ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Asistente para el entrenamiento aeróbico basado en técnicas de aprendizaje máquina

AUTOR: José Luis Sieiro Lomba
DIRECTOR: Javier Vales Alonso
Junio / 2015

Autor	José Luís Sieiro Lomba
E-mail del Autor	jlsielom@gmail.com
Director(es)	Javier Vales Alonso
E-mail del Director	javier.vales@upct.es
Título del PFC	Asistente para el entrenamiento aeróbico basado en técnicas de aprendizaje máquina
Descriptores	Asistente, monitorización biométrica, entrenamiento aeróbico, aprendizaje máquina

Resúmen

Los deportistas siguen rutinas específicas de entrenamiento, que se deben adaptar para conseguir maximizar el rendimiento en función de una planificación cuidadosa.

Además, la especialización característica de los deportes colectivos hace que se cuenten con distintos perfiles dentro del equipo, en los que es más o menos deseable explotar determinadas características o grupos musculares en función de la posición en la que el deportista desarrolle su función.

El proyecto SAETA tenía como objetivo desarrollar un sistema integrado de control de entrenamiento deportivo basado en técnicas de aprendizaje supervisado. En dicho sistema se ha implementado, por una parte, un complejo motor de decisión cuya misión es proporcionar ordenes personales de entrenamiento a los atletas y, por otra parte, se ha integrado dicho sistema de toma de decisiones en una estructura de sensorización que permite la monitorización de los atletas (datos de biometría y movimiento) y del entorno (posición, humedad, temperatura, etc.). El sistema completo dio lugar a un entrenador virtual cuya misión es asistir a los jugadores y a los entrenadores humanos.

El presente proyecto se inició para formar parte del proyecto SAETA y desarrollar un sistema de decisión para proporcionar asistencia personalizada a deportistas que permitiese mejorar su adiestramiento con énfasis en la monitorización biométrica y en la toma de decisiones de las planificaciones de manera automática que permitiese mejorar el adiestramiento aeróbico en particular.

El proyecto se estructura en dos partes, por un lado, la monitorización en tiempo real de las pulsaciones y de la saturación sanguínea del deportista, y por otro, el control del entrenamiento aeróbico basado en técnicas de aprendizaje máquina.

Titulación	Ingeniero de Telecomunicación
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Junio - 2015

1 Introducción

Los deportistas siguen rutinas específicas de entrenamiento, que se deben adaptar para conseguir maximizar el rendimiento en función de una planificación cuidadosa.

Además, la especialización característica de los deportes colectivos hace que se cuenten con distintos perfiles dentro del equipo, en los que es más o menos deseable explotar determinadas características o grupos musculares en función de la posición en la que el deportista desarrolle su función.

En el pasado, las rutinas de entrenamiento son personalizadas desde el punto de vista temporal, y en función de las características propias del deportista, tanto intrínsecas como las referentes a su posición dentro del equipo. Los preparadores físicos se encargan de este control, que bajo su conocimiento y experiencia, confeccionan una planificación de rutinas personalizadas.

El proyecto SAETA tenía como objetivo desarrollar un sistema integrado de control de entrenamiento deportivo basado en técnicas de aprendizaje supervisado. En dicho sistema se ha implementado, por una parte, un complejo motor de decisión cuya misión es proporcionar ordenes personales de entrenamiento a los atletas y, por otra parte, se ha integrado dicho sistema de toma de decisiones en una estructura de sensorización que permite la monitorización de los atletas (datos de biometría y movimiento) y del entorno (posición, humedad, temperatura, etc.). El sistema completo dio lugar a un entrenador virtual cuya misión es asistir a los jugadores y a los entrenadores humanos.

SAETA se inició como continuación de la acción de investigación “Sistemas de inteligencia ambiental para asistencia a deportistas con perfiles específicos - Caracterización del entorno y aplicaciones (AISAS)”. AISAS tuvo como objetivo el desarrollo de diversas tecnologías y sistemas de inteligencia ambiental para proporcionar asistencia personalizada a deportistas, con énfasis en tres pilares fundamentales: la adaptación automática del entorno al deportista (inteligencia ambiental), la localización y monitorización del deportista y su entorno, y la aplicabilidad general del sistema a distintos deportes, tanto en entornos interiores como en exteriores. Sobre dichos pilares se asienta también el proyecto SAETA.

Así mismo, el presente proyecto se inició para formar parte de SAETA, y más concretamente, en la monitorización biométrica del atleta y en la toma de decisiones de las planificaciones de manera automática que permitiese mejorar el adiestramiento aeróbico.

El proyecto se estructura en dos partes, por un lado, la monitorización biométrica en tiempo real de las pulsaciones y de la saturación sanguínea del deportista, y por otro, el control del entrenamiento aeróbico basado en técnicas de aprendizaje máquina.

2 Arquitectura y monitorización biométrica

La monitorización biométrica se integra dentro de la arquitectura en forma de sensor, cuyos datos se retransmitirán a través de los nodos de la infraestructura.

En este apartado se presenta la arquitectura del entrenador virtual del proyecto SAETA del cual forma parte el presente proyecto. La arquitectura del sistema SAETA parte de la desarrollada en el proyecto AISAS. El objetivo de AISAS fue diseñar una arquitectura general para la asistencia a deportistas y monitorización ambiental, a fin de proporcionar información en tiempo real. La infraestructura original estaba orientada principalmente hacia deportes o actividades de campo abierto. En estos escenarios el desempeño de atleta, no sólo depende de las condiciones físicas, sino también en las condiciones del terreno (pendiente, temperatura, viento, etc.). Estas características son similares a las requeridas en la arquitectura de SAETA.

Como se ha comentado, en el proyecto SAETA, el objetivo principal ha sido el desarrollo de un entrenador virtual integral, que sustituya las labores de un entrenador humano, mediante un sistema completo de análisis de los entrenamientos, basado en técnicas de aprendizaje máquina. Es decir, la creación de un entrenador virtual (un software) que monitoriza a los atletas durante su entrenamiento para aprender cual es la mejor forma de entrenarlos. Este entrenador virtual envía a los atletas las pautas de entrenamiento adecuadas en cada momento a partir del conocimiento adquirido y las condiciones actuales del deportista y del medio.

2.1 Introducción

A continuación se muestra un esquema de la arquitectura desarrollada en SAETA.

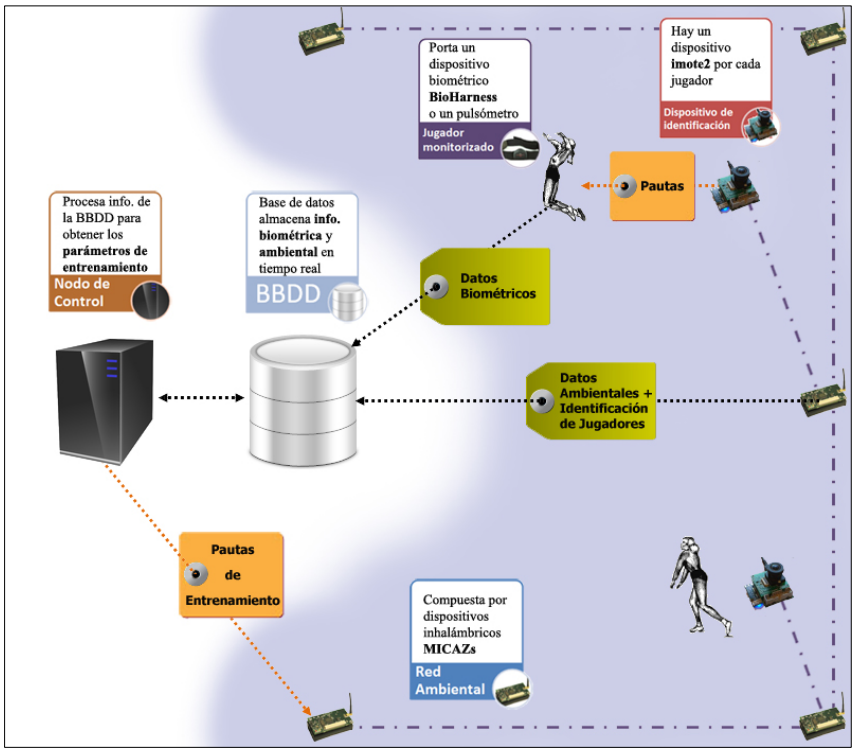


Figura 2.1 Arquitectura del sistema SAETA

La arquitectura del sistema SAETA está diseñada para adquirir dos tipos de información.

- Un **conjunto de datos estático**, con la información que no cambia durante el tiempo de entrenamiento, que a su vez se dividen en datos del entorno (por ejemplo, la posición de los sensores, la rutina de entrenamiento programada, pendiente del terreno, configuración de la ruta y otros), y en datos del usuario (edad, tipo de entrenamiento, objetivos del entrenamiento y otros).

- Un **conjunto de datos en tiempo real**, con información actualizada del entorno (variables ambientales, como la temperatura) y los datos del usuario (biometría, como las pulsaciones).

Para la toma de datos en tiempo real se ha desplegado una red de sensores. Esta red recoge datos ambientales y además sirve de red de comunicación entre el usuario y el elemento que procesa los datos. Además el usuario dispone de sensores que toman datos biométricos, y permiten la comunicación y localización del corredor. Por último, un nodo de control es el que procesa los datos y ejecuta los módulos de control del entrenador virtual.

2.2 Redes de sensores

Las redes de sensores o también conocidas como WSN[1] están formadas por un grupo de sensores con ciertas capacidades sensitivas y de comunicación los cuales permiten formar redes inalámbricas Ad-Hoc[2] sin infraestructura física preestablecida ni administración central.

Las redes de sensores es un concepto relativamente nuevo en adquisición y tratamiento de datos con múltiples aplicaciones en distintos campos tales como entornos industriales, domótica, entornos militares, detección ambiental.

Esta clase de redes se caracterizan por su facilidad de despliegue y por ser auto configurables, pudiendo convertirse en todo momento en emisor, receptor, ofrecer servicios de encaminamiento entre nodos sin visión directa, así como registrar datos referentes a los sensores locales de cada nodo. Otra de sus características es su gestión eficiente de la energía, que con ello conseguimos unas alta tasa de autonomía que las hacen plenamente operativas.

Cada nodo, como ente individual de una red de sensores, no deja de ser una pequeña computadora, con un pequeño procesador, una memoria de programa y una memoria para almacenar variables, pero al que también agregamos unos pequeños periféricos I/O (entrada/salida) tales como un modulo radio y un pequeño conversor A/D (Analógico/Digital) que sirve para adquisición de los datos de los sensores locales.

Una red de sensores contiene cientos o miles de estos nodos sensores, los cuales tienen la habilidad de comunicarse con cualquier otro nodo o directamente a una estación

[1] WSN: Wireless Sensor Network
[2] Ad-Hoc: Se refiere a una solución elaborada específicamente para un problema o fin preciso. En sentido amplio, ad hoc puede traducirse como «específico» o «específicamente».

base, conocida como sumidero o *sink*. Un mayor número de sensores permite sensar un área geográfica con mayor precisión. Básicamente, cada nodo sensor está compuesto por una unidad de sensado, procesamiento, transmisión, posicionamiento y unidad de potencia. Los nodos sensores normalmente son desplegados en un campo y se auto configuran para producir información de alta calidad acerca del ambiente físico. Cada nodo sensor tiene la capacidad de recolectar y enrutar los datos a otros sensores o a al sumidero. Un sumidero puede ser un nodo externo o un nodo móvil capaz de conectar la red de sensores a una infraestructura de comunicación existente o a Internet donde un usuario puede acceder a los datos recolectados.

La topología de la red de sensores aparece de forma esquemática en la siguiente figura.

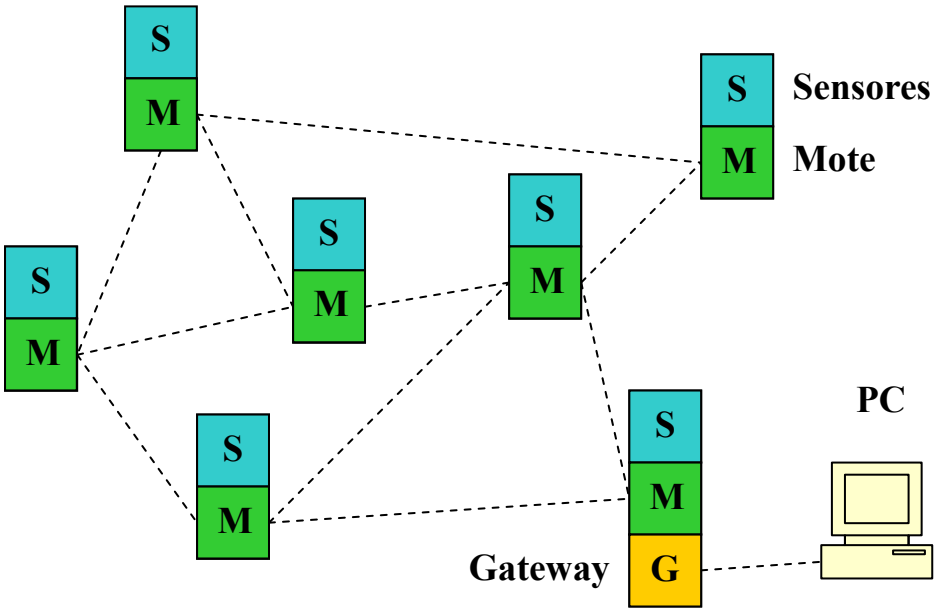


Figura 2.2: Topología general de una red de sensores.

Compuestas por:

- **Sensores**: Toman del medio la información y la convierten en señales eléctricas.
- **Nodos**: Toman los datos del sensor a través de sus puertas de datos, y envían la información a la estación base.
- **Gateway**: Estación base, que puede permitir la interconexión entre la red de sensores y un computador o directamente TCP/IP.
- **Red Inalámbrica**: Típicamente basada en el estándar 802.15.4 *ZigBee* [1].

Se pueden clasificar las redes de sensores según si son manejadas por tiempo o por evento. Las primeras son aquellas que requieren monitoreo periódico de datos. En las segundas, los nodos sensores reaccionan inmediatamente a cambios drásticos en el valor de un atributo debido a la ocurrencia de cierto evento [2].

Existen varias restricciones en las redes de sensores que deben ser consideradas en el diseño de cualquier protocolo para estas redes. Algunas de estas limitaciones son:

- ***Suministro de energía limitado:*** Requieren de protocolos eficientes en el consumo de energía.

- ***Capacidad limitada de cómputo:*** No pueden correr protocolos sofisticados de red.

- ***Comunicación:*** El ancho de banda de enlaces inalámbricos que conectan a los sensores es limitado.

No existe una pila de protocolos estándar para WSN. Se sugiere que dicha pila de protocolos consista de tres planos [3]: el plano de comunicación, el plano de coordinación y el plano de administración. El plano de comunicación habilita el intercambio de información entre los nodos de la red. Los datos recibidos por un nodo en el plano de comunicación son remitidos al plano de coordinación el cual decide cómo los nodos actúan. Sin embargo el plano de coordinación permite a los nodos ser modelados como una entidad social en términos de la coordinación y las técnicas de negociación que éste posea. El plano de administración es responsable de monitorear y controlar un nodo sensor para que éste opere apropiadamente. Este también provee la información necesaria por la capa de coordinación para la toma de decisiones.

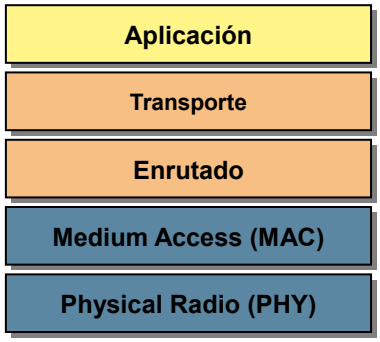
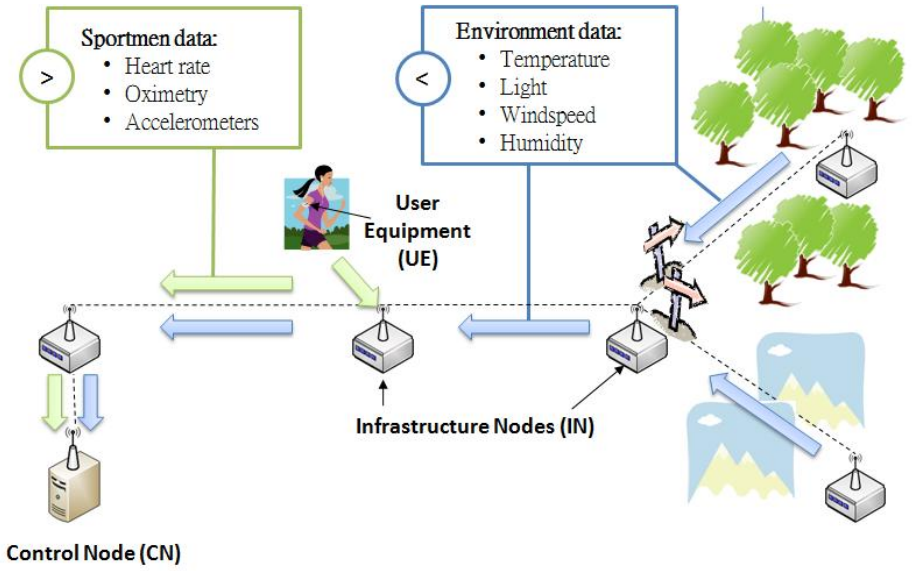


Figura 2.3: Plano de comunicaciones.

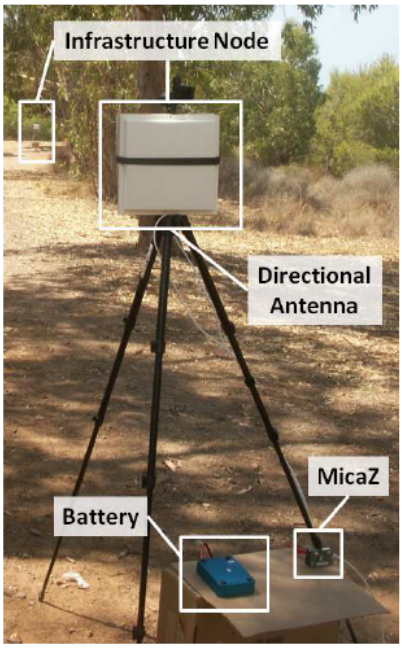
En diversos trabajos de investigación, se recomienda un modelo de inter-capas (*cross layer)*, [2] y [4] donde las capas sean integradas unas con otras. Mediante la propuesta de inter-capas, cada protocolo comparte sus datos con otros protocolos, evitando con ello ineficiencias. Para proveer una estructura de paquete unificado que incorpore las funcionalidades de cada protocolo, las capas de enrutamiento, MAC y física deben ser investigadas juntas. La idea básica de inter-capas es que cada capa de la pila de protocolos no sólo responda a las variaciones locales, sino también a la información de otras capas.

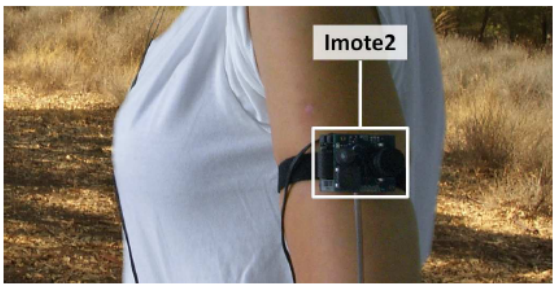
2.3 Arquitectura

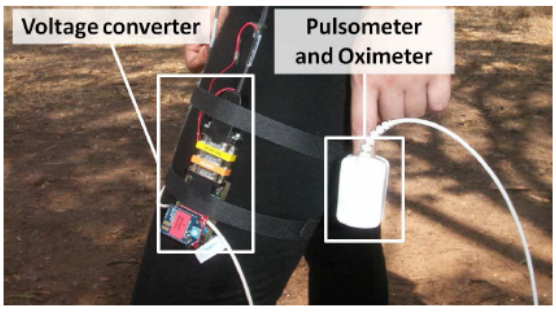
La arquitectura de SAETA fue diseñada para distintos tipos de entrenamientos y por tanto son posibles distintos sensores biométricos. El presente proyecto se centra en los entrenamientos aeróbicos y a continuación se muestra el esquema de dicha arquitectura.



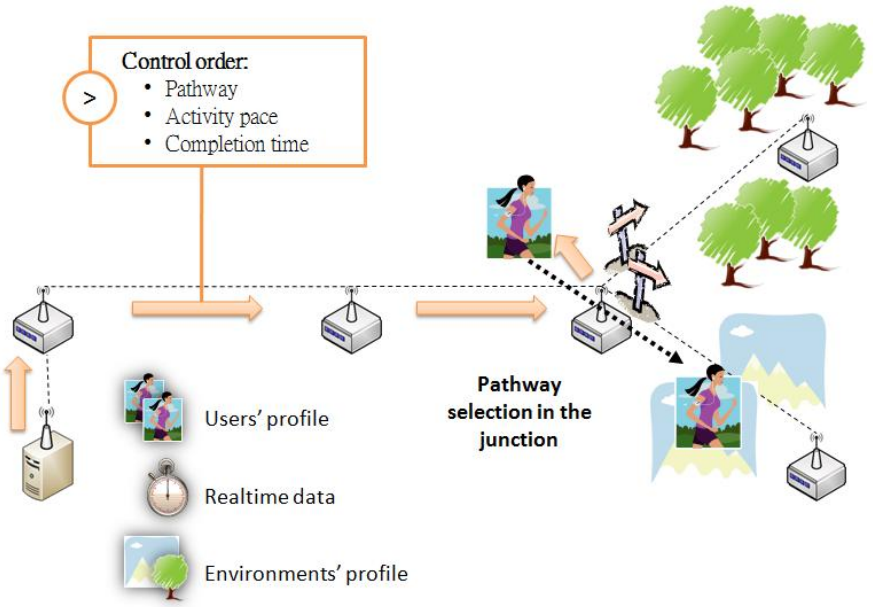
La WSN en cuestión ha sido implementada mediante sensores del tipo *MICAz*. El equipamiento de usuario UE está formado por un *IMOTE2*, que permite reproducir comandos de voz, y por un *iPod* (modelo 3211 de *Nonin Medical Company*), que permite la toma de datos biométricos.







Con los datos recogidos se caracterizó el nodo central, el Nodo Control que es el encargado de gestionar la comunicación entre sistema de monitorización y el servidor, siendo éste el sumidero de todos los datos recolectados en el sistema y punto donde se toman las decisiones (entrenador virtual) que regirán el entrenamiento de los usuarios.



Control orders are generated based on real-time and static data of sportmen and environment

Para la interactuación del usuario final con la aplicación SAETA, se realizó en Matlab una interfaz de usuario que facilite su utilización. Cuando la aplicación es lanzada muestra la ventana principal mostrada en la figura anterior, donde encontramos para la modalidad de entrenamiento que se desea realizar. Para cada uno de los entrenamientos se abre una nueva ventana que permite al entrenador seleccionar el jugador y los parámetros específicos deseados para su entrenamiento.

Figura 2.4 Pantalla principal de la aplicación SAETA

2.3.1 Hardware

A continuación se describen las características técnicas de los elementos hardware que pueden componer dicha red, comercializados por *Crossbow Technologies Inc*. [5] [6] o *Nonin Medical Company* [7]

2.3.1.1 Mote MPR2400

Este mote, también llamado MICAz, se muestra en la siguiente figura.

Figura 2.5 Dispositivo MICAz

Los *motes* MICAz se utilizan en redes de sensores de bajo consumo, están basados en el sistema operativo de código abierto TinyOS, y sus principales características [8] son las siguientes:

- Micro controlador Atmega 128L.
- RAM de 4KBytes.
- Memoria flash serie Atmel AT45DB041B de 512KB.
- 3 leds.
- Alimentación 2 pilas AA (rango de trabajo 2.7-3.3VDC).
- Conector MMCX para una antena.
- Usa el Chipcon CC2420 [9] que es un modulo radio en la banda de 2.4GHz a 2.48 GHz compatible con la IEEE/ZigBee 802.15.4. Espectro de Radio extendido resistente a interferencias electromagnéticas y seguridad hardware (AES-128).

- Tasa binaria máxima de 250 Kbps de velocidad de transferencia vía radio.

- Conector, modelo Hirose Serie DF9, de entrada/salida de 51 pines (10bits ADC, Entradas/Salidas Digitales, I2C, interfaces SPI y UART, etc.).

- Posibilidad de añadir placas de sensores, placas de adquisición de datos, Gateway y software (sensores de Luz, Temperatura, Presión Barométrica, Aceleración, Acústicos, Magnéticos y otros tipos de sensores).

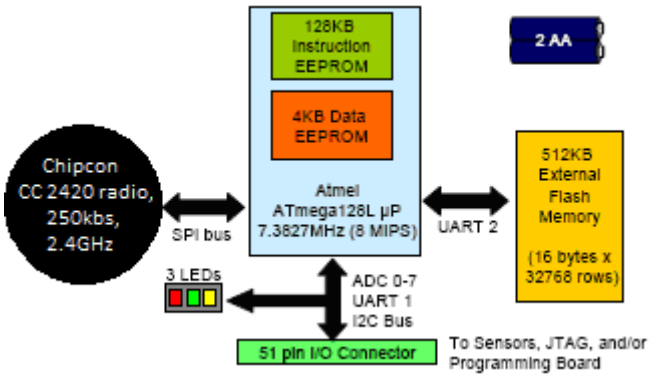


Figura 2.6: Diagrama de bloques del MICAz

Con este dispositivo se despliega la red ambiental, elemento principal para la monitorización ambiental, que a su vez sirve de infraestructura de comunicaciones. Tanto los nodos de infraestructura como el nodo de control incluyen MICAz como interfaz de comunicaciones.

2.3.1.2 Placa MTS400 (Sensorización ambiental)

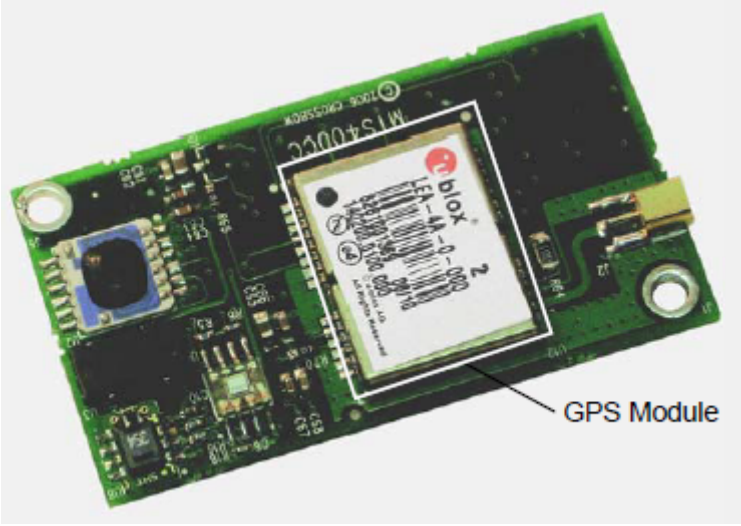


Figura 2.7 Placa MTS400 igual a la MTS420 pero sin módulo GPS

Estas placas de sensores han sido las empleadas en los nodos de la red ambiental acoplada a los dispositivos MICAz y proporcionan información sobre la humedad y temperatura.

2.3.1.3 Placa MIB520 (Gateway)

La placa de desarrollo MIB520 [10] actúa como interfaz *Gateway* entre el PC y los motes (interfaz con el nodo de control), al mismo tiempo que permite la programación de los dispositivos acoplados.

La placa MIB520 tiene un procesador (ISP) Atmega16L mediante el cual se programan los motes. El código se descarga al ISP a través del puerto USB, y es el ISP el que programa el código en el mote. Para programar los motes se ha de tener instalado en el PC el sistema operativo TinyOS[3], descrito más adelante. La placa de desarrollo tiene conectores para los motes MICAz o MICA2. Su alimentación y la del mote conectado se realizan mediante el USB, lo que elimina la necesidad de una toma de corriente y pilas para el mote.

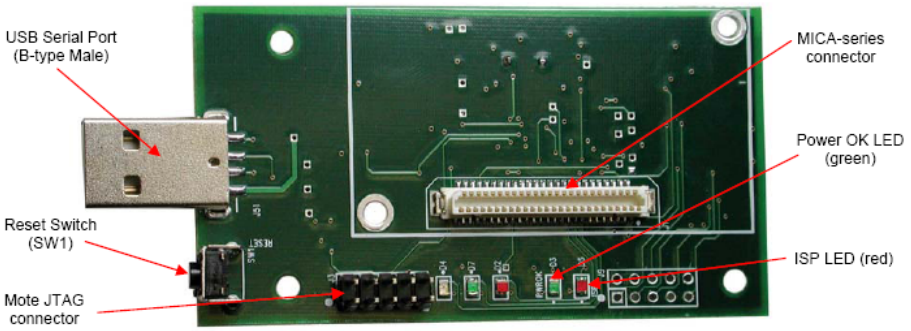


Figura 2.8: Vista superior de la placa MIB520[4]

El nodo control del sistema se compone de dos elementos, un dispositivo MICAz y una placa de conexión MIB520 que, a través del puerto USB permite enviar y recibir comandos.

Figura 2.9 Dispositivo MICAz acoplado a una placa de conexión MIB520

[3] TinyOS: Tiny microthreading Operative System.
[4] http://www.xbow.com/Products/productdetails.aspx?sid=227

2.3.1.4 Mote IPR2400 IMOTE2 y placa IMB400

La plataforma IPR2400 IMOTE2 aporta un nuevo nivel de rendimiento y capacidad para el procesador y la plataforma de radio en redes inalámbricas de sensores.

Figura 2.10 Dispositivo IMOTE2

El IMB400 añade capacidades multimedia a la plataforma IMOTE2 con la integración de una cámara, audio y la funcionalidad de detección de movimiento. En cuanto a programación y capacidades de radiocomunicación los IMOTE2 son idénticos a los MICAz.

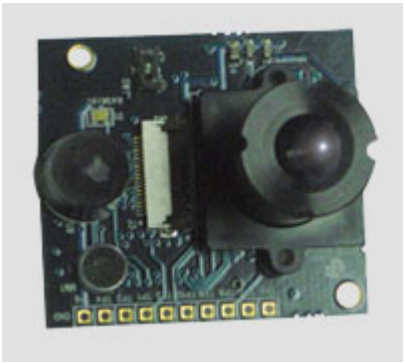


Figura 2.11 Placa de sensorización IMB400 para IMOTE2

Los dispositivos IMOTE2 desplegados en la red llevan acoplados la placa de sensorización IMB400, estos nodos, son portados por cada usuario o ubicados en su proximidad, de forma que cada nodo esta asociado a un único usuario. Estos nodos están integrados en el sistema de forma que a efectos de comunicación es irrelevante si se trata de un dispositivo MICAz o IMOTE2, ya que el intercambio de mensajes procede de forma transparente a la plataforma hardware.

Los dispositivos IMOTE2, no solo mantendrán identificado a los usuarios si no que, son los encargados de reproducir en audio las instrucciones recibidas desde el servidor para el usuario asociado.

2.3.1.5 Sensor iPod 3211[5] (Sensorización biométrica)

Uno de los objetivos del presente proyecto era la sensorización biométrica en tiempo real.

El sensor elegido para la toma de las medidas biométricas de saturación sanguínea (%SpO2) y las pulsaciones (HR) fue un sensor iPod (*Integrated Pulse Oximetry Device*) de NONIN [11], concretamente el modelo 3211 con un formato de datos serie tipo 1 (número de referencia 3830-101). Cuyas especificaciones pueden encontrarse en la web[6] del fabricante o en el anexo del presente documento.

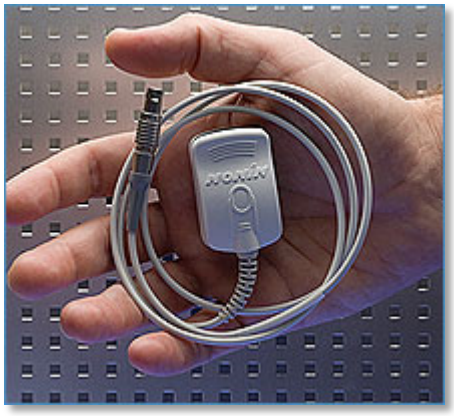


Figura 2.12 Sensor iPod

Los motivos tras su elección son, su bajo peso, tamaño, estabilidad en sus medidas y, sobre todo, por su fácil integración con la arquitectura, debido a su interfaz serie RS-232 con una configuración de 9600/8N1.

Estos sensores también forman parte del equipamiento de usuario y se ajustan al dedo índice. En el apartado de la monitorización biométrica volveremos a hablar sobre este sensor y, sobre todo, de su integración en la arquitectura.

[5] http://www.nonin.com/products/ipod-digital-oximeter-and-sensor-all-in-one/
[6] http://www.nonin.com/documents/ipod%20Specifications.pdf

2.3.2 Software

El hardware comercializado por *Crossbow Technologies Inc*. [5] [6] funciona bajo el sistema operativo TinyOS [12], desarrollado en la Universidad de Berkeley. En este apartado se describen las características de este sistema operativo y de su característico lenguaje de programación, puesto que se ha tenido que desarrollar un driver para la integración del sensor biométrico en la red de sensores.

2.3.2.1 TinyOS

TinyOS es un sistema operativo basado en componentes para sistemas empotrados inalámbricos y de libre distribución. Basado en responder a las características y necesidades de las redes de sensores, tales como reducido tamaño de memoria, bajo consumo de energía, operaciones de concurrencia intensiva, diversidad en diseños y usos, y finalmente operaciones robustas para facilitar el desarrollo confiable de aplicaciones. Además se encuentra optimizado en términos de uso de memoria y eficiencia de energía.

En este tipo de arquitecturas, basadas en componentes, las aplicaciones se crean enlazando componentes, permitiendo de esta manera la reutilización de código a través de sus interfaces y la abstracción de las características del hardware con el que se trabaja.

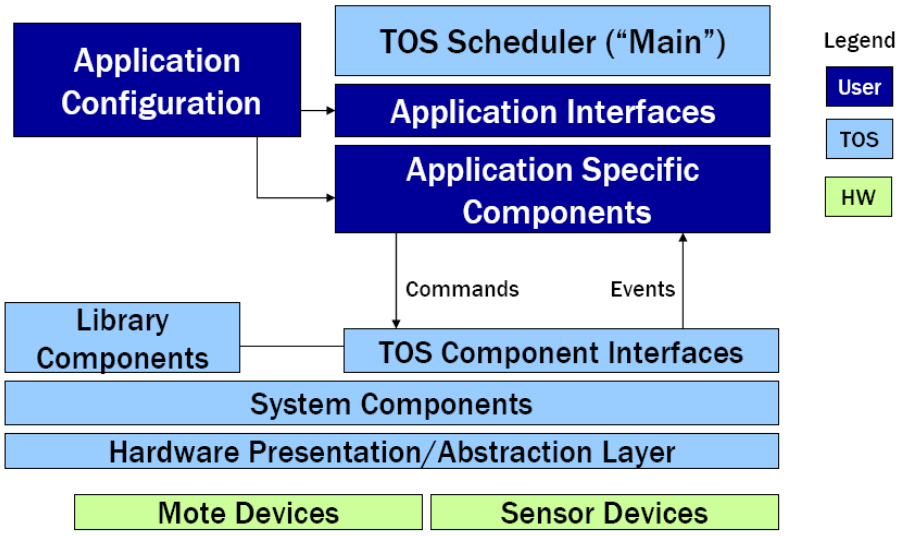


Figura 2.13: Arquitectura de TinyOS.

Para entender TinyOS, debemos de tener presentes 3 clases de abstracción que son la esencia para su entendimiento y posterior programación:

- Los comandos son las llamadas hacia abajo. Al llamar a un comando, éste dentro de su componente llama a otros componentes de capas inferiores.

- Los eventos conllevan el efecto inverso, una llamada hacia arriba. Un componente de bajo nivel avisa a uno de alto nivel de que ha sucedido algo. Si sucede un evento, éste tiene mayor prioridad que cualquier tarea y pasaría a ejecutarse. En caso de haber una interrupción dentro de una interrupción lo más probable es que no sea ejecutada esta hasta que acabara la anterior, siendo esto algo crítico.

- Las tareas son porciones de código que se ejecutan de forma asíncrona siempre y cuando la CPU no tenga que ejecutar ningún evento. Estas se ejecutan por orden de llamada (FIFO) y tal y como antes hemos comentado en caso de que la cola este vacía el procesador entrará en standby hasta que un nuevo evento haga despertar al procesador.

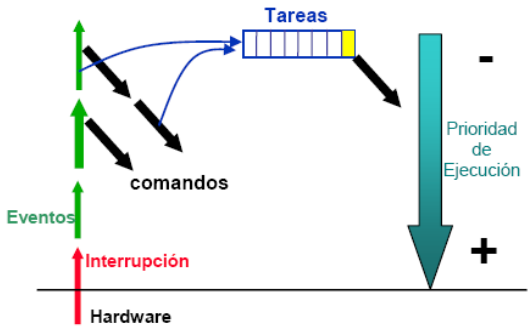


Figura 2.14: Orden y prioridades de ejecución en TinyOS.

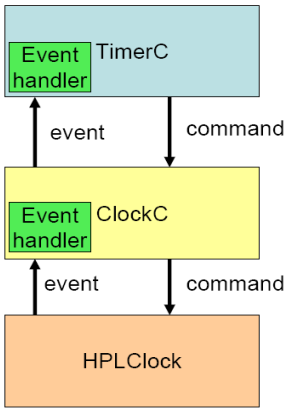


Figura 2.15: Diagrama ilustrativo de la relación entre los componentes y los tipos de abstracción.

Por tanto, el diseño del Kernel de TinyOS está basado en una estructura de dos niveles de planificación.

Eventos: Pensados para realizar un proceso pequeño (por ejemplo cuando el contador del timer se interrumpe, o atender las interrupciones de un detector de tono). Además pueden interrumpir las tareas que se están ejecutando.

Tareas: Las tareas son pensadas para hacer una cantidad mayor de procesamiento y no son críticas en tiempo (por ejemplo calcular el promedio en un arreglo). Las tareas se ejecutan en su totalidad, pero la solicitud de iniciar una tarea, y el término de ella son funciones separadas. Esta característica es propia de la programación orientada a componentes, la cual se presentará con más detalle en la sección siguiente.

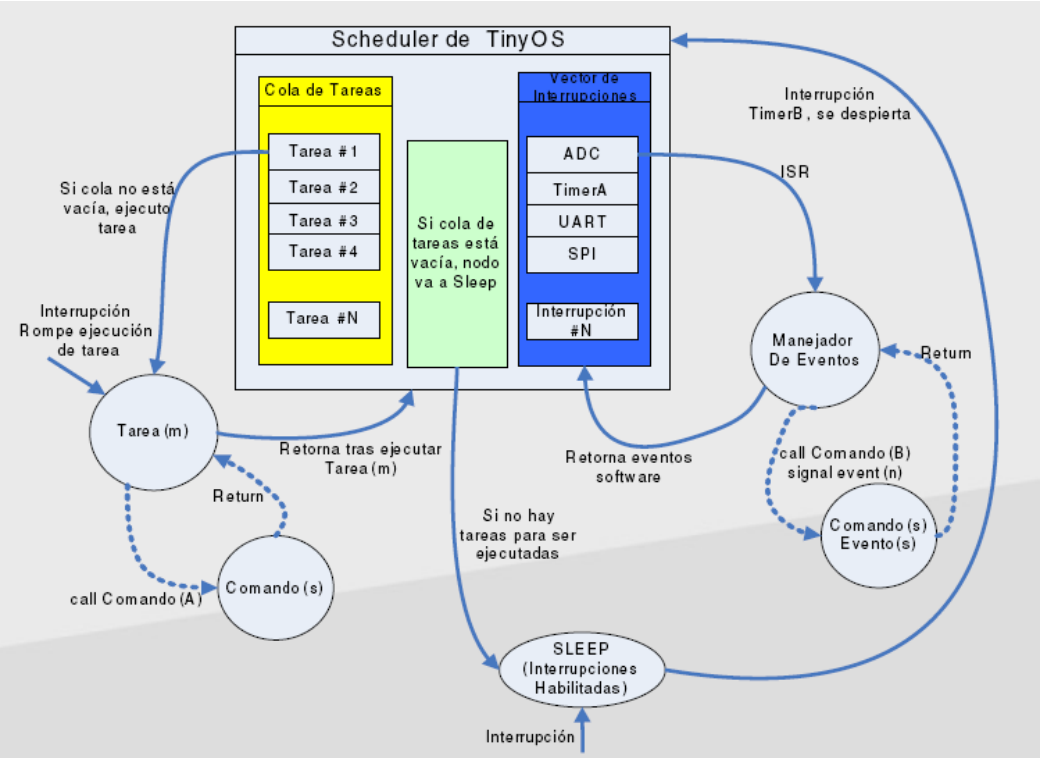


Figura 2.16: Kernel de TinyOS.

Con este diseño permitimos que los eventos (que son rápidamente ejecutables), puedan ser realizados inmediatamente, pudiendo interrumpir a las tareas (que tienen mayor complejidad en comparación a los eventos).

Su modo de ejecución esta basado en eventos, y la concurrencia la realiza a través de tareas.

El enfoque basado en eventos es la solución ideal para alcanzar un alto rendimiento en aplicaciones de concurrencia intensiva. Adicionalmente, este enfoque usa las capacidades de la CPU de manera eficiente y de esta forma gasta el mínimo de energía.

Cuando programamos y compilamos una aplicación, esta va siempre unida de forma implícita al sistema operativo, de forma que no existe el sistema operativo (previamente precargado) y el programa compilado, sino el conjunto que es subido al sistema empotrado cada vez que necesitamos actualizar el programa. Aún con todo, su tamaño en Bytes, sigue siendo minúsculo.

Resumen de características:

- Núcleo reducido, de pocos cientos de bytes.
- Arquitectura orientada a componentes.
- Abstracción de capas, a nivel de interfaces (bidireccionales, con comandos y eventos, que realizan servicios).
- Reutilización de código.

- Adaptado para recursos limitados como: energía, procesamiento, almacenamiento y ancho de banda.
- Concurrencia de tareas y basada en eventos (*Event Driven*).
- Lenguaje de programación NesC (similar a C).
- Los comandos los implementa el proveedor y los eventos el usuario de dicho componente.
- Un módulo implementa una interfaz.
- Los componentes proveen y usan interfaces (representado en el código por las etiquetas "*provide*" y "*use*").
- Una configuración enlaza las interfaces internas y externas (*wire*).
- Una aplicación consiste en una configuración de alto nivel y todos los módulos asociados.

2.3.2.2 NesC

TinyOS se encuentra programado en NesC[7] [13], un lenguaje diseñado para reflejar las ideas propias del enfoque de componentes, incorporando además un modelo de programación que soporta concurrencia, manejo de comunicaciones y fácil interacción con el medio (manejo de hardware).

Es un lenguaje estático, por lo cual no existen asignaciones de memoria de manera dinámica y los gráficos de llamados quedan totalmente definidos en el tiempo de compilación. Esto facilita enormemente el análisis de programas, y por ende la detección de errores.

NesC y TinyOS se encuentran profundamente relacionados, cualquier avance o desarrollo de uno se verá reflejado en el otro. Es por esto que se hace necesario en su totalidad el buen manejo de este lenguaje.

La biblioteca de componentes de TinyOS incluye protocolos de red, servicios distribuidos, manejadores de sensores, y herramientas de adquisición de datos, que pueden ser ocupadas directamente, o manipuladas para responder a necesidades más específicas.

En NesC cada aplicación está construida a base de componentes enlazados para formar un ejecutable. Los componentes pueden proporcionar o utilizar interfaces de otros componentes. La forma de acceder al uso en otro contexto de estos componentes es mediante estas interfaces que son bidireccionales es decir mediante eventos y mediante comandos. Con estas interfaces podemos acceder a la funcionalidad de los componentes.

En NesC existen dos tipos de componentes: los módulos y las configuraciones. Los componentes proporcionan y utilizan interfaces, que son los puntos de acceso hacia y

[7] NesC: Network Embedded Systems C

desde los componentes. En las interfaces se declaran comandos que ha de implementar el que proporciona la interfaz y eventos que han de ser implementados por el usuario de dicha interfaz. Los módulos contienen el código del programa y es donde se implementan las interfaces, mientras que las configuraciones son las que se usan para enlazar las interfaces de los componentes entre sí, proceso que se conoce también como *wiring*. Tanto módulos como configuraciones utilizan la misma extensión **.nc*, la diferencia entre ambos está en la sintaxis que utilizan y es por ello que siguen un código para diferenciar unos de otros. Por ejemplo cuando trabajamos con un programa "*ejemplo*", *ejemploM.nc* seria el módulo y *ejemplo.nc* seria el fichero configuración, en ocasiones a este último se lo nombre de la forma *ejemploC.nc*.

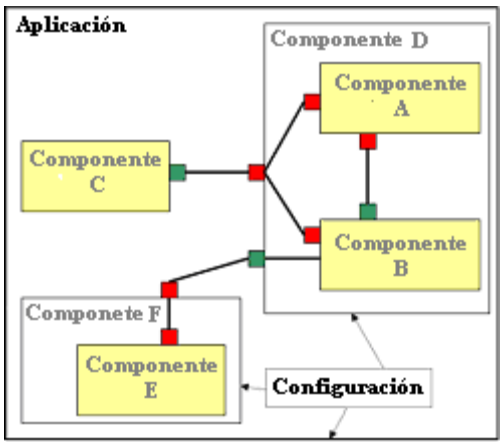


Figura 2.17: Ejemplo de aplicación de TinyOS.

La idea que hay detrás de este tipo de programación, es que el propio sistema operativo en conjunción con las casas que venden los dispositivos de sensores proporcionan de forma intrínseca ciertos componentes ya implementados que ofrecen al programador un montón de funciones y utilidades para que el programador de este tipo de dispositivos pueda utilizar dichos componentes y centrarse solo en programar la funcionalidad que desea en el dispositivo sin necesidad de tener que preocuparse por todos estos aspectos.

Por otro lado, NesC combina ciertos aspectos de la orientación a objetos, en el sentido de que se basa en una programación orientada a interfaces y de la orientación a eventos, en el sentido de que posee un manejo de eventos propio de este tipo de lenguajes como Visual Basic.

Todos los componentes que ofrece de forma intrínseca el sistema operativo se denominan componentes primitivos y los componentes proporcionados por terceros, contribuciones, librerías, aplicaciones se denominan componentes complejos.

El sentido que se quería resaltar con lo dicho anteriormente de que es un lenguaje orientado a objetos, es que todos los componentes ya sea primitivos o complejos proporcionan unas interfaces, y si un programador quiere utilizar un componente, la forma de hacerlo es usar dichas interfaces, pero recordemos que son interfaces en el sentido OO, por lo que en un momento dado se podría cambiar un componente por otro siempre y cuando este otro proporcionase la misma interfaz y este cambio no afectaría al código de la implementación de la aplicación.

En cuanto a la orientación a eventos, decir que básicamente la forma de programar

la aplicación no es del todo secuencial, sino que atiende a una programación basada en eventos, de manera que se programan las acciones que se desea realizar cuando se produzca un evento y este fragmento se ejecutará exactamente cada vez que se lleve a cabo dicho evento.

Básicamente NesC ofrece:

- Separación entre la construcción y la composición. Hay dos tipos de componentes en NesC: módulos y configuraciones. Los módulos proveen el código de la aplicación, implementando una o más interfaces. Estas interfaces son los únicos puntos de acceso a la componente. Las configuraciones son usadas para unir las componentes entre sí, conectando las interfaces que algunas componentes proveen con las interfaces que otras usan.

- Interfaces bidireccionales: tal como ya se explicaba anteriormente las interfaces son los accesos a las componentes, conteniendo comandos y eventos, los cuales son los que implementan las funciones. El proveedor de una interfaz implementa los comandos, mientras que el que las utiliza implementa eventos.

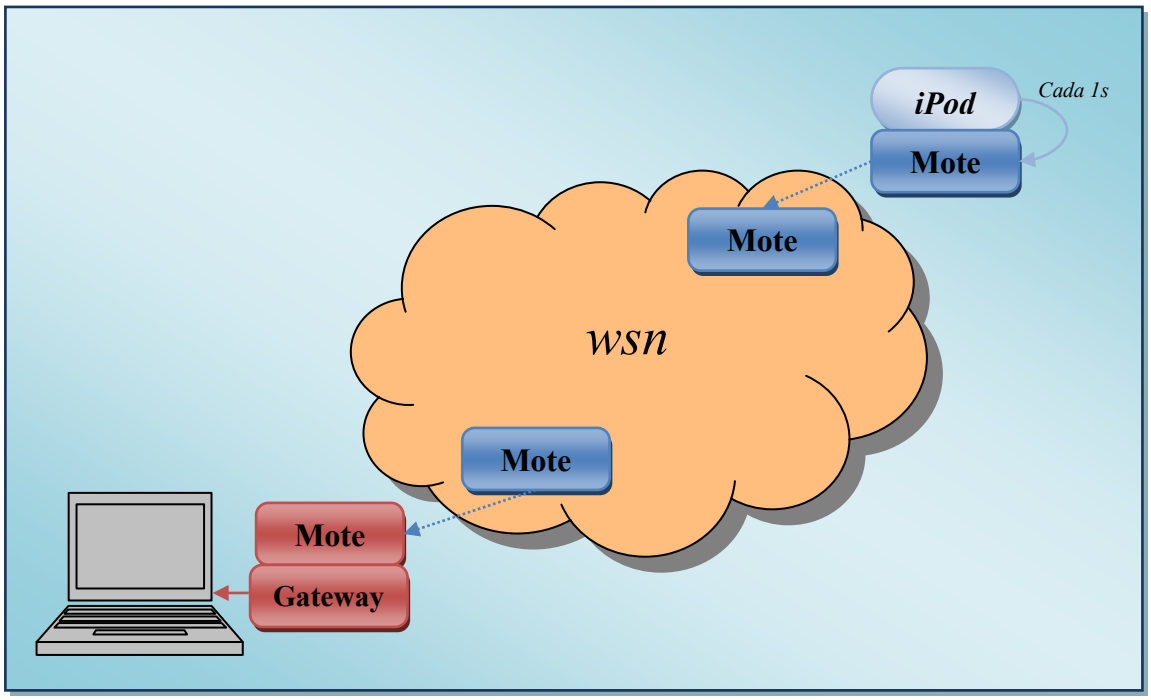
- Unión estática de componentes, vía sus interfaces. Esto aumenta la eficiencia en tiempos de ejecución.

- Los tipos de datos implementados en NesC son los que ya dispone C, pues es un lenguaje que derivó de éste, además de los tipos *uint8_t* (*unsigned integer* de 8 bits), *uint16_t* (lo mismo pero de 16 bits), *uint32_t* (32 bits), *uint64_t* (64 bits), *result_t* y *bool*.

2.4 Monitorización biométrica

Como ya se comento, uno de los objetivos del proyecto es la monitorización biométrica en tiempo real de un atleta en el transcurso de sus entrenamientos. Dicha monitorización se realizó mediante un sensor biométrico iPod y un sistema wireless que se comunica con una base a través de una red de sensores.

El sistema de toma de medidas se puede esquematizar como se muestra a continuación.



Esquema 2.18: Topología general de la sensorización biométrica.

2.4.1 Interfaz entre iPod y módulo wireless

Dado que el sensor biométrico iPod tiene una interfaz serie RS-232, cuyo rango de tensiones de salida es de 0 a 5 voltios, tienen que adaptarse a la interfaz TTL a 3.3 voltios del módulo wireless. Por lo tanto, fue necesario una pequeña placa de adaptación entre ellos que convierte las tensiones RS-232 a TTL y viceversa.

Además, durante el desarrollo del proyecto se llevaron a cabo dos distintas implementaciones del sistema de monitorización que comentaremos a continuación.

2.4.2 Implementación ZigBee

En una primera fase del proyecto, se desplego una red ZigBee para la toma de datos biométricos. Esta red está formada por dispositivos de comunicación XBee que se alimentan a 3.3 voltios, y por tanto, fue necesario realizar una pequeña placa adaptadora cuyo esquema se muestra a continuación.

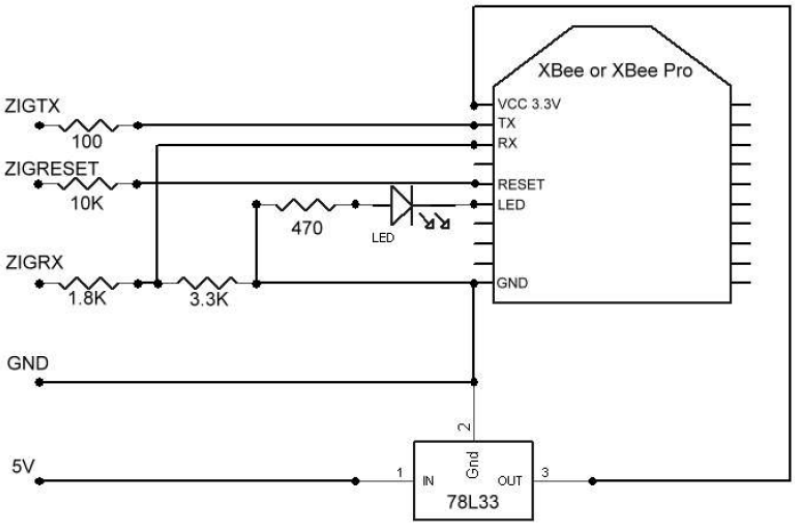


Figura 2.19 Adaptador de tensión para implementación ZigBee

No ha sido necesario crear un driver para el control y comunicación con el sensor iPod, simplemente se ha configurado el módulo XBee con la configuración del sensor para una correcta comunicación con el nodo de control.

Para la captura de los datos en el PC se ha creado una aplicación en C# de la plataforma .NET que vuelca los datos a un fichero Excel y cuyo código se adjunta junto con este documento.

2.4.3 Implementación TinyOS

Finalmente la monitorización biométrica se integró en la arquitectura de SAETA utilizado el sistema operativo embebido TinyOS y su lenguaje de programación nesC en los motes MICAz, que por su alimentación a 3,3 voltios también es necesaria su conversión.

En este caso, el esquema de la adaptación de tensiones que fue necesario para poder interconectar el sensor y el módulo de la unidad UE se muestra a continuación.

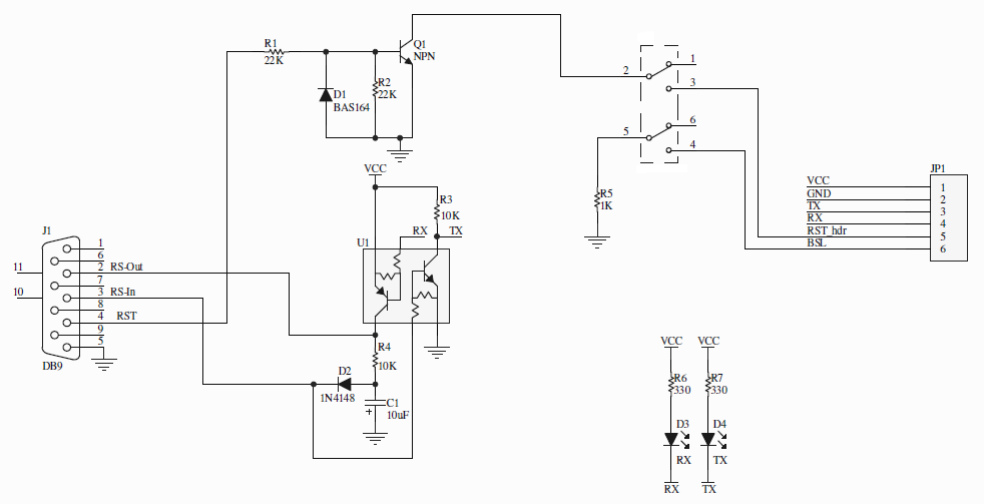


Figura 2.20 Adaptador de tensión para implementación TinyOS

Para la comunicación y control del sensor iPod por parte del mote MICAz ha sido necesario crear un driver (o componente) en nesC para TinyOS que utiliza la UART que disponen estos dispositivos wireless, como se aprecia a continuación en la tabla con la interfaz de su conector Hirose.

MICAz Sensor Interface.

Pin	Name	Description	Pin	Name	Description
1	GND	Ground	27†	UART_RXDO	UART_0 Receive
2	VSNR	Sensor Supply	28†	UART_TXDO	UART_0 Transmit
3	INT3	GPIO	29	PWO	GPIO/PWM
4	INT2	GPIO	30	PW1	GPIO/PWM
5	INT1	GPIO	31	PW2	GPIO/PWM
6	INT0	GPIO	32	PW3	GPIO/PWM
7††	CC_CCA	Radio Signal	33	PW4	GPIO/PWM
8†	LED3	Green LED	34	PW5	GPIO/PWM
9†	LED2	Yellow LED	35	PW6	GPIO/PWM
10†	LED1	Red LED	36†	ADC7	ADC CH7, JTAG TDI
11	RD	GPIO	37†	ADC6	ADC CH6, JTAG TDO
12	WR	GPIO	38†	ADC5	ADC CH5, JTAG
13	ALE	GPIO	39†	ADC4	ADC CH4, JTAG
14	PW7	GPIO	40	ADC3	GPIO/ADC CH3
15	USART1_CLK	USART1 Clock	41	ADC2	GPIO/ADC CH2
16††	PROG_MOSI	Serial Program MOSI	42	ADC1	GPIO/ADC CH1
17††	PROG_MISO	Serial Program MISO	43	ADC0	GPIO/ADC CH0
18††	SPI_CLK	SPI Serial Clock	44	THERM_PWR	Temp Sensor Enable
19	USART1_RXD	USART1 Receive	45	THRU1	Thru Connect 1
20	USART1_TXD	USART1 Transmit	46	THRU2	Thru Connect 2
21	I2C_CLK	I2C Bus Clock	47	THRU3	Thru Connect 3
22	I2C_DATA	I2C Bus Data	48††	RSTN	Reset (Neg.)
23	PWM0	GPIO/PWM0	49	PWM1B	GPIO/PWM1B
24	PWMIA	GPIO/PWM1A	50	VCC	Digital Supply
25	AC+	GPIO/AC+	51	GND	Ground
26	AC-	GPIO/AC-			

(†OK to use but has shared functionality. ††Do not use)

En el anexo se puede encontrar el código nesC del driver y de un ejemplo de uso.

3 Aprendizaje máquina

3.1 Introducción

En términos generales, el aprendizaje es considerado como una habilidad natural, mediante la cual se realiza un tipo de adaptación al entorno. Es frecuente encontrar en problemas de ingeniería, la necesidad de algún tipo de adaptación para su correcto funcionamiento en el entorno en el que operan. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información no estructurada suministrada en forma de ejemplos. Es, por lo tanto, un proceso de inducción del conocimiento.

Para esta tarea se precisan un conjunto de medidas o ejemplos asociado al proceso que se quiere modelar. Este tipo de aprendizaje, denominado aprendizaje inductivo, se convierte de hecho en un aprendizaje con ejemplos que es en el fondo un problema de aproximación de una función de la que se conoce únicamente un conjunto de puntos.

3.2 Tipos de aprendizaje máquina

Existen dos métodos fundamentales. El primero, conocido con el nombre de **aprendizaje no supervisado**, donde todo el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos formado tan sólo por entradas al sistema. No se tiene información sobre las categorías de esos ejemplos.

En el segundo tipo de aprendizaje, llamado **aprendizaje supervisado**, partimos de un conjunto de datos de entrenamiento (entradas) para las que observamos una serie de resultados (salidas) para un grupo de objetos (tales como personas). Utilizando estos datos se construye un modelo de predicción o aprendiz, que permitirá predecir los resultados que se obtendrán para un nuevo objeto. La calidad del aprendiz viene determinada por la precisión de estos resultados.

Este proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada. El supervisor comprueba la salida generada por el sistema y en el caso de que no coincida con la esperada, se procederá a modificar los pesos de las conexiones.

En este tipo de aprendizaje se suelen distinguir a su vez tres formas de llevarlo a cabo:

- **Aprendizaje por corrección de error:** Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos en la salida.

- **Aprendizaje por refuerzo:** este tipo de aprendizaje es más lento que el anterior y se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado; es decir, de no indicar durante el entrenamiento la salida exacta que se desea que proporcione la red ante una determinada entrada. Aquí la función del supervisor se reduce a indicar mediante una señal

de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito = +1 o fracaso = -1) y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.

- **Aprendizaje estocástico:** consiste básicamente en realizar cambios aleatorios en los valores de los pesos y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad. Una red que utiliza este tipo de aprendizaje es la red Boltzman Machine, ideada por Hinton, Ackley y Sejnowski en 1984 y la red Cauchy Machine desarrollada por Szu en 1986.

Se dispuso de un conjunto de muestras de entradas y salidas obtenidas de las pruebas de campo, por lo que el estudio se centrará en la metodología de aprendizaje supervisado, en la cual, tratando de que el sistema se adapte a los progresos realizados por los deportistas, nos centraremos en el subconjunto referido como aprendizaje por refuerzo.

4 Algoritmo de control del entrenamiento

El objetivo de este apartado es exponer el método de control de entrenamiento aeróbico implementado.

Control del entrenamiento aeróbico mediante sistemas de decisión de Markov: en este apartado se ha desarrollado un sistema de control que selecciona parámetros del entrenamiento aeróbico (en nuestro caso, se selecciona una pista de entrenamiento) con el fin de obtener los rangos de frecuencias cardíacas deseadas. El sistema de control emplea un sistema basado en programación dinámica, ya que así pueden obtenerse políticas óptimas que abarquen todas las etapas del entrenamiento.

A fin de obtener datos para el sistema de aprendizaje supervisado se han medido los parámetros de saturación sanguínea y pulsaciones durante diferentes sesiones de entrenamiento aeróbico a diferentes deportistas. Como resultado se obtuvo una base de conocimiento para el desarrollo del algoritmo. Asimismo, el módulo de decisión se han integrado en un prototipo junto con el sistema de inteligencia ambiental creado en el proyecto AISAS. De este modo, pueden obtenerse medidas ambientales y un control en tiempo real y para distintos usuarios de modo transparente. Se reciben y envían datos a través de la interfaz creada con una red de sensores bajo TinyOS.

4.1 Control del entrenamiento aeróbico

El objetivo buscado es mantener el ritmo cardíaco del usuario dentro de un rango de intensidad determinado. Caracterizar el ritmo cardíaco resulta una labor compleja, debido a la no linealidad de los mecanismos involucrados en la regulación cardiovascular [15]. Además, la relación entre ritmo cardíaco y ejercicio físico está influida por muchos factores, tales como los efectos de la temperatura y el ejercicio físico. Teniendo presentes dichas dificultades, en este apartado se plantea un problema restringido en el cual los usuarios entrenan (habitualmente ejercicios de carrera continuada) en un conjunto de pistas, cada una de ellas con una dificultad intrínseca distinta (por ejemplo, debido a una mayor pendiente o porque se corre en el exterior con más calor). Típicamente, los entrenadores seleccionan un número de vueltas a cada pista, pero no tienen un control directo sobre el efecto en el ritmo cardíaco de los usuarios.

El sistema seguido (ver Figura 4.1) proporciona un control explícito sobre los objetivos del entrenamiento aeróbico. El entrenador selecciona un programa concreto de entrenamiento, indicando los ritmos cardíacos y los tiempos que desea para cada jugador. El módulo de control selecciona entonces, a partir de los datos de la base de conocimiento, la mejor estrategia para cada jugador. El decisor empleado se basa en un esquema de programación dinámica mediante procesos de Markov. Éste tiene la ventaja de que las decisiones tienen en cuenta todo el horizonte restante de la prueba aeróbica (no sólo una etapa).

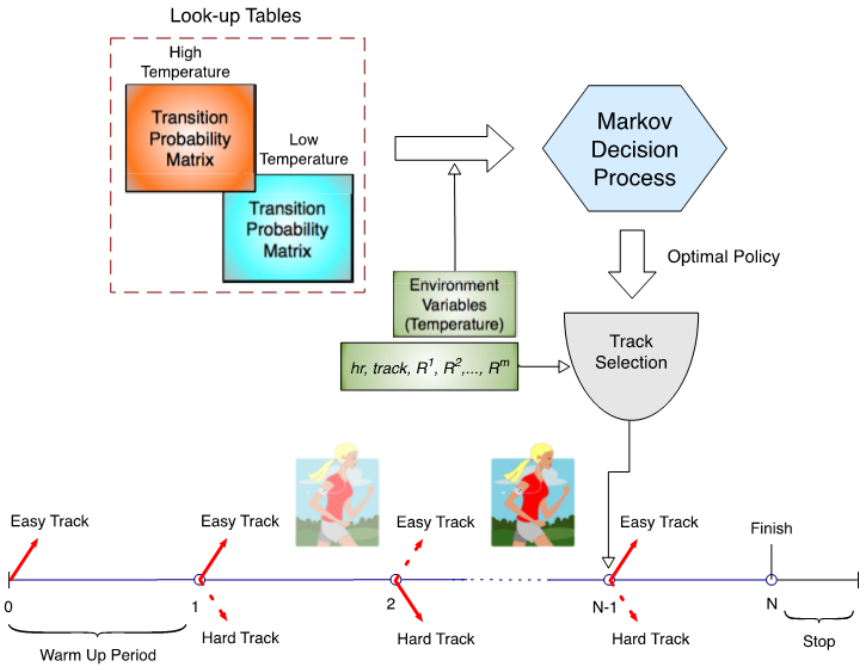


Figura 4.1 Esquema del entrenador virtual para entrenamiento aeróbico

4.1.1 Descripción matemática

El ritmo cardiaco (en adelante se notara con el acrónimo HR) se puede dividir en diferentes clases o niveles de intensidad en función del tipo de entrenamiento [16]:

1. **Bajo** (*Low*) si el HR está por debajo del 50% del HR máximo recomendado.
2. **Actividad moderada** (Mantenimiento/*Warm up*, MOD) si el HR está entre el 50% y el 60% del HR máximo recomendado.
3. **Control del Peso** (Quema grasas/*Fitness*, WEI) si el HR está entre el 60% y el 70% del HR máximo recomendado.
4. **Aeróbico** (resistencia cardiovascular/*Cardio-Training,* AE) si el HR está entre el 70% y el 80% del HR máximo recomendado.
5. **Anaeróbico** (funcional intenso, ANA) si el HR está entre el 80% y el 90% del HR máximo recomendado.
6. **VO2 MAX** (esfuerzo máximo, MAX) si el HR supera el 90% del HR máximo recomendado.

El HR máximo puede obtenerse mediante pruebas de intensidad o bien con aproximaciones como la de [Tan01]:

$$HR_{max} = 208 - 0.7 \times age$$

En la formulación utilizada en este estudio asumimos una partición de rangos de HR de m niveles no-solapados, consistentes, cada uno, en un conjunto $HR^i = [hr_i, hr_{i+1})$ para

$i = 1,\ldots,m$, donde hr_i denota el límite inferior de ritmo cardiaco asociado al rango i.

Tal y como hemos comentado, en nuestro escenario, los usuarios entrenan en un circuito con caminos o *tracks* de diferente nivel de dificultad, (t *tracks*). En cada fase de entrenamiento, seleccionan el camino a seguir. Una sesión de entrenamiento consiste, entonces, en una secuencia de N caminos. El objetivo de la sesión de entrenamiento podría ser recorrer todos los caminos manteniendo un HR dentro de un determinado rango, o llevar a cabo un entrenamiento multi-objetivo, por ejemplo realizando $N-4$ caminos en un rango HR^2 y los cuatro restantes en un HR^3. El HR dependerá, entre otros, de los siguientes aspectos: la condición física del atleta (HR), las fases de entrenamiento previas, la dureza de la pista, y las condiciones ambientales. Con estas variables, el sistema debe seleccionar el camino para cada fase de entrenamiento.

Por lo tanto el nodo de control debe seleccionar, en cada momento de toma de decisión, el mejor camino para llevar a cabo el objetivo de entrenamiento global. Estudios previos [14] consideran sólo el problema de seleccionar la mejor decisión para una fase de entrenamiento, con independencia de la evolución futura. Esto es, la optimización se efectúa sólo para la siguiente fase de entrenamiento (escenario *single-step*).

En este trabajo consideramos el caso de optimización multi-estado. Esto es, la selección del camino para completar los objetivos de entrenamiento, considerando la evolución futura de los atletas. Podemos formular este problema mediante programación dinámica, siguiendo la ecuación de Bellman.

$$J^n\left(x_n\right) = \max_{u_n}\left\{E\left\{r_n\left(x_n,u_n\right) + J^{n+1}\left(f_n\left(x_n,u_n\right)\right)\right\}\right\}, n = 0\ldots N$$

Donde:

- x_n: estado del sistema en la fase de entrenamiento n.
- u_n: control aplicado en la fase de entrenamiento n. Esto es, la "decisión" en esa fase.
- $J^n\left(x_n\right)$: valor de la función en la fase de entrenamiento n, partiendo desde el estado x_n
- $r_n\left(x_n,u_n\right)$: recompensa obtenida cuando el estado del sistema es x_n, y el control que se aplica es u_n.
- $f_n\left(x_n,u_n\right)$: función que genera el nuevo estado partiendo del estado x_n, y aplicando el control u_n

Esta ecuación recursiva expresa con claridad que el valor de la función en la fase de entrenamiento n, $J^n\left(x_n\right)$, puede calcularse como la suma de las recompensas esperadas en el estado x_n más el valor de la función para la siguiente fase de entrenamiento, $n+1$. Una estrategia óptima tiene la propiedad de que, independiente del estado y decisión iniciales,

las decisiones restantes constituyen una estrategia óptima, considerando el estado resultante de la primera decisión (principio de optimización de Bellman).

En nuestro problema, el estado del sistema x_n incluye la siguiente información:

- La clase de HR del usuario en el estado x_n (uno de los m tipos posibles), que denominamos como hr_{x_n}
- El camino seleccionado en el estado x_n (uno de los t caminos posibles), $track_{x_n}$
- La cantidad de vueltas restantes en cada clase de HR hasta completar el entrenamiento, que notaremos como $R^1_{x_n}, R^2_{x_n}, \ldots, R^m_{x_n}$

El control que se aplica en cada fase o periodo de entrenamiento, u_n, es el camino seleccionado (cada uno con un nivel de dificultad diferente). Hay que resaltar que el control también está incluido en el estado del sistema. Es necesario para calcular el valor de la recompensa.

En el caso general multi-objetivo, se deben efectuar R^1 caminos en HR^1, R^2 en HR^2, y así sucesivamente. Por tanto, $N = R^1 + R^2 + \cdots + R^m$. Siempre que un camino es recorrido con la clase HR "i" si $R^i > 1$ debe añadirse una recompensa positiva al valor de la función. De otra forma no hay recompensa. Por lo tanto, la función de recompensa para el problema es la siguiente.

$$r_n\left(x_n, u_n\right) = \begin{cases} 1, \ R^{hr_{x_n}} > 0 \\ 0, \ \text{resto} \end{cases}$$

Podemos ver como en este caso, la recompensa no depende ni del control seleccionado ni de la fase de entrenamiento, sino sólo del estado. Por lo tanto, la ecuación de Bellman puede reescribirse como sigue.

$$(1)\ J^n\left(x_n\right) = r\left(x_n\right) + \max_{u_n}\left\{E\left\{J^{n+1}\left(f_n\left(x_n, u_n\right)\right)\right\}\right\}, n = 0 \ldots N$$

Para resolver esta ecuación, debemos conocer el mapa de transiciones $f\left(\cdot\right)$. Sin embargo, las transiciones de un estado a otro no se pueden computar de forma analítica debido a la naturaleza aleatoria del HR en la práctica de deporte al aire libre. Para solventar esta limitación, podemos modelar nuestro sistema como un Proceso de Decisión de Markov (MDP). Esto es, las decisiones deben realizarse en un contexto de Markov, donde el sistema evoluciona de forma estocástica de un estado al siguiente. En un MDP se asumen conocidas las probabilidades $p_{ij}^{(n)}\left(u_n\right)$ de pasar de un estado i a un estado j, aplicando el control u_n en la fase de entrenamiento n, y generando las matrices de probabilidad de transición $\mathbf{P}^{(n)}\left(u_n\right)$. Esto es, debemos calcular:

$$p_{ij}^{(n)}(u_n) = p\left\{ \overbrace{\left(hr_i, \text{track}_i, R^1{}_i, R^2{}_i, \ldots, R^m{}_i\right)}^{\text{state } i} \rightarrow \overbrace{\left(hr_j, \text{track}_j, R^1{}_j, R^2{}_j, \ldots, R^m{}_j\right)}^{\text{state j}} / \left(u_n = \text{track}_j\right) \right\}$$

Donde $u_n = \text{track}_j$. En este caso podemos reescribir la ecuación (1) como:

$$J^n(x_n) = r(x_n) + \max_{u_n}\left\{\sum_{\forall j} p_{x_n,j}^{(n)}(u_n) J^{n+1}(j)\right\}, n = 0 \ldots N$$

En forma matricial:

$$J^n = \mathbf{r} + \max_{u_n}\left\{\mathbf{P}^{(n)}(u_n) J^{n+1}\right\}$$

Por lo tanto, el problema de obtener $f(\cdot)$ queda reducido al cálculo de las matrices de transición del sistema. Por simplicidad, en esta aproximación inicial hemos considerado una matriz de transición estática. Esto es, la matriz de transición no depende de la fase de entrenamiento n. En nuestros experimentos, hemos calculado las matrices de probabilidad de transición evaluando experimentalmente las frecuencias de los sucesos de transición. Esto resulta sencillo, ya que:

$$\text{(2)}\ p_{ij}^{(n)}(u_n) = \begin{cases} p\left\{\left(hr_i, \text{track}_i\right) \rightarrow \left(hr_j, \text{track}_j\right)\right\}, & \begin{matrix} R_i^{hr_j} - R_j^{hr_j} = 1 \\ R_i^k = R_j^k \ \forall k \neq hr_j \end{matrix} \\ 0, \text{ resto} \end{cases}$$

Por lo tanto, sólo necesitamos calcular lar probabilidades de transición de pasar desde $\left(hr_i, \text{track}_i\right)$ a $\left(hr_j, \text{track}_j\right)$.

Finalmente, la ecuación (1) queda expresada de la siguiente forma:

$$\text{(3)}\ J^n = \mathbf{r} + \max_{u_n}\left\{\mathbf{P}(u_n) J^{n+1}\right\}$$

Debido a que consideramos N fases de entrenamiento, el programa dinámico tiene un horizonte finito y el valor de la función para la última fase es $\mathbf{J}^N = \mathbf{r}$. Además, en un entrenamiento real el atleta comienza con una fase/periodo de calentamiento, que asumiremos sin recompensa, esto es, $\boldsymbol{r}_0 = \mathbf{0}$.

Si el estado inicial x_0 es conocido, el valor de la function se evalúa sencillamente como $J^0(x_0)$. Con una función de recompensa como la definida previamente, este valor podemos interpretarlo como la cantidad de caminos que, en media, un jugador efectuará en el rango de HR correcto si aplicamos el control óptimo. El entrenamiento resultará mejor conforme el valor se aproxime a N. En la siguiente sección evaluamos este modelo con un experimento sencillo, y comparamos los resultados con algunas estrategias no-óptimas.

4.1.2 Pruebas

El sistema requiere de la información obtenida de las sesiones de entrenamiento previas para construir las matrices de transición. Esos datos son utilizados como entrada del bloque MDP de la Figura 4.1, que selecciona la estrategia óptima, utilizando, para ello, la tabla que corresponde con las condiciones ambientales actuales. Podemos observar como la estrategia óptima consiste en una asociación entre todos los estados del sistema y sus *mejores* controles (los caminos o *tracks*). Esta forma de representar la estrategia óptima se denomina habitualmente como *lookup table* en la literatura sobre MDP. En cada momento de decisión, el bloque encargado de seleccionar el camino (*Track Selection*) obtiene de la tabla de datos (*lookup table*) el camino que debe seguir el usuario en función de su HR, del último camino seleccionado, y de los caminos restantes para cada clase de HR (los parámetros que determinan el estado actual).

Para las pruebas los usuarios han realizado un conjunto de pruebas de entrenamiento que permiten calcular las matrices de transición. En la Figura 4.2 podemos ver un ejemplo de una sesión de entrenamiento, donde se indican los distintos rangos para la frecuencia cardíaca, según la clasificación establecida anteriormente. Puede verse que las muestras de HR pueden pertenecer a un rango diferente de HR, incluso para la misma fase de entrenamiento. En este trabajo consideramos que las muestras en una fase de entrenamiento pertenecen a un tipo o rango de HR si las muestras dentro de ese rango son las predominantes.

Las pruebas realizadas tienen las siguientes características:

- Se han considerado solamente dos pistas, "*easy*" (o fácil) y "*hard*" (o difícil), atendiendo a su dificultad.

- Hay una fase inicial de calentamiento. Durante este periodo el usuario entrena en la pista *easy*. Los datos recogidos en esta fase son descartados (en nuestro modelo matemático no hay recompensa asociada para la fase de calentamiento).

- Cada sesión de pruebas ha consistido en 12 vueltas y en cada una de ellas el atleta puede recorrer el sector fácil o el difícil. El orden de entrenamiento fue el mismo para todas las sesiones (Tabla 1). Se monitorizó también la temperatura ambiente.

Tabla 1

Vuelta	1	2	3	4	5	6	7	8	9	10	11	12	13
Dureza	WU	E	E	E	H	E	E	H	H	E	H	H	H

- Con los datos recogidos de estos experimentos se computó la relación de transiciones de un estado a otro según la ecuación (2), y con ellas se construyeron las matrices de transición. De hecho se calcularon tres

matrices:

- Matriz de temperaturas bajas, que sólo tenemos en cuenta para las transiciones a baja temperatura (temperatura media del camino inferior a 25 grados Celsius).
- Matriz de temperaturas altas, que sólo tenemos en cuenta para las transiciones a alta temperatura (temperatura media del camino superior a 25 grados Celsius).
- Matriz absoluta, calculada utilizando la información de todas las transiciones, sin considerar la temperatura.

Además, para reducir la cantidad global de estados del sistema, que podría convertir el algoritmo en impracticable, sólo hemos utilizado tres clases de HR:

- HR^1, que recoge las clases de HR Baja, Moderada y Quema-Grasas.
- HR^2, que corresponde con la clase Aeróbica de HR.
- HR^3, que comprende las clases de HR Anaeróbica, VO2 MAX y Alto.

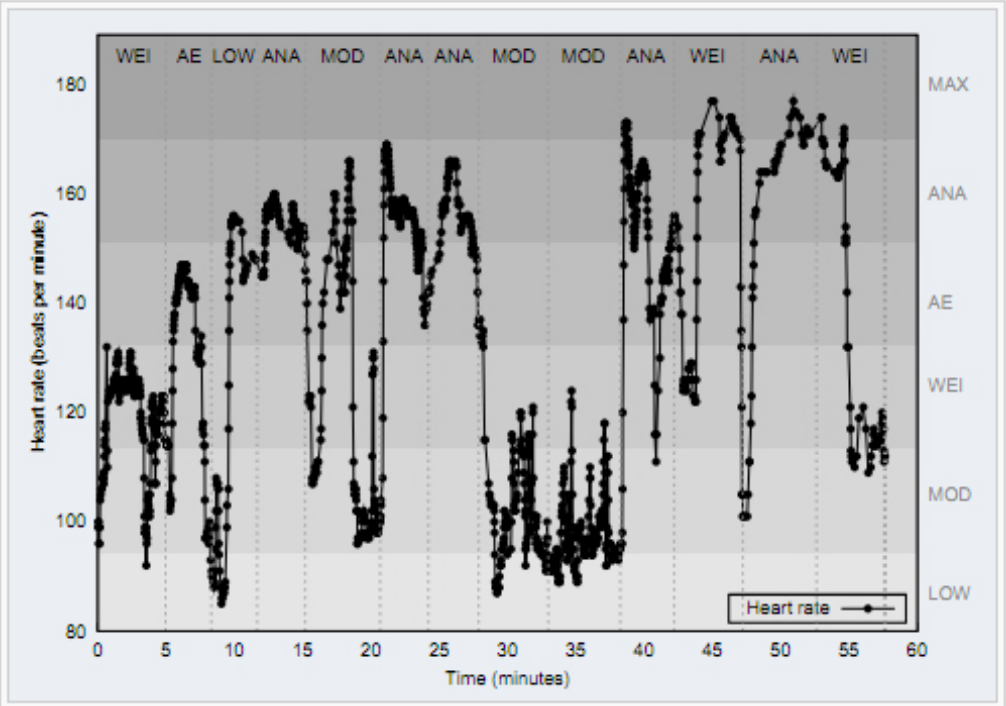


Figura 4.2 Ejemplo de una sesión de entrenamiento aeróbico

Hemos realizado un total de 12 sesiones de pruebas con los datos codificados tal y cómo se recibían de la red ambiental. Podemos ver un ejemplo en la Figura 4.3. La información presente en cada *byte* se describe en las Figura 4.4 y Figura 4.5

#TIEMPO	BYTE1	BYTE2	BYTE3
18:30:48	128	94	98
18:30:49	128	93	97
18:30:50	128	99	97
18:30:51	128	103	97
18:30:52	128	106	97
18:30:53	128	106	97
18:30:54	128	106	97

Figura 4.3 Ejemplo de datos sin procesar

Byte 1 - Status							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
1	SNSD	OOT	LPRF	MPRF	ARTF	HR8	HR7
***Note**: Bit 7 is always set							

Byte 2 - Heart Rate							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	HR6	HR5	HR4	HR3	HR2	HR1	HR0
***Note**: Bit 7 is always clear							

Byte 3 - SpO2							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	SP6	SP5	SP4	SP3	SP2	SP1	SP0
***Note:** Bit 7 is always clear							

Figura 4.4 Contenido de los *bytes*

SNSD:	Sensor Disconnect	Sensor is not connected to oximeter or sensor is inoperable.
OOT:	Out Of Track	An absence of consecutive good pulse signals.
LPRF:	Low Perfusion	Amplitude representation of low/no signal quality.
MPRF:	Marginal Perfusion	Amplitude representation of low/marginal signal quality.
ARTF:	Artifact	Indicated artifact condition on each pulse.
HR8 – HR0:	Heart Rate	Standard 4-beat average values without display holds.
SP6 – SP0:	SpO_2	Standard 4-beat average values without display holds.

Figura 4.5 Descripción del contenido

Estos datos se han procesado para cada tramo para obtener la siguiente información:

- TIEMPO_INICIO: Tiempo de inicio del tramo
- TIEMPO FINAL: Tiempo de finalización del tramo
- TIEMPO_TOTAL: Tiempo total del tramo
- TIPO: Tipo de prueba [F: Fácil, D: Difícil, Calentar: Calentamiento]
- NUM_MUESTRAS: Número de muestras recibidas en ese tramo
- PULSO_MEDIO: Valor medio de la frecuencia cardíaca por tramo

- PULSO_VARIANZA: Varianza de la frecuencia cardíaca por tramo
- SpO2_MEDIA: Consumo medio de oxígeno por tramo
- SpO2_VARIANZA: Varianza del consumo de oxígeno por tramo

Podemos ver un ejemplo en la Figura 7.5.

#TIEMPO_INICIO	TIEMPO_FINAL	TIEMPO_TOTAL	TIPO	NUM_MUESTRAS	PULSO_MEDIO	PULSO_VARIANZA	SpO2_MEDIA	SpO2_VARIANZA
18:30:47	18:35:33	47	Calentar	204	127,1715686	183,2009564	97,68137255	0,217103998
18:35:33	18:39:24	50	F	108	150,462963	8,563443073	97,2037037	0,180727023
18:39:24	18:43:25	1	F	73	151,3013699	69,68999812	96,24657534	0,185775943
18:43:25	18:47:26	1	F	60	152,4166667	105,2430556	96,25	0,2875
18:47:26	18:53:10	44	D	56	164,25	525,8660714	95,44642857	0,747130102
18:53:10	18:56:56	45	F	89	141,4494382	438,2923873	95,5505618	0,247443505
18:56:56	19:00:34	39	F	59	162,6610169	183,3088193	95,49152542	0,249928182
19:00:34	19:06:39	5	D	177	171,4293785	206,9907753	95,51412429	0,37409429
19:06:39	19:12:19	39	D	170	126,0411765	1229,145363	95,35882353	0,265363322
19:12:19	19:17:07	48	F	87	130,816092	650,1960629	95,20689655	0,279032897
19:17:07	19:22:32	26	D	152	109,2828947	314,9791811	94,71052632	0,297783934
19:22:32	19:28:18	46	D	147	125,5034014	561,9370633	94,60544218	0,279698274
19:28:18	19:34:20	1	D	151	116,192053	510,4995395	94,78145695	0,170781983

Figura 4.6 Ejemplo de archivo procesado

4.1.3 Validación

El rendimiento del programa dinámico puede calcularse aplicando las matrices de transición de los experimentos en el método expuesto en el apartado anterior. En él se indicaba como el valor de la función representa la cantidad de caminos que se ejecutan con éxito en alguna de las clases de HR de la sesión de entrenamiento. Los resultados están descritos en esta sección. Además, la estrategia óptima es comparada con cada una de las siguientes:

- Estrategia fácil, seleccionando siempre el camino fácil o "*easy track*" del circuito
- Estrategia difícil, seleccionando siempre el camino difícil o "*hard track*" del circuito
- Estrategia peor, puede calcularse fácilmente sustituyendo el operador *max* por el operador *min* en el cómputo de la ecuación (3).

El estado inicial siempre es $x_0 = \left(HR^1, Easy, R^1, R^2, R^3 \right)$, donde R^1, R^2, R^3 denotan la configuración de entrenamiento deseada. Por tanto, el valor de la función para la prueba completa se calcula calculando $\mathbf{J}^0\left(x_0\right)$. Remarquemos que esta métrica del rendimiento es un escalar, ya que todos los atletas comienzan la prueba desde el mismo estado inicial. En las siguientes secciones evaluamos diferentes programas de entrenamiento. Además, en todas las pruebas, la temperatura se ha considerado constante durante todo el entrenamiento.

4.1.3.1 Optimización mono-objetivo

El este caso el objetivo del programa de entrenamiento es $(R^1 = 0, R^2 = i, R^3 = 0)$

lo que equivale a realizar todas las pistas en el régimen aeróbico (*cardio-training*). La Figura 4.7 muestra los resultados para $i = 1, \cdots, 10$ utilizando la matriz absoluta. Tal y como se esperaba, la estrategia óptima consigue mejor rendimiento que las no óptimas. Además, la figura proporciona una comparación de la estrategia óptima utilizando una matriz específica (alta y baja temperatura), o con la matriz absoluta. Utilizando cada una de estas matrices los resultados mejoran considerablemente especialmente para condiciones de temperatura alta.

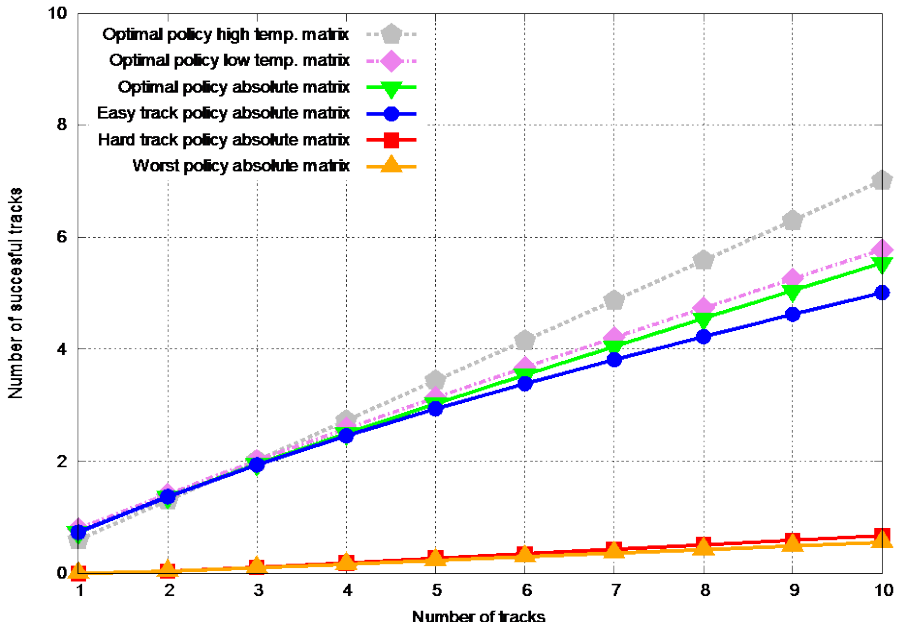


Figura 4.7 Entrenamiento mono-objetivo (0, i, 0). Comparación de estrategias. Resultados de la estrategia óptima con matrices específicas y absoluta

Este experimento demuestra que utilizar información ambiental mejora el resultado y resulta, por tanto, recomendable.

4.1.3.2 Optimización multi-objetivo

Se han estudiado adicionalmente dos tipos de entrenamiento multi-objetivo:

- $(R^1 = 10 - i, R^2 = i, R^3 = 0)$. En este caso, 10 vueltas deben completarse con una distribución diferente en dos clases de HR. La Figura 4.8 muestra los resultados. La tendencia resulta similar al del experimento mono-objetivo, siendo aún mayor la diferencia entre la estrategia óptima y las no-óptimas.

- $(R^1 = 7 - i, R^2 = i, R^3 = 3)$. En este caso realizamos siempre 7 vueltas y una distribución de HR de tres clases. La Figura 4.9 muestra los resultados. De la misma forma que en anteriores experimentos los resultados muestra una tendencia similar, confirmando los beneficios de nuestro planteamiento.

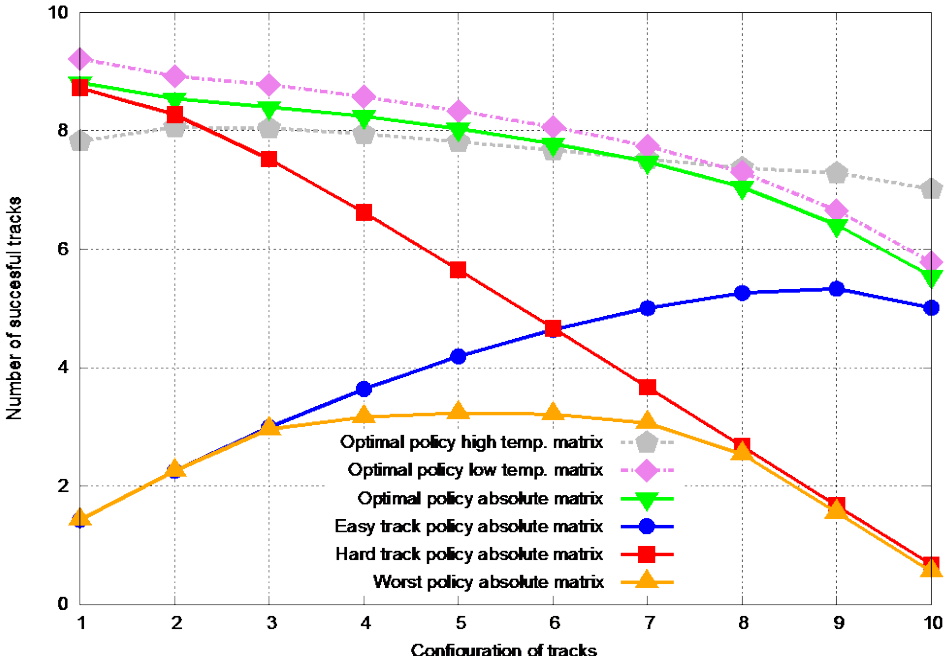


Figura 4.8 Entrenamiento multi-objetivo (10-i, i, 0). Comparación de estrategias. Resultados de la estrategia óptima con matrices específica y absoluta

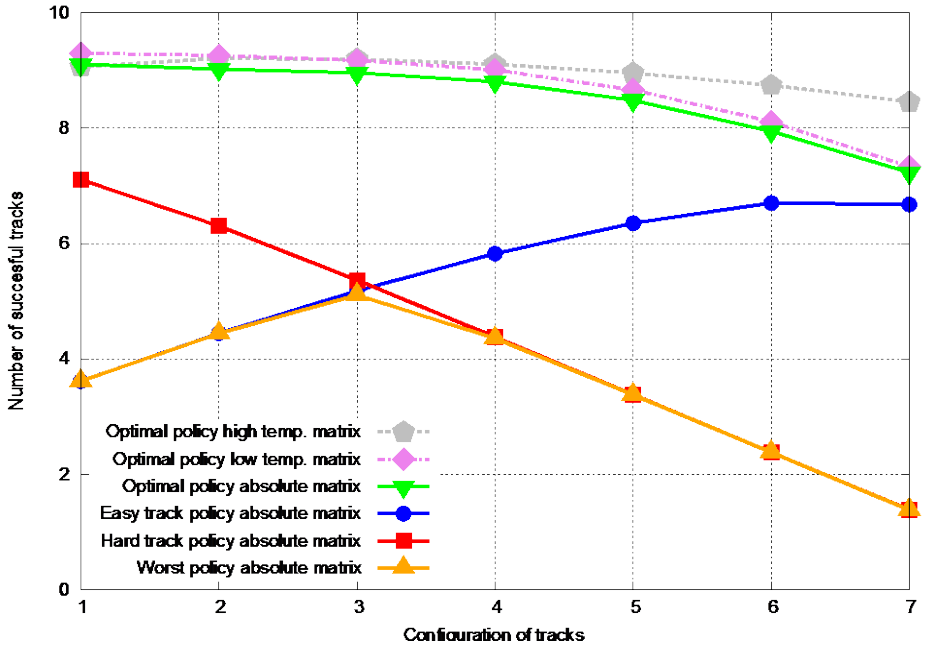


Figura 4.9 Entrenamiento multi-objetivo (7-i, i, 3). Comparación de estrategias. Resultados de la estrategia óptima con matrices específica y absoluta

5 Conclusiones

A modo de resumen, el módulo de control ha presentado un marco teórico para aplicar la metodología de programación dinámica al entrenamiento aeróbico en circuitos prefijados. El cómputo de estrategias óptimas requiere el cálculo de matrices de transición de los procesos de decisión de Markov. Estas matrices se han obtenido experimentalmente mediante medidas en entrenamientos, lo que nos ha permitido evaluar nuestra metodología para diferentes objetivos.

Por lo cual podemos concluir que los resultados demuestran las ventajas de las estrategias óptimas frente a las demás, así como la necesidad de monitorizar las condiciones ambientales para mejorar el rendimiento del sistema.

6 Líneas futuras

Como posibles mejoras para el sistema podríamos enumerar las siguientes, que pueden afrontarse mediante el empleo de técnicas de inteligencia ambiental y de aprendizaje por computador.

- **Control posicional y motriz de los jugadores** para seguir la ejecución táctica de las jugadas durante los entrenamientos/partidos y **permitir** así **la corrección de errores de posicionamiento**.

- **Gestión masiva de datos** biométricos y ambientales, para **el tratamiento común de múltiples fuentes de datos heterogéneas**.

7 BIBLIOGRAFÍA

[1] ZigBee Alliance

http://www.zigbee.org/

ZigBee/IEEE 802.15.4 Summary

http://www.eecs.berkeley.edu/~csinem/academic/publications/zigbee.pdf

[2] **Dung Van Dinh, Minh Duong Vuong, Hung Phu Nguyen, Hoa Xuan Nguyen, "***Wireless Sensor Actor Networks And Routing Performance Analysis*".

[3] **Jamal N. Al-Karaki, Raza Ul-Mustafa, Ahmed E. Kamal, "***Data Aggregation in Wireless Sensor Networks - Exact and Approximate Algorithms*". Proceedings of IEEE Workshop on High Performance Switching and Routing (HPSR) Phoenix, Arizona, USA. April 2004.

[4] **Ian F. Akyildiz, Ismail H Kasimoglu,** "*Wireless sensor and actor networks: research challenges*". Broadband and Wireless Networking Laboratory. Georgia Institute of Technology, May 2004.

[5] Crossbow Technologies Inc.

http://www.xbow.com/

[6] Guía de referencia de productos Crossbow

http://www.xbow.com/Support/Support_pdf_files/Product_Feature_Reference_Chart.pdf

[7] Nonin Medical Company

http://www.nonin.com

[8] Mote MPR2400 "MICAz" Download MICAz Datasheet

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf

[9] Chipcon CC2420 Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee(TM) Ready RF Transceiver

http://focus.ti.com/lit/ds/symlink/cc2420.pdf

[10] Placa de desarrollo MIB520-CB (Download Datasheet)

http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB520_Datasheet.pdf

[11] Sensor iPod NONIN

http://www.nonin.com/documents/ipod%20Specifications.pdf

[12] TinyOS

http://www.tinyos.net/

[13] NesC: A Programming Language for Deeply Networked Systems

http://nescc.sourceforge.net/

[14] **P. López-Matencio, J. Vales-Alonso, F.J. González-Castaño, J.L. Sieiro, J.J. Alcaraz**, "*Ambient Intelligence Assistant for Running Sports Based on k-NN Classifiers*", Congreso HSI 2010. Lugar de celebración: Rzeszow, Poland.

[15] **Huikuri H, Makikallio T, Perkiomaki J**, "*Measurement of heart rate variability by methods based on nonlinear dynamics.*" Jour-nal of Electrocardiology, vol. 36, pp. 95–99. 2003.

[16] **Haskell W, Lee I, et al.**,"*Physical activity and public health: updated recommendation for adults from the American College of Sports Medicine and the American Heart Association.*" Circulation, 116(9). 2007.

8 Anexo

8.1 Especificaciones del sensor iPod NONIN

Integrated Pulse Oximetry Device
Specifications

1.	Oxygen Saturation Range	0 to 100%	
2.	Pulse Rate Range	18 to 300 pulses per minute	
3.	Measurement Wavelengths	Red - 660 Nanometers Infrared - 925 Nanometers	
4.	Accuracy		
	SpO_2 (70-100%) (±1 SD)◆	No Motion	
		- Adults	±2 digits
		Motion	
		- Adults	±3 digits
		Low Perfusion	
		- Adults	±3 digits
	Heart Rate	No Motion (18 - 300 BPM)	
		- Adults	±3 digits
		Motion (40 - 240 BPM)	
		- Adults	±5 digits
		Low Perfusion (40 - 240 BPM)	
		- Adults	±3 digits
5.	Temperature		
	a) Operating	0° C to +50° C	
	b) Non Operating	-30° C to +50° C	
6.	Humidity		
	a) Operating	10 to 90% Non Condensing	
	b) Non Operating	10 to 95% Non Condensing	
7.	Power Draw	60 mW - typical operating	
8.	Voltage Input	2 to 6 volts dc operating Note: Sensor is not isolated from input voltage	
9.	Output Digital Signals	0 - 5 volts (nominally)	
10.	Patient Isolation	Meets IEC 60601-1 Dielectric Withstand	
11.	Leakage Current	Not applicable	
12.	Dimensions	1.26" x 1.25" x 2" (32 x 32 x 51mm)	
13.	Weight	100g (including 1m cable and connector)	
14.	Ruggedness immersion		
	a) Shock	Per IEC 68-2-27	
	b) Vibration	Mil-standard 810C, method 514-2	

◆ Standard Deviation is a statistical measure: up to 32% of the readings may fall outside these limits.

4106-000 REV 06

INPUTS:
Red Wire = V+ (2-6VDC, 60mW typical)
Black Wire = Ground
Cable Shield = Ground (Both Black wire and cable shield must be attached to ground on the host device)
Note: Sensor is not isolated from input voltage.

OUTPUTS:
Green Wire = Serial Output (TTL)
Blue Wire = Serial Output (RS-232)

FORMATTING OPTIONS:

PART #	MODEL #	SERIAL DATA FORMAT #	LENGTH	CONNECTOR
3830-001	3211	#1	1m	No
3830-101	3211	#1	1m	Yes
3831-001	3212	#2	1m	No
3831-101	3212	#2	1m	Yes

SERIAL DATA FORMAT #1:

1) Serial format 9600, n, 8, 1
2) Rate Sends 3 bytes of data once a second.
3) Data

1st byte = **Status**

BIT 7 = **ALWAYS SET TO "1"**
BIT 6 = SENSOR DISCONNECTED, SET IF TRUE
BIT 5 = OUT OF TRACK, SET IF TRUE
BIT 4 = LOW PERFUSION, SET IF TRUE
BIT 3 = MARGINAL PERFUSION, SET IF TRUE
BIT 2 = BAD PULSE, SET IF TRUE
BIT 1 = HEART RATE BIT 8
BIT 0 = HEART RATE BIT 7

2nd byte = **Heart Rate** (511 = bad data) BIT "7" IS ALWAYS SET TO "0".
HEART RATE DATA = BITS 0 - 6
PLUS BITS 0 & 1 OF THE STATUS BYTE TO PROVIDE 9 BITS OF RESOLUTION.

3rd byte = **SpO2** (127 = bad data)

SERIAL DATA FORMAT #2:

1) Serial format — 9600, n, 8, 1
2) Rate — Sends 5 bytes of data 75 times a second.
3) Data

a. HR value bits 7&8 (128-511), 511 = bad data	1 byte	3 times a second
b. HR value bits 0-6 (0-127)	1 byte	3 times a second
c. SpO2 value 0 - 100	1 byte	3 times a second
d. Firmware revision level	1 byte	3 times a second
e. Status byte 128 - 255	1 byte	75 times a second

Bit 0 frame Sync, set for 1 of 25, clear for 2-25 of 25
Bit 1 green perfusion, set if true only during pulse
Bit 2 red perfusion, set if true only during pulse
Bit 3 sensor alarm, set if true
Bit 4 out of track, set if true
Bit 5 bad pulse, set if true
Bit 6 sensor disconnected, set if true
Bit 7 always set

Note: bits 1 & 2 are set for yellow perfusion.

f. Plethysmographic pulse value 0 - 254	1 byte	75 times a second
g. Sync character (01)	1 byte	75 times a second
h. Checksum = byte 1 + byte 2 + byte 3 + byte 4	1 byte	75 times a second

Extended Averaging Data

i. E-HR value bits 7&8 (128-511), 511 = bad data	1 byte	3 times a second
j. E-HR value bits 0-6 (0-127)	1 byte	3 times a second
k. E-SpO2 value 0 - 100	1 byte	3 times a second

Non-Slew Limited with Standard Averaging

l. SpO2 Slew value 0 - 100, 127 = bad data	1 byte	3 times a second

Beat to Beat Value (No Averaging or Slew Limiting)

m. SpO2 B-B value 0 - 100, 127 = bad data	1 byte	3 times a second

Display Data

SpO2 Display Value with Standard Averaging

n. 0-11, 127 = bad data	1 byte	3 times a second

E-SpO2-D Display Value with Extended Averaging

o. 0-100, 127 = bad data	1 byte	3 times a second

HR-D-MSB Display Value with Standard Averaging

p.HR Value bits 7&8, 511 = bad data	1 byte	3 times a second

HR-D-LSB Display Value with Standard Averaging

q. HR Value bits (0-6) (0-127)	1 byte	3 times a second

E-HR-D-MSB Display Value with Extended Averaging

r. HR Value bits 7&8, 511 = bad data	1 byte	3 times a second

E-HR-D-LSB Display Value with Extended Averaging

s. HR Value Bits 0-6 (0-127)	1 byte	3 times a second

Data is sent in the following format

Hz	BYTE					Hz	BYTE					Hz	BYTE				
1/75	1	2	3	4	5	1/75	1	2	3	4	5	1/75	1	2	3	4	5
1	01	STATUS	PLETH	HR MSB	CHK	26	01	STATUS	PLETH	HR MSB	CHK	51	01	STATUS	PLETH	HR MSB	CHK
2	01	STATUS	PLETH	HR LSB	CHK	27	01	STATUS	PLETH	HR LSB	CHK	52	01	STATUS	PLETH	HR LSB	CHK
3	01	STATUS	PLETH	SpO2	CHK	28	01	STATUS	PLETH	SpO2	CHK	53	01	STATUS	PLETH	SPO2	CHK
4	01	STATUS	PLETH	REV	CHK	29	01	STATUS	PLETH	REV	CHK	54	01	STATUS	PLETH	REV	CHK
5	01	STATUS	PLETH	*	CHK	30	01	STATUS	PLETH	*	CHK	55	01	STATUS	PLETH	*	CHK
6	01	STATUS	PLETH	*	CHK	31	01	STATUS	PLETH	*	CHK	56	01	STATUS	PLETH	*	CHK
7	01	STATUS	PLETH	*	CHK	32	01	STATUS	PLETH	*	CHK	57	01	STATUS	PLETH	*	CHK
8	01	STATUS	PLETH	*	CHK	33	01	STATUS	PLETH	*	CHK	58	01	STATUS	PLETH	*	CHK
9	01	STATUS	PLETH	*SpO2-D*	CHK	34	01	STATUS	PLETH	*SpO2-D*	CHK	59	01	STATUS	PLETH	*SpO2-D*	CHK
10	01	STATUS	PLETH	*SpO2 Slew*	CHK	35	01	STATUS	PLETH	*SpO2 Slew*	CHK	60	01	STATUS	PLETH	*SpO2 Slew*	CHK
11	01	STATUS	PLETH	*SpO2 B-B*	CHK	36	01	STATUS	PLETH	*SpO2 B-B*	CHK	61	01	STATUS	PLETH	*SpO2 B-B*	CHK
12	01	STATUS	PLETH	*	CHK	37	01	STATUS	PLETH	*	CHK	62	01	STATUS	PLETH	*	CHK
13	01	STATUS	PLETH	*	CHK	38	01	STATUS	PLETH	*	CHK	63	01	STATUS	PLETH	*	CHK
14	01	STATUS	PLETH	*E-HR MSB*	CHK	39	01	STATUS	PLETH	*E-HR MSB*	CHK	64	01	STATUS	PLETH	*E-HR MSB*	CHK
15	01	STATUS	PLETH	*E-HR LSB*	CHK	40	01	STATUS	PLETH	*E-HR LSB*	CHK	65	01	STATUS	PLETH	*E-HR LSB*	CHK
16	01	STATUS	PLETH	*E-SpO2*	CHK	41	01	STATUS	PLETH	*E-SpO2*	CHK	66	01	STATUS	PLETH	*E-SpO2*	CHK
17	01	STATUS	PLETH	*E-SpO2-D*	CHK	42	01	STATUS	PLETH	*E-SpO2-D*	CHK	67	01	STATUS	PLETH	*E-SpO2-D*	CHK
18	01	STATUS	PLETH	*	CHK	43	01	STATUS	PLETH	*	CHK	68	01	STATUS	PLETH	*	CHK
19	01	STATUS	PLETH	*	CHK	44	01	STATUS	PLETH	*	CHK	69	01	STATUS	PLETH	*	CHK
20	01	STATUS	PLETH	*HR-D MSB*	CHK	45	01	STATUS	PLETH	*HR-D MSB*	CHK	70	01	STATUS	PLETH	*HR-D MSB*	CHK
21	01	STATUS	PLETH	*HR-D LSB*	CHK	46	01	STATUS	PLETH	*HR-D LSB*	CHK	71	01	STATUS	PLETH	*HR-D LSB*	CHK
22	01	STATUS	PLETH	*E-HR-D MSB*	CHK	47	01	STATUS	PLETH	*E-HR-D MSB*	CHK	72	01	STATUS	PLETH	*E-HR-D MSB*	CHK
23	01	STATUS	PLETH	*E-HR-D LSB*	CHK	48	01	STATUS	PLETH	*E-HR-D LSB*	CHK	73	01	STATUS	PLETH	*E-HR-D LSB*	CHK
24	01	STATUS	PLETH	*	CHK	49	01	STATUS	PLETH	*	CHK	74	01	STATUS	PLETH	*	CHK
25	01	STATUS	PLETH	*	CHK	50	01	STATUS	PLETH	*	CHK	75	01	STATUS	PLETH	*	CHK

* Undefined

Input/Output Formatting Options

INPUTS

Red Wire = V+ (2-6VDC, 60 mw typical)
Black Wire = Ground
Cable Shield = Ground (both Black wire and Cable Shield must be attached to ground on the host device)
Yellow Wire = ECG Sync (Optional)

Note: Sensor is not isolated from input voltage.

OUTPUTS

Green Wire = Serial Output

FORMATTING OPTIONS

Order #	Model #	Serial Format #	With Connector
3873-001	3011	#1	No
3873-002	3012	#2	No
3873-101	3011	#1	Yes
3873-202	3012	#2	Yes

SERIAL DATA FORMAT #1

Packet Description

Three bytes of data are transmitted 1 once per second.

Byte 1 - Status							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
1	SNSD	OOT	LPRF	MPRF	ARTF	HR8	HR7
***Note**: Bit 7 is always set							

Byte 2 - Heart Rate							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	HR6	HR5	HR4	HR3	HR2	HR1	HR0
***Note**: Bit 7 is always clear							

Byte 3 - SpO2							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
0	SP6	SP5	SP4	SP3	SP2	SP1	SP0
***Note:** Bit 7 is always clear							

The following are all active high:

SNSD:	Sensor Disconnect	– Sensor is not connected to oximeter or sensor is inoperable.
OOT:	Out Of Track	– An absence of consecutive good pulse signals.
LPRF:	Low Perfusion	– Amplitude representation of low signal quality (holds for entire duration).
MPRF:	Marginal Perfusion	– Amplitude representation of medium signal quality (holds for entire duration).
ARTF:	Artifact	– A detected pulse beat didn't match the current pulse interval.
HR8 – HR0:	Heart Rate	– Standard 4-beat average values not including display holds.
SP6 – SP0:	SpO_2	– Standard 4-beat average values not including display holds.

When SpO_2 and HR cannot be computed, the system will send a missing data indicator. For missing data, the HR equals 511 and the SpO_2 equals 127.

Serial Data Format #2:

Packet Description

A frame consists of 5 bytes; a packet consists of 25 frames. Three packets (75 frames) are transmitted each second.

Frame

Packet

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
1	01	STATUS	PLETH	HR MSB	CHK
2	01	STATUS	PLETH	HR LSB	CHK
3	01	STATUS	PLETH	SpO_2	CHK
4	01	STATUS	PLETH	REV	CHK
5	01	STATUS	PLETH	reserved	CHK
6	01	STATUS	PLETH	reserved	CHK
7	01	STATUS	PLETH	reserved	CHK
8	01	STATUS	PLETH	reserved	CHK
9	01	STATUS	PLETH	SpO2-D	CHK
10	01	STATUS	PLETH	SpO_2 Fast	CHK
11	01	STATUS	PLETH	SpO_2 B-B	CHK
12	01	STATUS	PLETH	reserved	CHK
13	01	STATUS	PLETH	reserved	CHK
14	01	STATUS	PLETH	E-HR MSB	CHK
15	01	STATUS	PLETH	E-HR LSB	CHK
16	01	STATUS	PLETH	E-SpO_2	CHK
17	01	STATUS	PLETH	E-SpO_2-D	CHK
18	01	STATUS	PLETH	reserved	CHK
19	01	STATUS	PLETH	reserved	CHK
20	01	STATUS	PLETH	HR-D MSB	CHK
21	01	STATUS	PLETH	HR-D LSB	CHK
22	01	STATUS	PLETH	E-HR-D MSB	CHK
23	01	STATUS	PLETH	E-HR-D LSB	CHK
24	01	STATUS	PLETH	reserved	CHK
25	01	STATUS	PLETH	reserved	CHK

Notes:

- Byte number 1 in each frame is set to a value of 1.
- Reserved bytes are undefined.

Byte 2 - Status							
BIT7	**BIT6**	**BIT5**	**BIT4**	**BIT3**	**BIT2**	**BIT1**	**BIT0**
1	SNSD	ARTF	OOT	SNSA	YPRF		SYNC
					RPRF	GPRF	
***Note**: Bit 7 is always set.							

The following are all active high:

SNSD:	Sensor Disconnect	– Sensor is not connected to oximeter or sensor is inoperable.
ARTF:	Artifact	– A detected pulse beat didn't match the current pulse interval
OOT:	Out Of Track	– An absence of consecutive good pulse signals
SNSA:	Sensor Alarm	– Sensor is providing unusable data for analysis
RPRF:	Red Perfusion	– Amplitude representation of low signal quality (occurs only during pulse)
YPRF:	Yellow Perfusion	– Amplitude representation of medium signal quality (occurs only during pulse)
GPRF:	Green Perfusion	– Amplitude representation of high signal quality (occurs only during pulse)
SYNC:	Frame Sync	(occurs 1 of 25)

Generic HR Format:

	7	6	5	4	3	2	1	0
HR MSB	X	X	X	X	X	X	HR8	HR7

	7	6	5	4	3	2	1	0
HR LSB	X	HR6	HR5	HR4	HR3	HR2	HR1	HR0

Generic SpO_2 Format:

	7	6	5	4	3	2	1	0
SPO_2	X	SP6	SP5	SP4	SP3	SP2	SP1	SP0

HR: 4-beat average values in standard mode.

SpO_2: 4-beat average values in standard mode.

HR-D: 4-beat average displayed values in display mode

SpO_2-D: 4-beat average displayed values in display mode

SpO_2 Fast: Non-slew limited saturation with 4-beat averaging in standard mode.

SpO_2: B-B: Un-averaged, non-slew limited, beat to beat value in standard mode.

E-HR: 8-beat average values in standard mode.

E-SpO_2: 8-beat average values in standard mode.

E-HR-D: 8-beat average displayed values in display mode

E-SpO_2-D: 8-beat average displayed values in display mode

PLETH: 8-Bit Plethysmographic Pulse Amplitude

SREV: Oximeter Firmware Revision Level

CHK: Checksum = (Byte 1) + (Byte 2) + (Byte 3) + (Byte 4) modulo 256

When SpO_2 and HR cannot be computed, the system will send a missing data indicator. For missing data, the HR equals 511 and the SpO_2 equals 127.

Mode	In Track	Out of Track
Standard	SpO_2 and pulse rate updated on every pulse beat	SpO_2 and Heart Rate values are set to missing data values and out of track indicated.
Display	SpO_2 and pulse rate updated every 1.5 seconds	Last in track values transmitted for ten seconds and out of track indicated. After ten seconds, values are set to missing data values.

8.2 Código del driver para la monitorización biométrica.

```
//Nonin.h
#ifndef NONIN_H
#define NONIN_H

enum {
  NONIN_FRAME_SIZE = 3,
};
/* #define SDA_TYPE_SP02 0 */
/* #define SDA_TYPE_PULSE 1 */

struct _nonin_frame_t {

      // Serial Format #1
      // Three bytes of data are transmitted 1 once per second.

      /* Byte 0 - Status */
    unsigned char pulse78:2;  // Bits 7 and 8 of heart rate
    unsigned char artf:1;  // Artifact: A detected pulse beat didn't match the current pulse
    interval
    unsigned char mprf:1;  // Marginal Perfusion: Amplitude representation of medium signal
    quality (holds for entire duration)
    unsigned char lprf:1;  // Low Perfusion: Amplitude representation of low signal quality
    (holds for entire duration)
    unsigned char oot:1;   // Out Of Track: An absence of consecutive good pulse signals
    unsigned char snsd:1;  // Sensor Disconnect: Sensor is not connected to oximeter or
    sensor is inoperable
    unsigned char status_msb:1; // Bit 7 is always set

      /* Byte 1 - Heart Rate - Data range 18-321 beats per minute (BPM) */
      // Standard 4-beat average values not including display holds
    unsigned char pulse:7;
    unsigned char pulse_msb:1;  // Bit 7 is always clear

      /* Byte 2 - SpO2 - Data range 0-100 (%) */
      // Standard 4-beat average values not including display holds
    unsigned char sp02:7;
    unsigned char spO2_msb:1;   // Bit 7 is always clear

      // NOTE: When SpO2 and HR cannot be computed, the system will send a missing data
      indicator.
      //       For missing data, the HR equals 511 and the SpO2 equals 127.
      //       O lo que es lo mismo, todo unos para spO2, pulse y pulse78.

} __attribute__ ((packed));
typedef struct _nonin_frame_t nonin_frame_t;

#endif
```

```
//PulseoxP.nc
#include "Nonin.h"
#include "Pulseox.h"

module PulseoxP
{
    provides
    {
        interface Read<uint16_t> as ReadPulse;
        interface Read<uint16_t> as ReadOxygen;
        interface SplitControl;
    }
    uses
    {
        interface Nonin;
        interface SplitControl as NoninControl;
    }
}
implementation
{
      // pulse ox data placeholders
    uint16_t sp02;
    uint16_t pulse;
    bool running = FALSE;

    task void readPulseTask()
        {
            uint16_t retPulse;
            atomic
            {
                retPulse = pulse;
            }
            signal ReadPulse.readDone(SUCCESS, retPulse);
        }

    task void readOxygenTask()
        {
            uint16_t retOxygen;
            atomic
            {
                retOxygen = sp02;
            }
            signal ReadOxygen.readDone(SUCCESS, retOxygen);
        }

    command error_t SplitControl.start()
        {
            if(running == FALSE)
                return call NoninControl.start();
            else
                return SUCCESS;
        }

    command error_t SplitControl.stop()
        {
            if(running == TRUE)
                return call NoninControl.stop();
            else
                return SUCCESS;
```

```
        }

    event void NoninControl.startDone(error_t result)
        {
            running = TRUE;
            signal SplitControl.startDone(result);
        }

    event void NoninControl.stopDone(error_t result)
        {
            running = FALSE;
            signal SplitControl.stopDone(result);
        }

    command error_t ReadPulse.read()
        {
            post readPulseTask();
            return SUCCESS;
        }

    command error_t ReadOxygen.read()
        {
            post readOxygenTask();
            return SUCCESS;
        }

    async event void Nonin.frameReceived(nonin_frame_t *frame)
        {
              // save oxygen data
              // save pulse data
            atomic
            {
                sp02 = frame->sp02;
                pulse = ((uint16_t)frame->pulse78 << 7) | frame->pulse;
            }
        }
}



//Pulseox.nc
includes Pulseox;

interface Pulseox {

    command void start();
    command void stop();
    async event void dataReceived(pulseox_data_t *data);

}



//Pulseox.h

#ifndef PULSEOX_H
#define PULSEOX_H

// enum
// {
//   PULSEOX_TYPE_SP02 = 0x00,      /\* Data range 0-100 (%) *\/
//   PULSEOX_TYPE_PULSE = 0x01,           /\* Data range 18-321 beats per minute (BPM)*\/
//   PULSEOX_TYPE_SENSORSTATUS = 0x04,     /\* See flags description *\/
// };

#endif
```

```
//PulseC.nc
#include "Pulseox.h"

generic configuration PulseC()
{
    provides interface Read<uint16_t>;
    provides interface SplitControl;
}
implementation
{
    components PulseoxP;
    components NoninC;

    Read = PulseoxP.ReadPulse;
    SplitControl = PulseoxP;
    PulseoxP.NoninControl -> NoninC.SplitControl;
    PulseoxP.Nonin -> NoninC.Nonin;
}
```

```
//OxygenC.nc
#include "Pulseox.h"

generic configuration OxygenC()
{
    provides interface Read<uint16_t>;
    provides interface SplitControl;
}
implementation
{
    components PulseoxP;
    components NoninC;

    Read = PulseoxP.ReadOxygen;
    SplitControl = PulseoxP;
    PulseoxP.NoninControl -> NoninC.SplitControl;
    PulseoxP.Nonin -> NoninC.Nonin;
}
```

```
//NoninP.nc
#include <Atm128Uart.h>
#include "Nonin.h"
#include "Pulseox.h"

module NoninP
{
  provides
  {
    interface Nonin;
    interface SplitControl;
  }
  uses
  {
    interface UartStream;
    interface StdControl as UartControl;
/*     interface GeneralIO as PW0; */
    interface Atm128Calibrate;
    interface Timer<TMilli> as InitTimer;
    interface Leds;
  }
}
implementation
{
  uint16_t currOffset = 0; // current byte offset for incoming UART data
  uint8_t frameData[NONIN_FRAME_SIZE]; // place holder for incoming UART pulseox data
  bool running = FALSE;

  static void initSerial();

  task void stopTask()
  {
    atomic
    {
      running = FALSE;
    }
    call InitTimer.stop();
/*     call PW0.clr(); */
    call UartControl.start();
    call UartControl.stop();

    call Leds.led0Off();

    signal SplitControl.stopDone(SUCCESS);
  }

  command error_t SplitControl.start()
  {
    if(running == FALSE)
    {
      atomic
      {
        currOffset = 0;
      }
      initSerial();
        // cada segundo cambia el estado del led 0
      call InitTimer.startPeriodic(1000);
      call Leds.led0On();
```

```
/*      call PW0.set(); */
    }
    atomic
    {
      running = TRUE;
    }
    // done with startup, we can now get data
    signal SplitControl.startDone(SUCCESS);
    return SUCCESS;
  }

  static void initSerial()
  {
    // these data structures are defined in tos/chips/atm128/Atm128Uart.h
    Atm128UartMode_t mode;
    Atm128UartStatus_t stts;
    Atm128UartControl_t ctrl;
    uint16_t ubrr0;

    // initialize the UART and then set the registers to our specs
    call UartControl.start();
    call UartControl.stop();

    // perform ops similar to what is found in tos/chips/atm128/HplAtm128UartP.nc
    // enable tx/rx interrupts and tx/rx
    ctrl.bits = (struct Atm128_UCSRB_t) {rxcie:1, txcie:1, rxen:1, txen:1};
    // double rate
    stts.bits = (struct Atm128_UCSRA_t) {u2x:1};
    // 8-bit no parity, 1 stop bit
    mode.bits = (struct Atm128_UCSRC_t) {ucsz:ATM128_UART_DATA_SIZE_8_BITS};

    // set to 9600 baud, double rate
    ubrr0 = call Atm128Calibrate.baudrateRegister(9600L);
    // write to registers
    UBRR0L = ubrr0;
    UBRR0H = ubrr0 >> 8;
    UCSR0A = stts.flat;
    UCSR0C = mode.flat;
    UCSR0B = ctrl.flat;
  }

  event void InitTimer.fired()
  {
        // cada segundo cambia el estado del led 0
      call Leds.led0Toggle();
  }

  command error_t SplitControl.stop()
  {
    post stopTask();
    return SUCCESS;
  }

  async event void UartStream.receivedByte(uint8_t byte)
  {
    // add each byte to the existing frame until reception is complete
    if(!running)
      return;
    atomic
```

```
    {
      if (byte & 0x80)
      {
          currOffset = 0;
      }
      if (currOffset < NONIN_FRAME_SIZE)
      {
          frameData[currOffset] = byte;
          currOffset++;
          if(currOffset == NONIN_FRAME_SIZE)
        {
            signal Nonin.frameReceived((bci_frame_t *)frameData);
          }
      }
    }
  }

  async event void UartStream.receiveDone( uint8_t* buf, uint16_t len, error_t error) { }

  async event void UartStream.sendDone( uint8_t* buf, uint16_t len, error_t error) { }
}


//NoninC.nc
// requires atm128/1281 hardware

configuration NoninC
{
    provides
    {
        interface Nonin;
        interface SplitControl;
    }
}
implementation
{
    components NoninP;

    components Atm128Uart0C as UartC;
/*      components MicaBusC; */
    components PlatformC;

    components new TimerMilliC() as InitTimer;
    components LedsC;



    Nonin = NoninP;
    SplitControl = NoninP;

    NoninP.UartStream -> UartC.UartStream;
    NoninP.UartControl -> UartC.StdControl;
/*      NoninP.PW0 -> MicaBusC.PW0; */
    NoninP.Atm128Calibrate -> PlatformC;

    NoninP.InitTimer -> InitTimer;
    NoninP.Leds -> LedsC;
}



//Nonin.nc
#include "Nonin.h"

interface Nonin
{
  async event void frameReceived(nonin_frame_t *frame);
}
```

8.3 Computo de las matrices de transición

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%
    %% FICHEROS PARA MATLAB %%
    %%%%%%%%%%%%%%%%%%%%%%%%%%

% Años del usuario en el momento de la prueba
age=27;
% Age-predictal maximal HR revisited
pmax=208-0.7*age;
% Carga los datos (pulsaciones) y numero de muestras en cada intervalo (etapas) de las
sesiones
% dificultad = Transiciones entre etapas:
% EE = 1 (Easy to Easy)
% EH = 2 (Easy to Hard)
% HE = 3 (Hard to Easy)
% HH = 4 (Hard to Hard)
% temperatura = Temperatura de las sesiones:
% L = 0 (Low)
% H = 1 (High)
load('muestras');
ps={p1 p2 p3 p4 p5 p6 p7 p8 p9 p10 p11 p12};
es=[e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12];
% Suma acumulativa
cumes=cumsum(es);
save('datos','pmax','ps','es','cumes', 'dificultad', 'temperatura');

    %%%%%%%%%%%%%%%%%%%
    % procesados.mat %
    %%%%%%%%%%%%%%%%%%%

clear all;
load('datos');
% Etiquetar las etapas
% LOW=1 (adyacente inferior)
% MOD=2 (50% de pmax)
% WEI=3 (60% de pmax)
% AER=4 (70% de pmax)
% ANA=5 (80% de pmax)
% MAX=6 (90% de pmax)
% SUP=7 (adyacente superior)
% Para los 5 rangos de entrenamiento
%rangos=[pmax*[0.5:0.1:0.9] inf];
% Para contar con los rangos adyacentes a los 5 rangos de entrenamiento
rangos=[0 pmax*[0.5:0.1:1.0] inf];
ifemaxs=[];
for s=1:1:12,
  ife_tmp=histc(ps{s}(1:cumes(1,s),2), rangos);
  ife_tmp(end)=[];
  [femax ifemax]=max(ife_tmp);
  ife_tmp=ifemax;
  for e=1:1:size(cumes(:,s))-1,
    fe=histc(ps{s}(cumes(e,s)+1:cumes(e+1,s),2), rangos);
    fe(end)=[];
    % Se busca el rango de frecuencia mayor (femax)
    [femax ifemax]=max(fe);
    % Si hay mas de un máximo se queda con el primero (con el rango mas bajo)
    ife_tmp=[ife_tmp; ifemax];
  end
  ifemaxs=[ifemaxs ife_tmp];
end
```

```
save('procesados', 'ifemaxs', 'rangos', 'dificultad', 'temperatura');

    %%%%%%%%%%%%%%%%
    % matrices.mat %
    %%%%%%%%%%%%%%%%

load('procesados');
% Número total de transiciones para cada matriz
N1 = zeros(8,1);
N2 = zeros(4,1);
N3 = zeros(4,1);
N4 = zeros(2,1);
N1label = ['PEELT'; 'PEHLT'; 'PHELT'; 'PHHLT'; 'PEEHT'; 'PEHHT'; 'PHEHT'; 'PHHHT';];
N2label = ['PEExx'; 'PEHxx'; 'PHExx'; 'PHHxx'];
N3label = ['PxELT'; 'PxHLT'; 'PxEHT'; 'PxHHT'];
N4label = ['PxExx'; 'PxHxx'];
% Inicialización de Matrices
PEELT = zeros(7, 7); PEHLT = zeros(7, 7); PHELT = zeros(7, 7); PHHLT = zeros(7, 7);
PEEHT = zeros(7, 7); PEHHT = zeros(7, 7); PHEHT = zeros(7, 7); PHHHT = zeros(7, 7);
PEExx = zeros(7, 7); PEHxx = zeros(7, 7); PHExx = zeros(7, 7); PHHxx = zeros(7, 7);
PxELT = zeros(7, 7); PxHLT = zeros(7, 7); PxEHT = zeros(7, 7); PxHHT = zeros(7, 7);
PxExx = zeros(7, 7); PxHxx = zeros(7, 7);

for s=1:12,
    for e=1:12,
        a = ifemaxs(e, s);
        b = ifemaxs(e+1, s);
        switch temperatura(s)
        case 0
            switch dificultad(e)
            case 1
                PEELT(a, b) = PEELT(a, b) + 1;   N1(1) = N1(1) + 1;
                PEExx(a, b) = PEExx(a, b) + 1;   N2(1) = N2(1) + 1;
                PxELT(a, b) = PxELT(a, b) + 1;   N3(1) = N3(1) + 1;
                PxExx(a, b) = PxExx(a, b) + 1;   N4(1) = N4(1) + 1;
            case 2
                PEHLT(a, b) = PEHLT(a, b) + 1;   N1(2) = N1(2) + 1;
                PEHxx(a, b) = PEHxx(a, b) + 1;   N2(2) = N2(2) + 1;
                PxHLT(a, b) = PxHLT(a, b) + 1;   N3(2) = N3(2) + 1;
                PxHxx(a, b) = PxHxx(a, b) + 1;   N4(2) = N4(2) + 1;
            case 3
                PHELT(a, b) = PHELT(a, b) + 1;   N1(3) = N1(3) + 1;
                PHExx(a, b) = PHExx(a, b) + 1;   N2(3) = N2(3) + 1;
                PxELT(a, b) = PxELT(a, b) + 1;   N3(1) = N3(1) + 1;
                PxExx(a, b) = PxExx(a, b) + 1;   N4(1) = N4(1) + 1;
            case 4
                PHHLT(a, b) = PHHLT(a, b) + 1;   N1(4) = N1(4) + 1;
                PHHxx(a, b) = PHHxx(a, b) + 1;   N2(4) = N2(4) + 1;
                PxHLT(a, b) = PxHLT(a, b) + 1;   N3(2) = N3(2) + 1;
                PxHxx(a, b) = PxHxx(a, b) + 1;   N4(2) = N4(2) + 1;
            end
        case 1
            switch dificultad(e)
            case 1
                PEEHT(a, b) = PEEHT(a, b) + 1;   N1(5) = N1(5) + 1;
                PEExx(a, b) = PEExx(a, b) + 1;   N2(1) = N2(1) + 1;
                PxEHT(a, b) = PxEHT(a, b) + 1;   N3(3) = N3(3) + 1;
                PxExx(a, b) = PxExx(a, b) + 1;   N4(1) = N4(1) + 1;
            case 2
```

```
                PEHHT(a, b) = PEHHT(a, b) + 1;   N1(6) = N1(6) + 1;
                PEHxx(a, b) = PEHxx(a, b) + 1;   N2(2) = N2(2) + 1;
                PxHHT(a, b) = PxHHT(a, b) + 1;   N3(4) = N3(4) + 1;
                PxHxx(a, b) = PxHxx(a, b) + 1;   N4(2) = N4(2) + 1;
            case 3
                PHEHT(a, b) = PHEHT(a, b) + 1;   N1(7) = N1(7) + 1;
                PHExx(a, b) = PHExx(a, b) + 1;   N2(3) = N2(3) + 1;
                PxEHT(a, b) = PxEHT(a, b) + 1;   N3(3) = N3(3) + 1;
                PxExx(a, b) = PxExx(a, b) + 1;   N4(1) = N4(1) + 1;
            case 4
                PHHHT(a, b) = PHHHT(a, b) + 1;   N1(8) = N1(8) + 1;
                PHHxx(a, b) = PHHxx(a, b) + 1;   N2(4) = N2(4) + 1;
                PxHHT(a, b) = PxHHT(a, b) + 1;   N3(4) = N3(4) + 1;
                PxHxx(a, b) = PxHxx(a, b) + 1;   N4(2) = N4(2) + 1;
            end
        end
    end
end
if 1==1
    for x=1:7,
        PEELT(x,:) = PEELT(x,:) / (sum(PEELT(x,:)) + (sum(PEELT(x,:))==0));
        PEHLT(x,:) = PEHLT(x,:) / (sum(PEHLT(x,:)) + (sum(PEHLT(x,:))==0));
        PHELT(x,:) = PHELT(x,:) / (sum(PHELT(x,:)) + (sum(PHELT(x,:))==0));
        PHHLT(x,:) = PHHLT(x,:) / (sum(PHHLT(x,:)) + (sum(PHHLT(x,:))==0));
        PEEHT(x,:) = PEEHT(x,:) / (sum(PEEHT(x,:)) + (sum(PEEHT(x,:))==0));
        PEHHT(x,:) = PEHHT(x,:) / (sum(PEHHT(x,:)) + (sum(PEHHT(x,:))==0));
        PHEHT(x,:) = PHEHT(x,:) / (sum(PHEHT(x,:)) + (sum(PHEHT(x,:))==0));
        PHHHT(x,:) = PHHHT(x,:) / (sum(PHHHT(x,:)) + (sum(PHHHT(x,:))==0));

        PEExx(x,:) = PEExx(x,:) / (sum(PEExx(x,:)) + (sum(PEExx(x,:))==0));
        PEHxx(x,:) = PEHxx(x,:) / (sum(PEHxx(x,:)) + (sum(PEHxx(x,:))==0));
        PHExx(x,:) = PHExx(x,:) / (sum(PHExx(x,:)) + (sum(PHExx(x,:))==0));
        PHHxx(x,:) = PHHxx(x,:) / (sum(PHHxx(x,:)) + (sum(PHHxx(x,:))==0));

        PxELT(x,:) = PxELT(x,:) / (sum(PxELT(x,:)) + (sum(PxELT(x,:))==0));
        PxHLT(x,:) = PxHLT(x,:) / (sum(PxHLT(x,:)) + (sum(PxHLT(x,:))==0));
        PxEHT(x,:) = PxEHT(x,:) / (sum(PxEHT(x,:)) + (sum(PxEHT(x,:))==0));
        PxHHT(x,:) = PxHHT(x,:) / (sum(PxHHT(x,:)) + (sum(PxHHT(x,:))==0));

        PxExx(x,:) = PxExx(x,:) / (sum(PxExx(x,:)) + (sum(PxExx(x,:))==0));
        PxHxx(x,:) = PxHxx(x,:) / (sum(PxHxx(x,:)) + (sum(PxHxx(x,:))==0));
    end
    save('matrices', 'PEELT', 'PEHLT', 'PHELT', 'PHHLT', 'PEEHT', 'PEHHT', 'PHEHT', 'PHHHT');
    save('matrices', 'PEExx', 'PEHxx', 'PHExx', 'PHHxx', '-append');
    save('matrices', 'PxELT', 'PxHLT', 'PxEHT', 'PxHHT', '-append');
    save('matrices', 'PxExx', 'PxHxx', '-append');
else
    PEELT = PEELT / N1(1);
    PEHLT = PEHLT / N1(2);
    PHELT = PHELT / N1(3);
    PHHLT = PHHLT / N1(4);
    PEEHT = PEEHT / N1(5);
    PEHHT = PEHHT / N1(6);
    PHEHT = PHEHT / N1(7);
    PHHHT = PHHHT / N1(8);

    PEExx = PEExx / N2(1);
    PEHxx = PEHxx / N2(2);
    PHExx = PHExx / N2(3);
```

```
    PHHxx = PHHxx / N2(4);

    PxELT = PxELT / N3(1);
    PxHLT = PxHLT / N3(2);
    PxEHT = PxEHT / N3(3);
    PxHHT = PxHHT / N3(4);

    PxExx = PxExx / N4(1);
    PxHxx = PxHxx / N4(2);

    save('matrices', 'PEELT', 'PEHLT', 'PHELT', 'PHHLT', 'PEEHT', 'PEHHT', 'PHEHT', 'PHHHT',
    'N1', 'N1label');
    save('matrices', 'PEExx', 'PEHxx', 'PHExx', 'PHHxx', 'N2', 'N2label', '-append');
    save('matrices', 'PxELT', 'PxHLT', 'PxEHT', 'PxHHT', 'N3', 'N3label', '-append');
    save('matrices', 'PxExx', 'PxHxx', 'N4', 'N4label', '-append');
end
```