.

1.2.1 Comparació gràfica

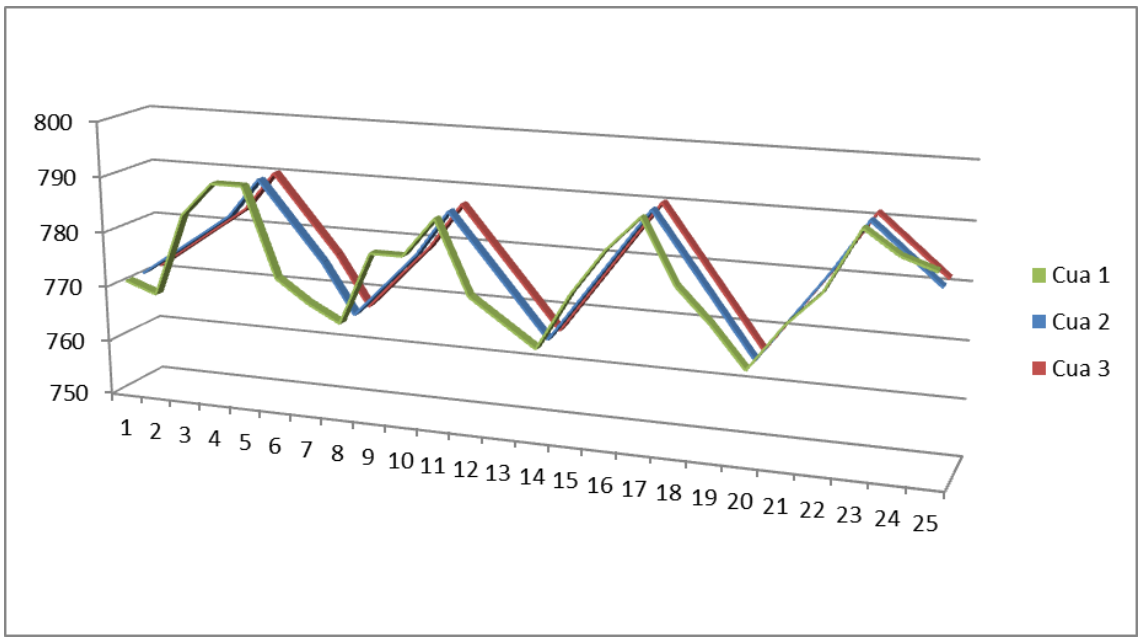
La comparació visual de les gràfiques de les mostres comprimides amb les no comprimides mostra si la forma de les gràfiques es bàsicament igual, és a dir, si es compleix l'objectiu de que la compressió només elimini detalls poc importants. En general el resultat d'aquesta comparació es acceptable, en el sentit de que no es detecten anomalies evidents. Veiem a continuació la gràfica corresponent a un tram de 100 mostres de la sèrie 4. La cua 1 no està comprimida, la 2 sí que ho està i la 3 ho està més. S'ha utilitzat una gràfica de línies, i es visualitza en 3 dimensions per a evitar un solapament excessiu.



Gràfica 1 – Tram de 100 mostres de la sèrie 4

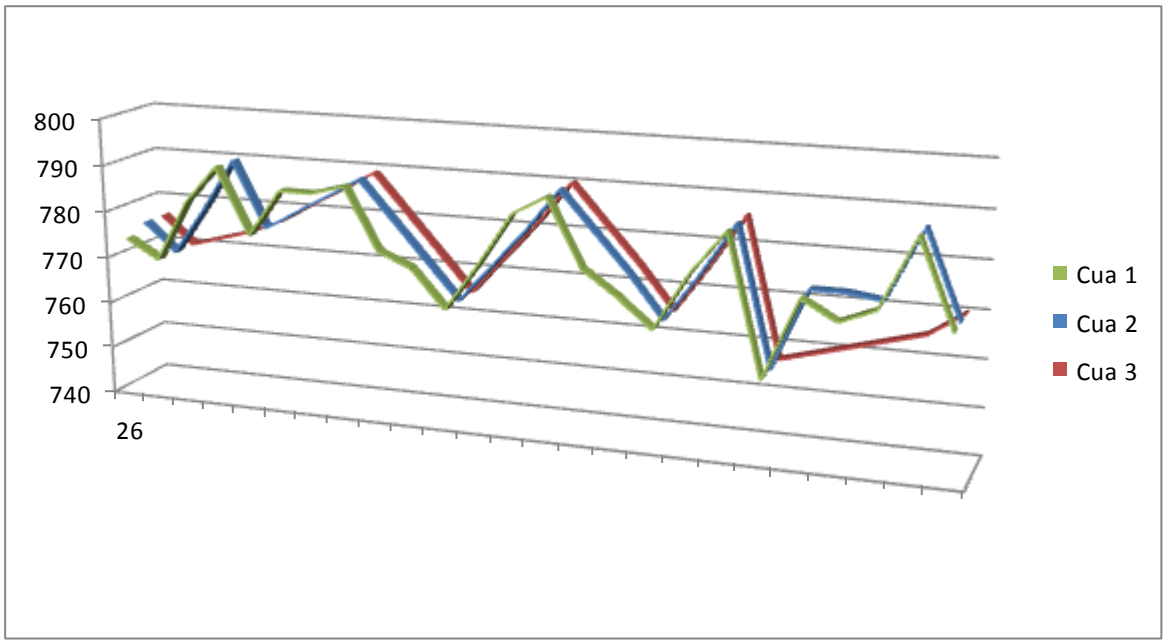
Tal com és d'esperar la línia corresponent a la cua no comprimida és la que presenta més irregularitats, mentre que la corresponent a la cua conté més segments rectes. Veiem també, però, que els valors màxims i mínims es mantenen en general.

Am la finalitat d'analitzar el tram en més detall, el dividirem en 4 parts.



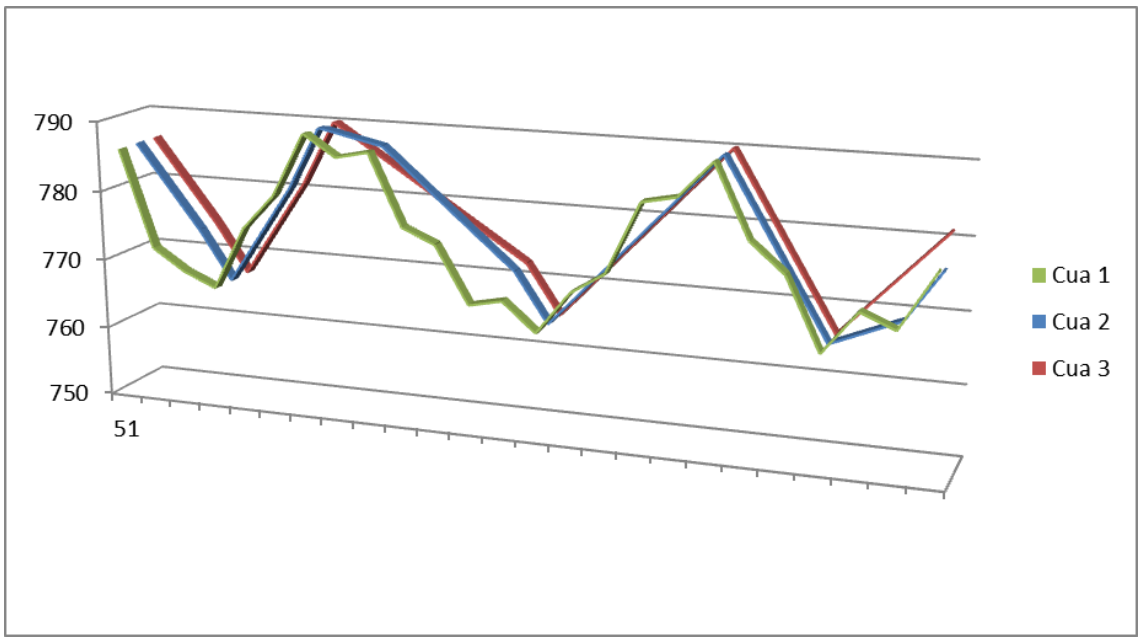
Gràfica 2 – Tram de mostres 1 a 25 de la sèrie 4

Veiem que es respecten les 4 sinusoides, tot i que en la 1ª i 2ª, que són més irregulars, la compressió fa perdre detalls.



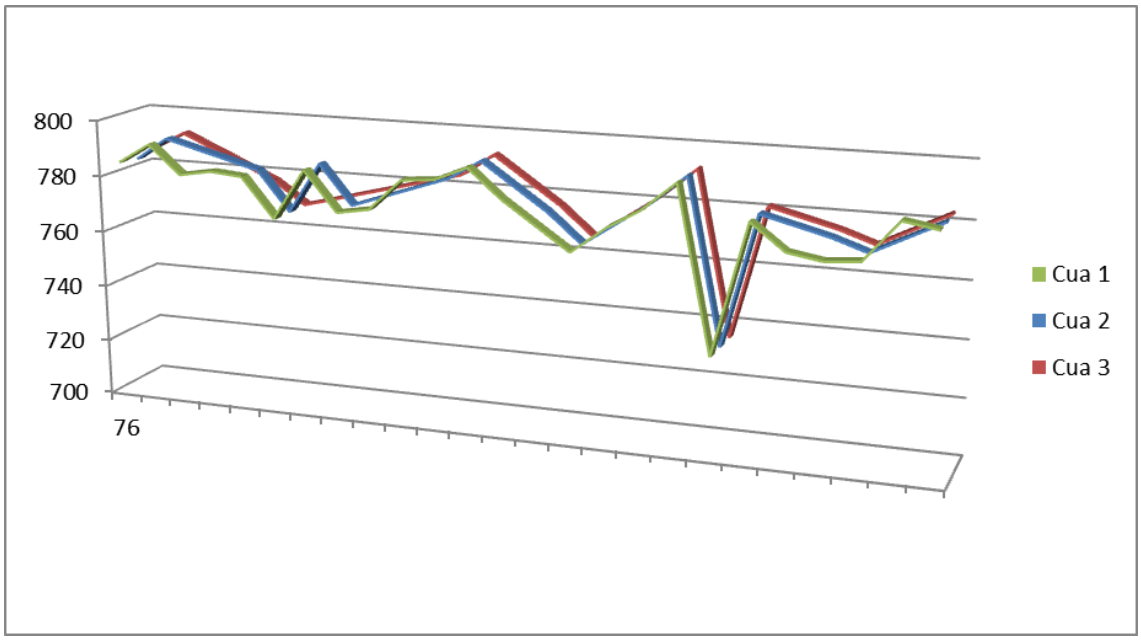
Gràfica 3 – Tram de mostres 26 a 50 de la sèrie 4

Aquí observem que les dades menys comprimides, corresponents a la cua 2, respecten els pics, però les més comprimides (cua 3) han fet desaparèixer el primer i l'últim.



Gràfica 4 – Tram de mostres 51 a 75 de la sèrie 4

En aquest tram veieim únicament una pèrdua de detalls, però es manté la forma general.



Gràfica 2 – Tram de mostres 76 a 100 de la sèrie 4

En aquest tram és en el que observem una menor pèrdua de detalls. Això s'explica perquè les dades originals (cua 1) a penes tenen oscil·lacions petites.

1.2.2 Comparació quantitativa

La comparació aritmètica de les mostres comprimides amb les no comprimides permet avaluar amb precisió la desviació de les dades comprimides respecte de les originals, i permet també establir comparacions objectives entre el resultat de la compressió a diferents nivells. S'han fet dos càlculs:

- La suma del valor absolut de les diferències dividit pel nombre de mostres. És a dir: per a cada lectura de la mostra original se li resta el valor de la lectura descomprimida. Aquesta dada en valor absolut ens diu quant s'aparta la mostra comprimida de l'original. El sumatori de tots aquests valors dividit pel nombre de mostres ens dóna la mitjana de desviacions, que podem considerar com a indicador de fidelitat de la compressió. Com més s'apropi a cero, més bona haurà estat la reconstrucció. La taula següent conté els resultats obtinguts en les 4 sèries:

	Cua 1 – cua 2	Cua 1 – cua 3
Sèrie 1	4,709486166	6,335968379
Sèrie 2	7,720930233	12,96899225
Sèrie 3	6,053956835	8,190647482
Sèrie 4	2,50990099	3,920792079

  Tot i que és difícil fer una valoració acurada d'aquests resultats, el que resulta evident és que no hi ha anomalies significatives. No hi ha cap valor destacable, i s'incrementen quan el nivell de compressió augmenta.

- La suma del valor de les diferències (amb signe) dividit pel nombre de mostres. Aquest és un indicador de la simetria de l'algorisme de compressió. Per a una mostra prou gran hauria de ser 0, independentment del nivell de compressió, ja que les desviacions en un sentit haurien de compensar-se amb les desviacions en l'altre sentit i, per tant, els valors positius haurien d'anul·lar els negatius. Si no és així l'algorisme és esbiaixat en algun sentit. La taula següent conté els resultats obtinguts en les 4 sèries:

	Cua 1 – cua 2	Cua 1 – cua 3
Sèrie 1	0,333992095	1,43083004
Sèrie 2	-4,139534884	-1,744186047
Sèrie 3	0,787769784	1,248201439
Sèrie 4	-0,618811881	0,198019802

No s'observa un biaix clar en la compressió. Hi ha casos amb signe positiu i altres amb signe negatiu. En 3 dels 4 casos el signe és el mateix per als 2 diferents nivells de compressió, la qual cosa sembla indicar que depèn més de les característiques de les dades que de l'algorisme.

6. Viabilitat tècnica

La implementació real del sistema dissenyat demostra la seva viabilitat tècnica. S'han de considerar, però, els següents aspectes:

- Ús de memòria de programa: pot considerar-se com a moderada. En la versió definitiva és de 22.864 bytes. Es reduiria considerablement si el component *CuaMostresP*, que és el més gran, no fos un component genèric i, per tant, el seu codi no estigués triplicat. No obstant això la memòria de programa que resta (pràcticament 6 KB) encara permet encabir-hi molt codi.
- Ús de la memòria RAM: és reduït (uns 650 bytes), deixant de banda l'espai reservat a les cues.
- Ús del processador: és més aviat alt en el moment de fer la compressió. Aquesta triga, com ja s'ha dit, 1,3 i 1,8 milisegons per cada element. Això repercuteix en diversos aspectes:
    - Pot alentir altres processos que hagi d'executar la mota quan el component de compressió formi part d'una altra aplicació.
    - Limita la freqüència de mostreig.
    - Incrementa el consum perquè manté actiu el processador durant un període prolongat.

  Aquestes conseqüències no invaliden l'aplicabilitat del component desenvolupat però s'han de prendre en consideració abans de decidir utilitzar-lo.
- Limitació en la freqüència de mostreig, com s'ha explicat a l'apartat 4.2.5. És també un condicionant a considerar en funció dels requeriments de la aplicació.

7. Conclusions finals

7.1 Proposta de millores

Podem pensar en dos tipus de millores:

1. Les que no afecte l'essència del projecte però milloren la implementació en alguns aspectes.
2. Es camins alternatius que podrien seguir-se en aspectes importants, de forma que el projecte resultant seria significativament diferent.

Dintre del primer grup (modificacions de detall) poden considerar les següents:

1. Convertir les lectures als valors adequats en funció de la magnitud mesurada pels sensors. No és part essencial del projecte, ja que la compressió es fa directament sobre els valors bruts de les lectures i és independent de quina magnitud representen. Però és convenient per a un ús real del paquet. Es podria incloure en l'aplicació de descompressió.
2. Rebre la parametrització amb la que ha treballat la mota per a fer el mostreig (període, sensors). Actualment es pot incloure en l'identificador de la configuració, codificat per l'usuari, però podria fer-se de manera automàtica i incloure aqueta informació en el fitxer de recollida.
3. Rebre informació sobre el nivell de compressió que s'ha utilitzat cada vegada que es rebin dades comprimides. Aquesta informació permet saber si les dades que s'han rebut són reals o no, i si no ho son fer-se una idea de quin és el seu grau de fidelitat.

Dintre del segon grup (modificacions substancials) podem considerar:

1. Implementar un altre algorisme de compressió i comparar eficiència i fidelitat. En l'anàlisi dels algorismes ha quedat clar que el millor candidat és el 2D wavelet, pel qual existeix ja una implementació per a TinyOS (encara que en xarxa). Com que l'algorisme és àmpliament utilitzat, no es pot posar en dubte la seva bondat, però sí que seria interessant comprovar, per exemple, quina és la freqüència màxima de mostreig que permet en una mota Cou24.
2. Fer possible que la memòria ocupada per les cues pogués configurar-se dinàmicament. En un sistema operatiu amb gestió de memòria dinàmica la implementació seria trivial, però en TinyOS no resulta fàcil, i requeriria un

estudi específic les possibilitats existents. Potser no és una característica imprescindible, ja que TinyOS tampoc és un sistema de propòsit general, i n'hi ha prou en determinar aquest paràmetre en tems de compilació. Però fins i tot en aquest cas caldria una modificació (que ja seria menor) per tal de que no fos necessari a modificar el codi font del component que gestiona la cua. S'hauria de permetre que la mida de la cua s'establis en el moment de fer el "wiring" del component.

7.2 Autoavaluació

Amb posterioritat al lliurament del codi s'han detectat alguns bugs:

1. En el PC, en el procés de descompressió hi ha un error en el tractament de l'últim paquet, si està comprimit. En aquest cas la classe *MaquinaDescomprimir* genera una sèrie de valors, i el procés que l'ha de guardar tanca el fitxer després d'haver escrit el primer valor. Per tant la resta de valors no s'arriba a guardar.
2. En la mota, en el procés de compressió de mostres, quan es crear una nova paraula de tipus *tempsIncr*, el procés de compactació no funciona correctament i deixa una paraula amb un 0 després del paquet. L'error no té transcendència perquè el procés de descompressió considera aquesta paraula com un increment de temps 0 i per tant no la té en compte.
3. En la mota, en la crida que fa el component *CuaMostres* a *EnviaMostraCua* per a enviar una paraula, no es comprova el valor de retorn. És un error greu, que pot provocar que si *EnviaMostraCua* està enviant en el moment en que la cua vol enviar un altre valor, el valor que vol enviar la cua es perdi.

També s'ha de destacar que la valoració dels resultats de la compressió ha de ser més acurada en termes matemàtics. L'indicador de fidelitat que s'ha obtingut s'hauria de posar en relació amb altres paràmetres. El més interessant seria el nivell de compressió utilitzat, però això no és possible perquè aquesta dada no és transmesa per la mota. Però hi ha com a mínim dos paràmetres que sí que es coneixen i no s'han considerat: l'index de compressió obtingut en cada cas, calculat com a relació entre el nombre de mostres sense comprimir i el nombre de mostres resultant de la compressió, i el rang de valors en el que es mouen les dades.

Malgrat aquests errors i insuficiències també s'ha de dir que la feina que s`ha fet ha estat intensa i curosa. Els objectius s'han assolit: s'ha implementat un sistema de

compressió de mostres que pot ser aplicat indefinidament i que va eliminant detalls però manté en la mesura del possible l'evolució de la magnitud mostrejada. S'ha dissenyat un algorisme que complís tots els requeriments, s'ha implementat en la mota i s'ha generat tot el programari necessari per a utilitzar-lo.

8. Glossari

conteLectura: camp d'una paraula que indica si és del tipus *mostra* o de tipus *tempsAdd*.

incrTemps: camp d'una paraula de tipus *tempsAdd* que conté informació sobre les unitats de temps transcorregudes entre la lectura anterior i l'actual.

Lectura: En una paraula de tipus *mostra*, valor de 10 bits que conté una lectura d'un sensor.

Mostra (quan el terme es refereix específicament a la codificació de les dades): Tipus de paraula que conté una lectura.

Paquet: Element que conté tota la informació sobre una lectura. Un paquet contindrà sempre una paraula de tipus **mostra**, que conté la lectura, però en el cas de el nivell de compressió sigui molt alt pot contenir també una paraula de tipus **tempsAdd**, que únicament contindrà informació sobre les unitats de temps que han transcorregut des de la lectura anterior.

Paraula: Cada unitat d'informació de 16 bits que es guarda a la cua, i que pot contenir informació sobre una lectura a més d'altra informació.

Temps (quan el terme es refereix específicament a la codificació de les dades): camp d'una paraula de tipus *mostra* que conté informació sobre les unitats de temps transcorregudes entre la lectura anterior i l'actual.

tempsAdd: tipus de paraula que només conté informació sobre unitats de temps.

9. Bibliografia

Projectes de compressió en TinyOS:

Web del projecte SenZip:
http://anrg.usc.edu/SenZip/index.html
Presentació del projecte SenZip:
http://anrg.usc.edu/~pattem/presentations/SenZipMay09.pdf
Descripció detallada del projecte SenZip:
SenZip: An Architecture for Distributed En-route Compression in Wireless Sensor Networks
(disponible en http://biron.usc.edu/~godwinsh/Papers/ESSA09.pdf)

Ning Xu, *Implementation of Data Compression and FFT on TinyOS*
(disponible en http://enl.usc.edu/~om_p/lzfft.pdf)

Algorismes de compressió:

H.V. Sorensen, D.L. Jones, M.T. Heideman and C. S. Burrus, *Real-valued fast Fourier trans-form algorithms, IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35 (1987), pp. 849–863.

Alexandre Ciancio and Antonio Ortega, *A distributed wavelet compression algorithm for wireless multihop sensor network using lífting*, University of Southern California, Signal and Image Processing Institute.
(disponible en http://sipi.usc.edu/~ortega/Papers/CiancioICASSP05.pdf)

Piotr Lipinski i Mykhaylo Yatsymirskyy, *New Algorithm For Calculating Wavelet Transforms*.
(disponible en http://www.iiisci.org/journal/CV$/sci/pdfs/ZS372XE.pdf)

DPCM

http://www.seas.gwu.edu/~ayoussef/papers/LossyDPCM-ISCA98

http://einstein.informatik.uni-oldenburg.de/rechnernetze/dpcm.htm

10. Annexos

10.1 Compilació i execució

10.1.1 Compilació del programa de la mota

El programa es troba a la carpeta **ComprMostres**, que conté:

- La carpeta **src** on hi ha els fitxers amb el codi font. El component principal és **ComprMostresAppC**.
- La carpeta **build/cou24**, on es deixa el resultat de la compilació.
- Un fitxer **Makefile** que permet compilar el programa.

Per a compilar el programa, des de la carpeta **ComprMostres** s'ha d'executar:

make cou24

Per a compilar amb èxit cal tenir correctament instal·lat l'entorn **TinyOS**.

10.1.2 Execució del programa de la mota

El programa s'envia a la mota amb la utilitat **meshprog**. Des de **ComprMostres**, la comanda seria

*meshprog –t/dev/ttyUSB**X** –f./build/cou24/main.srec*

on **X** s'ha de substituir pel número del port USB al que estigui connectada la mota.

Quan s'executa el programa en la mota, aquesta queda a l'espera de rebre un missatge de configuració. Mentrestant no llegeix cap sensor ni envia cap missatge. Només comença a actuar a partir de la recepció d'un missatge de configuració (i encén el led 0).

10.1.3 Compilació de les aplicacions en Java

Les aplicacions en Java es troben a la carpeta **JavaApps**. Aquesta carpeta conté:

- Una carpeta **src** on es guarden els fitxers font. Conté 3 subcarpetes:
  - **apps**, on hi ha els fonts de les 3 aplicacions.
  - **classesAux**, que conté alguna classe auxiliar.
  - **missatges**, que conté les classes Java per als diferents missatges que utilitza el sistema, generats a partir del fitxer **Msgs.h** del codi font de la mota utilitzant la utilitat **mig**.
- Una carpeta **bin**, on es guarden les classes generades per la compilació. Reprodueix l' estructura de **src**.

- Una fixter **compil** i un altre **fes** per a compilar i executar respectivament les aplicacions.

Per a compilar simplement s'ha d'executar **compil** des de la carpeta **JavaApps**.

10.1.4 Execució de les aplicacions en Java

Per a executar les aplicacions sense haver de teclejar les rutes d'accés, només cal executar des de **JavaApps**:

fes app paràmetres

on *app* és el nom de l'aplicació que es vol executar, i *paràmetres* els paràmetres que se li vulguin passar.

Les opcions de parametrització de les diferents aplicacions s'expliquen a l'annex I. Totes són aplicacions de línia de comandes i no tenen cap requeriment especial, a banda de l'entorn Java correctament instal·lat.

10.2 Manual d'usuari

Es descriuen a continuació els paràmetres que poden utilitzar-se en l'execució de cadascun dels programes amb interfície d'usuari.

10.2.1 ConfigSessio

Tots els paràmetres son opcionals. Descripció dels paràmetres:

-i <*Identificador de la sessió*>

Envia a la mota el paràmetre com a identificador de la sessió. Pot ser una cadena alfanumèrica de fins a 16 caràcters.

Valor per defecte: YYYYMMDDhhmmss00 (data i hora més "00").

-m <*Identificador de la mota*>

Valor numèric que identifica la mota a la que s'envia la configuració.

Valor per defecte: 1

-p <*Període de mostreig*>

Interval entre lectures successives del(s) sensor(s), en milisegons.

Valor per defecte: 3000

-s <*Sensors*>

Associa els sensors amb les cues. Cadena de fins a 3 dígits de 0 a 3, on la primera posició fa referència a la cua 1, la segona a la cua 2 i la 3a a la cua 3, i el valor 1 significa sensor 1, el valor 2 sensor 2 i el valor 3 sensor 3, mentre que el valor 0

significa que aquesta cua no s'utilitza. Per exemple, 123 significa que a la cua 1 es guarden les dades del sensor 1, a la 2 les del sensor 2 i a la 3 les del sensor 3; 201 significa que a la cua 1 es guarden les dades del sensor 2, la cua 2 no s'utilitza i a la cua 3 es guarden les dades del sensor 1.

Valor per defecte: 123

-f <*Modes*>

Especifica el mode de funcionament de cada cua: enviar dades o no enviar. Cadena de fins a 3 dígits 0/1 on 0 significa enviar i 1 significa no enviar.

Valor per defecte: 000

-ip <*ip*>

Permet configurar la ip del SerialForwarder.

Valor per defecte: localhost

-port <*num*>

Permet configurar el port del SerialForwarder.

Valor per defecte: 9002

10.2.2 RecullMostres

Nota sobre el funcionament

Quan arranca l'aplicació crea un fitxer on emmagatzemar les dades que puguin arribar, i li posa el nom YYYYMMDDhhmmssXX (data i hora més "XX") més el sufix "_mostres.txt". A partir de la recepció d'un missatge de confirmació de configuració, tanca aquest fitxer i crea un de nou que tindrà com a nom l'identificador de la configuració rebut en el missatge, més el sufix "-mostres.txt". Aquest procés es repeteix per cada nou missatge de confirmació de configuració.

Tots els paràmetres són opcionals. Descripció dels paràmetres:

-g

Visualitza missatges de debug.

Valor per defecte: no activat.

-n

Fa que no es visualitzin els missatges que es reben de mostres.

Valor per defecte: no activat.

-ip <*ip*>

Permet configurar la ip del SerialForwarder.

Valor per defecte: localhost

-port <*num*>

Permet configurar el port del SerialForwarder.

Valor per defecte: 9002

-d <*ruta*>

Permet especificar la ruta del directori on es guardaran els fitxers.

Valor per defecte: ../../mostres

10.2.3 DescompMostres

Invocació:

DescomprMostres <*FitxerEntrada*> [*FitxerSortida*]

Descripció dels paràmetres:

FitxerEntrada

Paràmetre obligatori: nom del fitxer generat per **RecullMostres** on hi ha les dades de les mostres. No s'ha d'incloure el sufix "_mostres.txt".

FitxerSortida

Nom base del fitxer de sortida. L'aplicació crea els fitxers de sortida concatenant a aquest nom "*_M_C*", on M és l'identificador de la mota i C el número de la cua.

Valor per defecte: el mateix *FitxerEntrada*.

10.2.4 Exemples d'ús

a) Es volen recollir dades dels 3 sensors de la mota 1 durant un cert temps de temps amb un període de mostreig de 10 segons. Les dades es volen guardar en un fitxer anomenat "captura123", a la carpeta ..//..//Mota/proves.

1. Executar
   fes RecullMostres –d ../../Mota/proves
2. Executar
   fes ConfigSessio –i captura123 –m 1 –p 10000 –s 123 –f 000
3. Quan es vulgui finalitzar la captura executar
   fes ConfigSessio –m1 –s000
4. A la carpeta *../../Mota/proves* hi haurà un fitxer anomenat *captura123_mostres.txt*. Per a descomprimir-lo executar

fes DescomprMostres ../../Mota/proves/captura123

5. Com a resultat, a la carpeta *../../Mota/proves* hi haurà els fitxers *captura123_1_1*, *captura123_1_2* i *captura123_1_3*, que contindran respectivament les dades del sensor 1, 2 i 3.

b) Es volen obtenir dades del sensor 2 sense comprimir i comprimides per tal de comparar-les:

1. Executar *fes RecullMostres –d ../../Mota/proves*
2. Executar
   fes ConfigSessio –i nocompr –m 1 –p 10000 –s 22 –f 01
3. Quan es vulgui finalitzar la captura (quan el nombre de missatges que han arribat de la cua 1 és més gran que la capacitat de la cua i, per tant, s'ha engegat la compressió), executar:
   fes ConfigSessio –i compr –m 1 –p 10000 –s 02 –f 10
4. A la carpeta *../../Mota/proves* hi haurà dos fitxers: anomenat *nocompr_mostres.txt* i *compr_mostres.txt*. Per a descomprimir-lo executar
   fes DescomprMostres ../../Mota/proves/nocompr
   i
   fes DescomprMostres ../../Mota/proves/compr
5. Com a resultat, a la carpeta *../../Mota/proves* hi haurà els fitxers *nocompr_1_1* , que contindrà les mostres sense comprimir, i *compr_1_2*, que contindrà les mostres que havien estat comprimides.