Universidad de la República

Facultad de Ingeniería

Proyecto de fin de carrera

MOSOBO

Monitoreo del Sonido Bovino

Valeria Olivera, Santiago Reyes, Cecilia San Román

Tutor: Ing. Juan Pablo Oliver

Montevideo, Uruguay

Junio 2010

Resumen

En este trabajo se presenta el prototipo Monitoreo del Sonido Bovino (MOSOBO), el cual consiste en un dispositivo capaz de grabar y almacenar el sonido que emiten los bovinos durante la masticación e ingesta de forraje, junto con un software capaz de detectar las actividades realizadas por el animal (pastoreo, rumia y descanso) a partir de las señales de audio obtenidas. Con estos datos es posible determinar en qué momento del día y durante cuánto tiempo el animal realizó alguna actividad, lo que permite estudiar el comportamiento ingestivo de los rumiantes.

El prototipo diseñado tiene las características de autonomía energética de más de 24 horas, grabar y almacenar audio de alta calidad, no afectar el comportamiento del animal, ser de bajo costo y robusto. El dispositivo se coloca fácilmente, junto con un bozal, y se ubica sobre la tabla del bovino.

En lo que respecta al hardware se logró desarrollar un dispositivo que presenta características de estabilidad y robustez. También se obtuvieron muy buenos resultados en el reconocimiento de las actividades de pastoreo y rumia (porcentajes de acierto del 100 %), mientras que el resultado del reconocimiento de la actividad de descanso no fue tan satisfactorio (porcentaje de acierto del 52 %). Pequeñas modificaciones del dispositivo, como por ejemplo cambiar la ubicación del micrófono, mejorarían significativamente los resultados obtenidos.

Agradecimientos

En primer lugar queremos agradecer a Martín do Carmo y Juan Pablo Oliver por habernos dado la oportunidad de realizar este proyecto y por su buena disposición, voluntad y entusiasmo.

Queremos agredecer especialmente a Pablo Cancela, Ernesto López y Martín Rocamora por sus buenos consejos; y a Sebastián Fernández, Juan Manuel Rincón, Raúl Rincón y Milthon Reyes por los inconvenientes que nos hicieron evitar.

A Carlos Aiello por sus constantes aportes a lo largo de todo el proyecto, a Andrés Aguirre por su colaboración en el desarrollo del sistema embebido, a Carlos Suárez por su participación en la construcción del bozal y a Dorrel Bentancor por ayudarnos, apoyarnos y acompañarnos en Cerro Largo.

Por supuesto también queremos agradecer a nuestras familias y amigos por el constante apoyo desde el comienzo del proyecto y sin los cuales no hubiera sido posible la realización del mismo.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción

Capítulo 1

Descripción del proyecto

1.1. Descripción

El objetivo del proyecto es desarrollar a nivel de prototipo un dispositivo capaz de grabar el sonido que realizan los bovinos durante la masticación e ingesta de forraje, junto con un software capaz de determinar y desplegar qué actividades realizó el animal (pastoreo, rumia o descanso), en qué orden y la duración de las mismas.

Además de las características mencionadas el prototipo debe ser de fácil colocación, bajo peso y costo reducido. Otra de las características que debe tener el prototipo es que permita grabar *ininterrumpidamente* a lo largo del día y sin que sea necesaria la presencia de humanos que puedan afectar el comportamiento ingestivo del animal.

1.2. Antecedentes

El comportamiento animal en pastoreo es resultado de factores abióticos (distancia al agua, la temperatura, la luz, el pH, el suelo, nutrientes, pendiente) y factores bióticos (cantidad y calidad del forraje por animal, estado fisiológico y metabólico del animal)[1]. Justamente la cantidad de forraje consumida por el animal depende del *tiempo de pastoreo* y su tasa de consumo[2].

Existen situaciones en las que el consumo de energía del animal no se modifi-

ca pero su productividad se ve modificada como consecuencia de tener diferente comportamiento ingestivo (ej. mayor traslado en busca de alimento debido a las características de distribución del alimento y/o estructura de la pastura). La medición del comportamiento y del consumo de energía contribuiría a explicar gran parte de los resultados en productividad animal a nivel experimental y comercial.

La Facultad de Agronomía de la Regional Norte de Paysandú está realizando actividades de investigación con bovinos y ovinos en la Unidad de Extensión de Cerro Largo. Algunas de ellas se basan en obtener el tiempo que dichos herbívoros se encuentran pastoreando, rumiando o descansando. Para medir estos tiempos utilizan el dispositivo *IGER Behaviour Analysis System*[1] el cual presenta problemas de colocación, interpretación de los datos adquiridos, autonomía energética y elevado costo. Su principio de funcionamiento consiste en captar los movimientos de la mandíbula, registrarlos y finalmente interpretarlos con el software comercial.

El proyecto se basa en las conclusiones de algunas investigaciones sobre el comportamiento de los rumiantes y los sonidos que los mismos emiten en las distintas etapas de la digestión[3].

Los clientes son Ing. Agr. Martín Do Carmo e Ing. Agr. Pablo Soca, de la Facultad de Agronomía de la Universidad de la República. Destinan USD 1000 para este proyecto.

[1] http://www.ultrasoundadvice.co.uk/pages/IGERhome.html

Capítulo 2

Objetivo general del proyecto

2.1. ¿Qué?

Este proyecto consiste en desarrollar a nivel de prototipo, un dispositivo capaz de grabar el sonido que realizan los bovinos durante la masticación e ingesta de forraje, junto con un paquete de software para PC que sea capaz de identificar las actividades que realiza el animal (pastoreo, rumia y descanso) y desplegar los resultados. Debe tener una interfaz con el usuario que se determinará posteriormente.

Los requisitos principales del dispositivo son que el sistema sea robusto (ya que se encontrará a la intemperie junto con el animal), que tenga autonomía energética por más de 24 horas, sea económico, de fácil colocación y bajo peso (menor a 1 kg), y que sea inofensivo y no modifique el comportamiento de los animales.

Con respecto al software el requisito principal es que sea capaz de identificar 3 actividades bien diferenciadas (pastoreo, rumia y descanso), en qué momento son realizadas y la duración de las mismas.

2.2. ¿Para qué?

Para poder determinar el tiempo que los rumiantes destinan a pastar, rumiar y descansar; y en qué momento del día lo realizan. Con estos datos los investi-

gadores pueden determinar por un lado cómo los factores bióticos[1] afectan al comportamiento ingestivo del animal[4] y por otro cómo afecta la altura de la pastura en el comportamiento ingestivo de vacas de cría en diferentes estados fisiológicos[5].

El proyecto se enfocará en distinguir adecuadamente cuánto tiempo los bovinos han dedicado a las acciones mencionadas anteriormente.

Los mismos análisis también pueden realizarse con ovinos; ya que tanto los bovinos como los ovinos son rumiantes, por lo que el proceso de digestión es el mismo. La única diferencia radica en los resultados obtenidos, como por ejemplo la cantidad de movimientos de mandíbula, mordidas y masticación[6].

2.3. Criterios de éxito

Los principales criterios de éxito son que el prototipo grabe y almacene correctamente los sonidos emitidos por los animales con una independencia energética de 24 horas y que el software identifique las actividades de pastoreo, rumia y descanso, durante cuánto tiempo y en qué momento del día son realizadas.

Otros criterios de éxito son que el dispositivo pueda permanecer a la intemperie junto con el animal sin presentar problemas de funcionamiento, sea de fácil colocación, no interfiera con el comportamiento habitual del animal y no genere lesiones al mismo. Además los sonidos del ambiente, como por ejemplo otros animales, factores meteorológicos y ondas electromagnéticas, no deben interferir con el resultado del análisis.

También debe cumplirse que el costo del dispositivo sea menor a 1000 USD y que su peso sea menor a 1 kg.

[1] Cantidad de forraje ofrecido por animal y/o biotipo animal

Parte II

Diseño

Capítulo 3

Ensayos preliminares

3.1. Introducción

Para poder determinar los requerimientos de frecuencia de muestreo y cantidad de bits del dispositivo de grabación es necesario conocer qué características tienen los sonidos emitidos por los bovinos. De la buena elección de estos requerimientos depende el éxito del software de análisis y reconocimiento.

Por esta razón nos trasladamos hasta la Estación Experimental "Dr. Mario A Cassinoni" (EEMAC), Facultad de Agronomía, Paysandú para realizar varios ensayos que nos permitieran determinar dichos requerimientos.

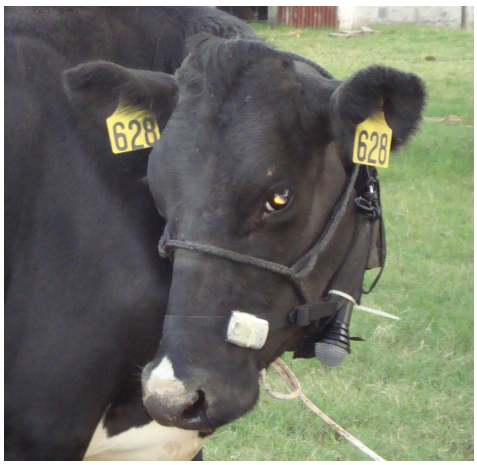
3.2. Materiales

Para realizar la adquisición de datos utilizamos: reproductores de mp4, radiograbadores de cassette, micrófono Philips omnidireccional (frecuencia de corte 10 kHz), micrófono Sony stereo solapero (frecuencia de corte 8 kHz), micrófono de PC stereo (frecuencia de corte 16 kHz) y PC (frecuencia de muestreo 48 kHz).

3.3. Observaciones realizadas durante los ensayos

Al principio los animales estaban muy inquietos e intimidados por nuestra presencia, lo que dificultó obtener datos relevantes (prácticamente no comieron ni

rumiaron). Los ensayos fueron realizados con reproductores de mp4 los cuales solo pueden grabar sonidos hasta 8 kHz con un conversor de 8 bits; y con un micrófono de PC con una PC que tienen la capacidad de muestrear a 48 kHz, 16 bits (Ver Figura 3.1).



(a)

(b)

(c)

Figura 3.1: Ensayos realizados para adquirir las señales a analizar

3.4. Procesamiento digital de las señales

Para analizar las señales obtenidas a partir de los ensayos mencionados anteriormente, se consultó al grupo de procesamiento de señales de la Facultad de

Ingeniería. Nos recomendaron el software *Wavesurfer* y con el mismo se observaron y obtuvieron los segmentos de las señales que aportaron información útil.

En la Figura 3.2 y en la Figura 3.3 se pueden observar algunas secuencias representativas de las actividades de rumia y pastoreo respectivamente.

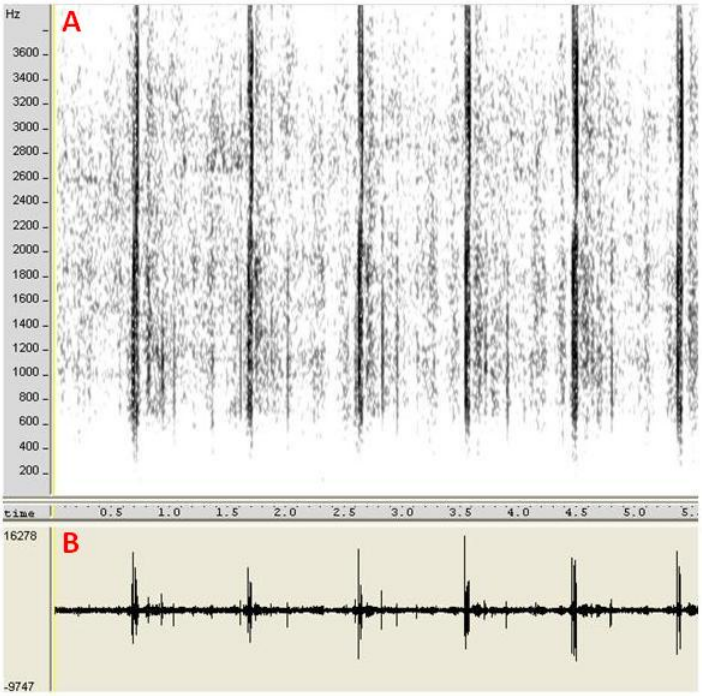


Figura 3.2: **Rumia** En A se puede ver el espectrograma mientras que en B la señal en función del tiempo.

La Figura 3.2 no brinda información sobre la frecuencia mínima de muestreo, pues se obtuvo con el reproductor mp3 el cual muestrea a 8kHz. Sin embargo, se puede notar fácilmente que la señal presenta un patrón rítmico.

En la Figura 3.3 se pueden identificar componentes de señal de interés de hasta 6 kHz. Esta señal a diferencia de la anterior, fue obtenida con el micrófono de PC y la PC.

El artículo *"Computational method for segmentation and classification of ingestive sounds in sheep"*[7] propone un método novedoso para análizar y reconocer automaticamente señales sonoras emitidas por ovinos al realizar pastoreo y rumia[1]. En el mismo se describen, por ejemplo, los materiales utilizados: videocá-

[1]El método propuesto puede ser utilizado indistintamente en bovinos y ovinos ya que desde el punto de vista del comportamiento ingestivo son considerados iguales.

mara y micrófono inalámbrico. Luego las señales obtenidas fueron digitalizadas y analizadas utilizando el sofware *Cool Edit Pro v2*, a una frecuencia de muestreo de 44,1kHz y una resolución de 16 bits.

En base a este artículo y en las características presentes en las señales obtenidas durante los ensayos se decidió que el dispositivo de grabación debía muestrear al menos a 22 kHz y tener una resolución de al menos 12 bits.

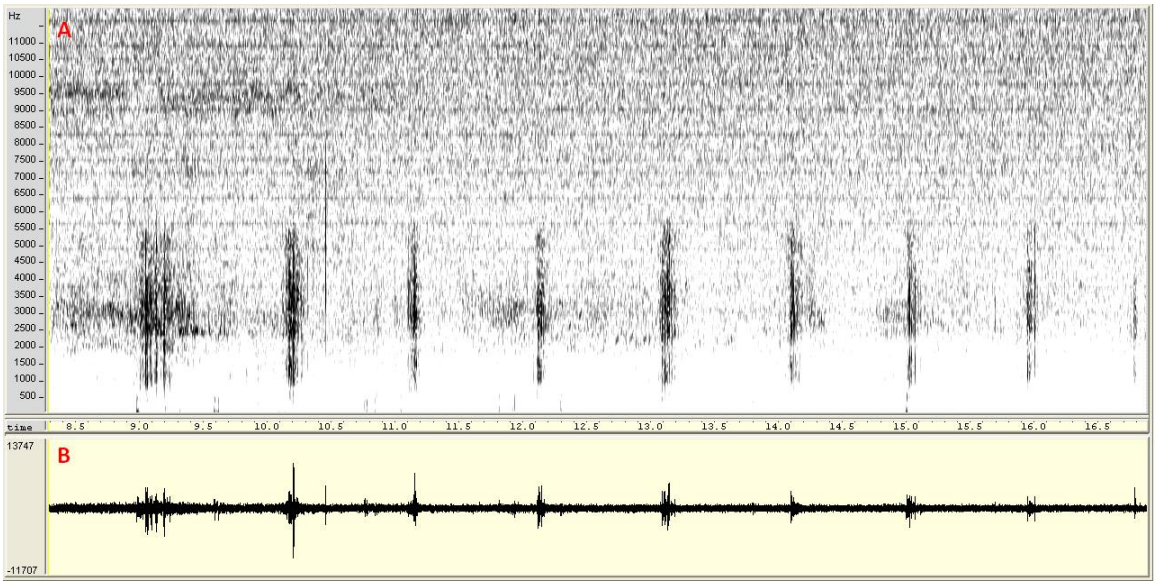


Figura 3.3: **Pastoreo**. En A se puede ver el espectrograma mientras que en B la señal en función del tiempo.

Capítulo 4

Hardware

4.1. Requerimientos del hardware

Para determinar los requerimientos del hardware nos basamos en los ensayos preliminares descriptos previamente y en los articulos [8] y [9]. Dichos artículos obtienen las señales a 44,1kHz, 16 bits en PCM, y luego le aplican un decimador de orden 2 lo cual es equivalente a haber grabado los datos a 22,05 kHz. A partir de los ensayos preliminares realizados, consideramos que con 22.05 kHz, 12 bits en PCM es suficiente para obtener una señal de buena calidad.

Teniendo en cuenta que el dispositivo debería ser capaz de grabar el sonido emitido por los rumiantes durante 24 horas, al obtener las señales a 22.05 kHz, 12 bits es necesario una memoria no volátil de al menos 3,5GB.

El dispositivo debe permanecer en el animal durante 1 día sin interrupcción, por lo que la batería debe ser capaz de alimentar el dispositivo durante este tiempo. Además debe tener un factor de forma aceptable y que sea liviana. Teniendo en cuenta las baterías que se encuentran en plaza que cumplen los requerimientos antes mencionados concluimos que la placa debe consumir menos de 1,5 W.

Uno de los objetivos del proyecto es determinar en que momento del día el animal realizó cada actividad, por lo que es necesario un reloj de tiempo real.

Una de las características deseables del dispositivo es la capacidad de interactuar con el usuario y para esto se utilizarán leds y botones.

Del análisis de los distintos requerimientos del sistema se llego a la siguiente

especificación:

- Muestreo a 22kHz, al menos 12 bits.
- Memoria no volátil de 3,5 GB.
- Bajo consumo, menor a 1,5 W.
- Real Time Clock o posibilidad de conectar uno.
- Botones para la interfaz con el usuario (2 o 3).
- 2 Leds.

4.2. Opciones consideradas

La primera opción que se planteó fue la de desarrollar el circuito utilizando microcontroladores o procesadores, ADC o tarjetas de audio. Esta opción se descartó rápidamente debido al tiempo que insumiría el diseño e implementación de la placa.

Otra opción era la de basar nuestro desarrollo mediante el uso de grabadoras digitales de audio, reproductores de mp3, mp4 o celulares capaces de adquirir audio. Se descartó la utilización de grabadoras digitales de audio debido a su elevado costo y consumo. Para poder utilizar los reproductores de mp3 o mp4 se les debería instalar un nuevo firmware, con la finalidad de que pudiera ser modificado para muestrear a la frecuencia y cantidad de bits necesarios; pero de todas formas los dispositivos seguían sin cumplir con los requerimientos. El uso de celulares fue descartado debido a su elevado consumo.

La tercera opción que nos planteamos fue la de utilizar placas ya diseñadas capaces de grabar una memoria SD o que tuvieran interfaz USB, cuyo consumo fuera lo menor posible (ultra low power) y que incluyeran ADC o tarjeta de sonido capaz de muestrear de acuerdo a los requerimientos establecidos.

Nuestra búsqueda inicial se basó en esta tercera opción y encontramos que la mayoría de las placas que incluían conversores analógicos digitales (ADC) o tarjetas de sonido on-board o bien muestreaban a altas frecuencias pero con la cantidad insuficiente de bits, o a muy bajas frecuencias y 16 bits o más. De esta búsqueda

concluimos que deberíamos incluir una tarjeta de sonido o ADCs externos a la placa para poder cumplir con los requerimientos.

4.3. Criterios principales de elección

1. Cumplir con los requerimientos del hardware
2. Costo de la placa
3. Costo de los kits necesarios para realizar la programación de la misma (pago de licencias de software)
4. Si soporta algún sistema operativo o si existen las libreras necesarias para el desarrollo del sistema embebido.

Para seleccionar el hardware se consideraron 2 opciones: dessarrollar el sistema embebido completo o utilizar una placa capaz de soportar un sistema operativo ya diseñado.

La primera opción tiene como ventaja que al ser un sistema propio sólo se desarrollarán las aplicaciones necesarias para el sistema, reduciendo así el espacio en memoria y optimizando el rendimiento del mismo. Pero el tiempo de desarrollo y testeo de errores sería mayor que usando un sistema operativo ya desarrollado.

Mientras que la segunda opción agilizaría el desarrollo de aplicaciones a más alto nivel y nos facilitaría todas las comunicaciones con los puertos y transmisiones de datos aunque tiene implementado funciones que no serán necesarias para nuestra aplicación, ocupando mayor espacio en memoria y procesos innecesarios.

Por las razones mencionadas previamente es que se decidió buscar *Single Board Computers* (SBC) que soportaran alguno de los siguientes sistemas operativos: Windows ce, Windows embedded o GNU/Linux.

Las SBC nos facilitan el desarrollo de las aplicaciones debido a que el sistema operativo ya tiene implementado funciones esenciales para el desarrollo planteado y ha sido testeado desde hace años.

En la Tabla 4.1 se muestran algunas placas que tomamos en consideración, para basar el desarrollo de nuestro proyecto.

Sinlge board computers seleccionadas					
	Gumstix Overo Earth[10]	**Titan PC/104**[11]	**PPM-LX800**[12]	**EPX-GX500**[13]	**ARM TS-7260**[14]
Procesador	ARM Cortex-A8 CPU 600 MHz	520MHz PXA270 XScale	AMD 500 MHz Geode LX800	AMD GeodeTM GX500	200MHz ARM9 CPU
Memoria	256 MB RAM 256 MB Flash	Up to 128 MB SDRAM Up to 64 MB of AMD MirrorBit Flash	Up to 1028 MB SDRAM	Up to 512 MB SDRAM	32MB SDRAM 32MB NAND Flash
RTC		Si			Si
ADC	10 bits			12-bit	2 12-bit
SD	2GB micro SD	SD/SDIO/MMC card socket	Si	Si	Si
USB	USB HS Host	2x USB 1.1	2 x 2.0 ports	2 X Two USB 1.1 ports	2 USB 2.0(12 Mbit/s max)
Tamaño (mmxm-mxmm)	17x58x4.2	96x91	90x96		95x120
Consumo	1W	1.5W	0.9W	1.0W	0.25W a la frecuencia mínima
Sist. Operativos que soporta		Windows CE and Linux	Windows XP, XP Embedded, Linux, DOS, x86 RTOS	Windows XP, XP Embedded, Linux, Windows CE, DOS, x86 RTOS	Linux out-of-the-box

Cuadro 4.1: Comparación de posibles placas

4.4. Elección del Hardware

4.4.1. SBC, tarjeta de sonido y unidades de almacenamiento

Luego de evaluar las placas anteriormente mencionadas y comprobar si cumplían los requerimientos estipulados (con pequeños agregados), se llegó a la conclusión que la placa **TS-7260 de ARM** junto con la **tarjeta de sonido USB 3D Sound modelo HY554** son la mejor opción.

TS-7260

Esta placa tiene instalado de fábrica el núcleo 2.4.26 de GNU/Linux, y está optimizado para la arquitectura de la misma y su consumo máximo se encuentra

en el orden de 1 W (sin el uso de periféricos y a máxima frecuencia).

Por defecto la SBC trae instalada la aplicación BusyBox[15]; pero la utilización de la misma fue descartada debido a distintas limitaciones [1]. Se decidió instalar una distribución GNU/Linux Debian Sarge 3.1 debido a su compatibilidad con nuestra arquitectura y ser un sistema sumamente estable.

Para poder utilizar esta opción se debe instalar la distribución en una unidad externa de almacenamiento dado que la memoria interna de la SBC no es suficiente ya que la distribución ocupa alrededor de 512 MB.

USB 3D Sound modelo HY554

Inicialmente se había decidido grabar a 22 kHz pero la tarjeta de sonido seleccionada muestrea únicamente a 48 kHz. Para lograr este objetivo se puede modificar el oscilador de la tarjeta de sonido o decimar mediante software en la SBC. La primera opción fue descartada ya que en el caso de una eventual rotura dificulta su reemplazo. La implementación de la segunda opción implica decimar las señales de audio en la SBC, lo que lleva a consumir el 100 % del CPU y finalmente el audio queda corrupto.

Para solucionar este inconveniente se decidió grabar a 48 kHz (consumiendo el 2 % del CPU de la SBC) y decimar en una computadora.

Unidades de almcenamiento

Para poder almacenar 24 hs a 48 kHz, 16 bits, se necesita una unidad de almacenamiento de al menos 7 GB.

Si se desea almacenar la distribución del GNU/Linux y el audio generado durante un día es necesaria una memoria de 8GB. La primera opción que se consideró fue utilizar una tarjeta SDHC, pero no es posible implementar esta solución debido a que no es posible arrancar el sistema operativo desde esta tarjeta[2].

Finalmente se decidió almacenar los datos en un pendrive de 8GB e iniciar el sistema desde una memoria SD.

[1] Con la aplicación BusyBox no es posible manejar módulos de audio, interfaz de red y almacenamiento.

[2] La placa TS-7260 solo acepta las tarjetas SDHC como medio de almacenamiento no así como medio de arranque

4.4.2. Batería

Para determinar la capacidad de la batería a utilizar se relevó el consumo de la placa en las posibles situaciones de funcionamiento: inicio del sistema operativo, estado de espera[3] o grabando y almacenando. Los datos relevados se detallan en la Tabla 4.4.2.

Actividad	Consumo (mA)	Potencia (W)
Inicio del sistema	300 máx	1,5
Estado de espera	220	1,1
Grabando y almacenando	240	1,2

Cuadro 4.2: Consumo del sistema embebido al ser alimentado por una fuente de 5V.

Eligiendo el peor caso (inicio del sistema) el consumo es de 1,5 W, lo cual nos lleva a consumir 36 Wh.

Teniendo en cuenta la potencia que debe suministrar, el rango de alimentación de la SBC[4], factor de forma aceptable y que se encuentre en plaza.

Se decidió utilizar 2 baterías de gel (modelo RB632C) de 6V y 3200 mAh.

Usando ambas baterias se llega a una potencia de 38,4 W, cumpliendo los requerimientos planteados anteriormente.

Para determinar la forma más conveniente de conectar ambas baterías se relevó el consumo de la placa al conectar las baterias en serie y en paralelo.

A partir de los datos relevados (Ver Tabla 4.3) se llegó a la conclusión que en el estados de espera y el estado de grabar y almacenar el consumo es muy similar. En el único estado que el consumo difiere bastante es en el inicio del sistema, sin embargo el tiempo que dura la inicialización del sistema es depreciable con respecto al del resto de los estados.

Dado que al conectar las baterias en paralelo se produce una corriente parásita en la malla formada por las baterías en caso de no tener un desgaste parejo; es que se decidió conectarlas en **serie**.

[3]Sistema luego del arranque a la espera de una orden del usuario.
[4]La SBC debe ser alimentada entre 4,5 y 20 V.

	Inicio del sistema		Estado de espera		Grabando y almacenando	
Serie (12 V)	150 mA	1,8 W	90 mA	1,1 W	100 mA	1,2 W
Paralelo (6 V)	230 mA	1,4 W	190 mA	1,1 W	200 mA	1,2 W

Cuadro 4.3: Comparación de consumo de la SBC al conectar las baterias en serie y paralelo.

Teniendo en cuenta todas estas limitantes, finalmente se optó por utilizar dos baterías de gel (modelo RB632C) de 6V y 3200 mAh las cuales pesan 600 g y tienen un factor de forma aceptable: 32mmx96mmx60mm.

Al utilizar estas baterías no logramos cumplir con el requerimiento del peso pero se priorizó la autonomía energética del dispositivo de al menos 24 hs, que se consiga en el mercado local a bajo costo y su recarga se realice con un cargador accesible en plaza.

Las baterías seleccionadas se cargan con un adaptador AC-DC.

4.4.3. Recolección de datos

Para recuperar los datos del dispositivo se consideraron las siguientes opciones:

- Que el usuario removiera del circuito el pendrive y bajara los archivos adquiridos a su computadora.
- Enviar los datos mediante radio frecuencia.

Desde el punto de vista del desarrollo del proyecto el primer punto es el más sencillo de implementar, sin embargo el segundo punto se basa fundamentalmente en la posibilidad de que el usuario verifique si el dispositivo fue colocado correctamente, mediante la escucha prácticamente en tiempo real del audio que está siendo adquirido.

Teniendo en cuenta que se necesitan transmitir 8 GB junto con los encabezados de cada paquete y considerando que al implementarlo la mayor velocidad de transmisión alcanzada fue de 2Mbps, se necesitan 8 horas para transmitir la totalidad de los datos consumiendo aproximadamente 3 W. Por estas razones se decidió no utilizar esta tecnología aún cuando ya estaba implementada (Ver Anexo A.4).

Finalmente, para recolectar los datos almacenados en el pendrive el usuario debe retirar el mismo y conectarlo en un PC.

4.4.4. Micrófono

Para la elección del micrófono se tuvieron en cuenta las siguientes características:

- Sensibilidad
- Ancho de banda (al menos 11kHz)
- Voltaje de operación ($\pm$ 5V)
- Robustez
- Bajo costo

Se decidió utilizar el micrófono **Ht-101** de la marca Xtreme que se encuentra en el mercado local a un precio accesible y posee las siguiente características:

- Gama de frecuencia: 100-16000 Hz
- Impedancia: 2.2 $k\Omega$
- Sensibilidad: -55db $\pm$ 3db
- Omni direccional
- Cable reforzado en tela
- Conexión jack 3.5

Capítulo 5

Documentos y esquemáticos del hardware

En la figura 5.1, se puede observar a nivel de diagrama de bloques los componentes del sistema.

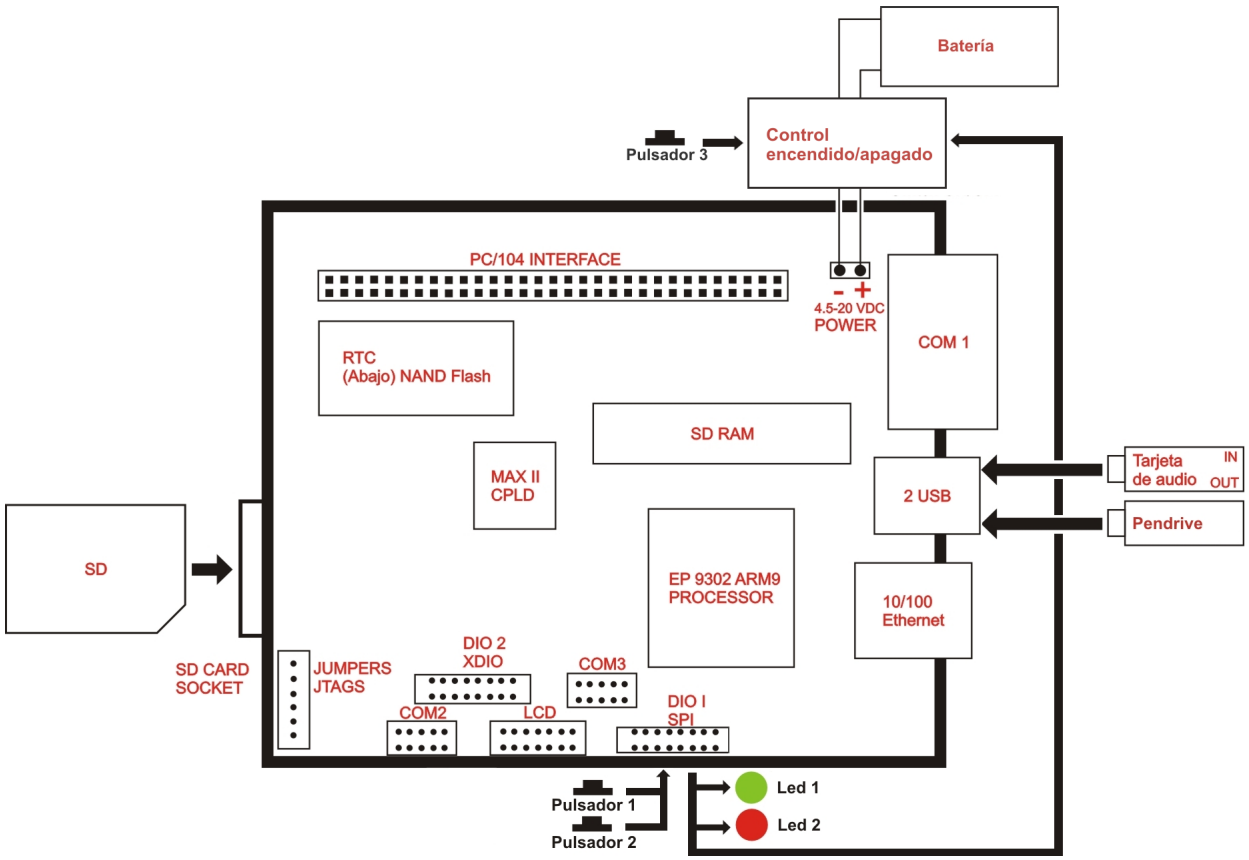


Figura 5.1: Diagrama de bloques del hardware.

5.1. Single Board Computer TS-7260

La placa TS-7260 desarrollada por Technologic Systems es una Single Board Computer (SBC) basada en el microprocesador de Cirrus EP9302 ARM9. Esta placa ademas incluye un software que controla la potencia que consumen los periféricos on-board. Características:

- Consume menos de 1 W a frecuencia máxima.
- 200 MHz ARM9 CPU con MMU
- 32 MB on-board NAND flash
- 32 MB SDRAM

- SD socket card
- 2 USB 2.0 Compatible OHCI ports (12 Mbit/s Max)
- 10/100 Ethernet port
- 30 total DIO pins
- Funciona entre -40º - 70 º
- Tamaño: 97mmx11mm
- RTC

5.1.1. Procesador

Cirrus EP9302

Opera desde 1.8 V, mientras que las I/O operan a 3.3 V y disipan entre 100 mW y 750 mW (depende de la velocidad). El CPU EP9302 tiene 16 KB de cache de instrucciones y 16 KB de cache para datos.

Interrupciones

El controlador de interrupciones de EP9302 permite hasta 54 interrupciones para generar un pedido de interrupcion (IRQ - Interruption Request) o un pedido de interrupción rápido (FIQ - Fast Interrupt Request).

Memoria

La TS-7260 usa 3 tipos de memoria. La SDRAM es la memoria volátil utilizada para correr aplicaciones del procesador y la memoria Flash on-board es la memoria no volátil para almacenamiento.

SD Memory Card

Technologic Systems cuenta con la licencia completa para utilizar las características adicionales de SD.

Real-Time Clock

La placa TS-7260 soporta un real time clock (RTC) con una batería de respaldo no volátil soldada en la placa. Utiliza un módulo ST Micro M48T86PC1 para la función de RTC. Este módulo contiene una batería de Litio, un cristal de 32,768 kHz y un chip de RTC con 114 bytes de RAM. Esto garantiza una operación continua del reloj de al menos 10 años en ausencia de energía.

5.2. Kit de desarrollo

El kit de desarrollo de TS-ARM incluye:

- Distribución Debian GNU/Linux 3.0 compilada para ARM.
- Compilador gcc 2.95.4 y gcc 3.0
- Sistema de administrador de paquetes de Debian: apt-get, tasksel, dselect.
- Toda la documentación.
- Un CD con el Kernel de Linux y código fuente de aplicaciones.
- Fuente de 5 VDC.
- Varios cables y conectores.

5.3. Placa de interfaz de usuario y control de energía

Para poder mantener una comunicación con el usuario se diseñó una placa interfaz de usuario y control de energía. La misma posee tres botones los cuales cumplen las funciones de comenzar a grabar señales de audio y prender y apagar el sistema. Al presionar los botones son activadas secuencias luminosas a través de los leds brindando la siguiente información al usuario:

Iniciando el sistema. El led verde se enciende y apaga cada 1 segundo.

Sistema iniciado. El led verde permanece encendido.

Comenzando grabación. El led verde tiene un funcionamiento cíclico con un período de 3 segundos; permanece encendido durante 1 segundo y apagado 2.

Tarjeta SD (contiene el SO) y/o la unidad de almacenamiento no están correctamente conectadas. El led rojo permanece encendido.

Apagando el sistema. El led rojo se enciende y apaga cada 1 segundo.

Para el manejo de los pulsadores y leds se utilizaron los pines del DIO1. Para proteger el sistema de posibles corrientes no deseadas y evitar que los pines queden en estado flotante al cambiar de estado, se colocaron resistencias de pull-up o pull-down según corresponda.

Uno de los problemas que se presentó al apagar la placa es que la misma sigue tomando corriente luego de que el sistema operativo es apagado. Para evitar este problema se estudiaron varías alternativas (Ver Anexo A.5) eligiendo el diseño menor consumo y menos costo.

En la figura 5.2 se puede observar el diseño de la placa interfaz de usuario y control de energía.

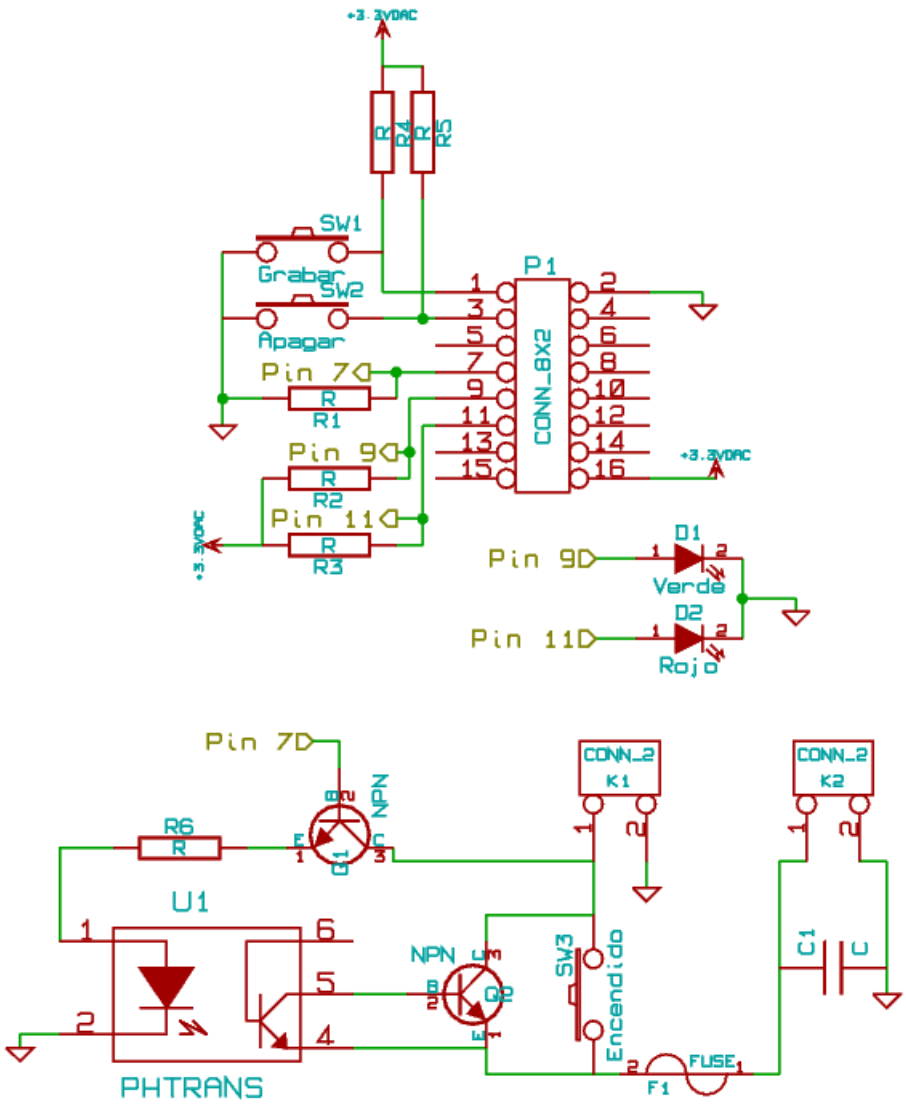


Figura 5.2: Esquema de la placa interfaz usuario y control de energía.

Como se puede observar, uno de los botones mecánicos se encuentra en paralelo a un botón lógico. El botón lógico se realizó mediante un opto acoplador y un transistor los cuales se disponen en una configuración Darlington.

Al presionar el botón de encendido la placa se prende y ejecuta su secuencia de arranque. Unos segundos después el sistema toma el control de los puertos de entrada-salida, se polariza el transistor Q1 (BC337) poniendo en "1" lógico (3.3V) el pin 7 del DIO, activando así al opto acoplador.

El transistor que realiza la salida del opto acoplador funciona como un interruptor ya que queda en estado de saturación. Esto a su vez produce la saturación del transistor Q2 (BC639) permitiendo así el pasaje de la corriente.

El proceso de encendido tiene una duración aproximada de 4 segundos y cuando finaliza enciende el led verde, indicando al usuario que el sistema está encendido y que puede soltar el pulsador.

El pin 7 permanecerá alto hasta que se mande apagar el sistema, la última acción que toma el sistema opertivo de la placa TS−7260 es bajar el pin 7 a "0". Esto hace que la SBC se quede sin alimentación, por lo que con esta solución la placa efectivamente no consumirá cuando no se desea.

Como última etapa de protección se colocó un fusible de 1 A para limitar la corriente que pude ser entregada a la SBC.

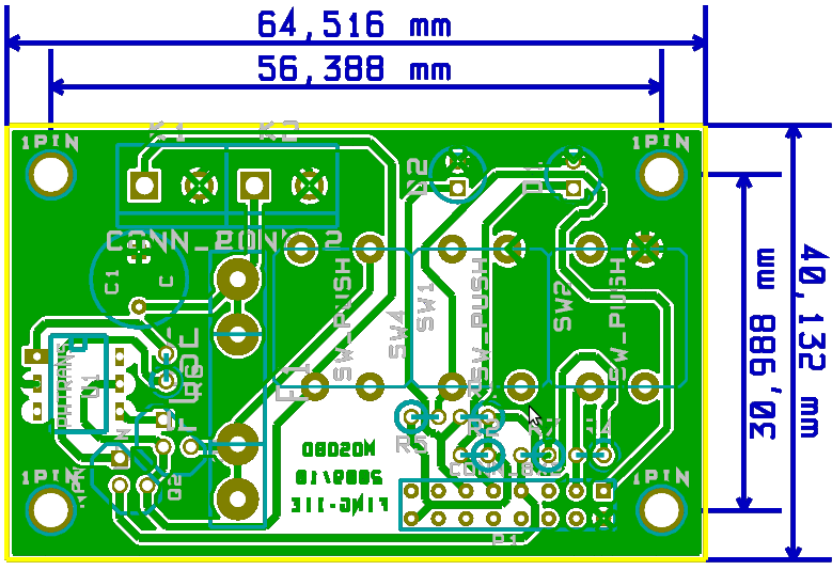


Figura 5.3: Board de la placa interfaz usuario y control de energía.

En la Figura 5.3 se puede observar el diseño de la placa interfaz y control de energía. La misma mide 6,4 cm de largo y 4,0 cm de ancho. Consta de dos borneras: una para la batería (CONN_2 K1) y otra para alimentar la placa TS−7260 (CONN_2 K2). Los headers son conectados a la SBC mediante un cable de 8x2 al DIO1.

5.4. Batería y cargador

Para alimentar el sistema se utilizarán 2 baterías de gel de 6 V y 3,2 Ah. Cada una pesa 600 g y tienen un factor de forma de 32mmx96mmx60mm. Las características de las mismas son:

- Uso en ciclo: 7.20V - 7.50V
- Uso en standby: 6.75V - 6.90V
- Corriente máxima de carga: 1.0A

El cargador de baterías seleccionado es un transformador AC-DC, modelo PW1001, con las siguientes características:

- Alimentación de entrada: 220VAC 50Hz
- Potencia: 18 W
- Alimentación de salida: 1,5-3-4,5-6-7,5-9-12 VDC
- Corriente máxima: 1000 mA

Capítulo 6

Diseño industrial

La caja seleccionada para contener la SBC, tarjeta de audio, baterías y placa interfaz es una *Pelican 1060 Micro Case*[16] que es a prueba de polvo, agua y golpes (ip67); y sus dimensiones son 20.9 x 10.8 x 5.7 cm.

Para la diagramación de los componentes que van dentro de la caja (ver Figura 6.1) se tuvo en cuenta que el usuario:

- Fuera capaz de retirar las baterías con facilidad.
- Fuera capaz de remover el pendrive fácilmente.
- Pudiera presionar los botones para comenzar y finalizar la grabación.

En la Figura 6.2, se puede observar el bozal diseñado y en donde se colocan el micrófono y la caja con el dispositivo. Para poder conectar el micrófono a la placa de audio se debió colocar un pasacables estanco de forma de no desaprovechar las propiedades de la caja (ver Figura 6.3).

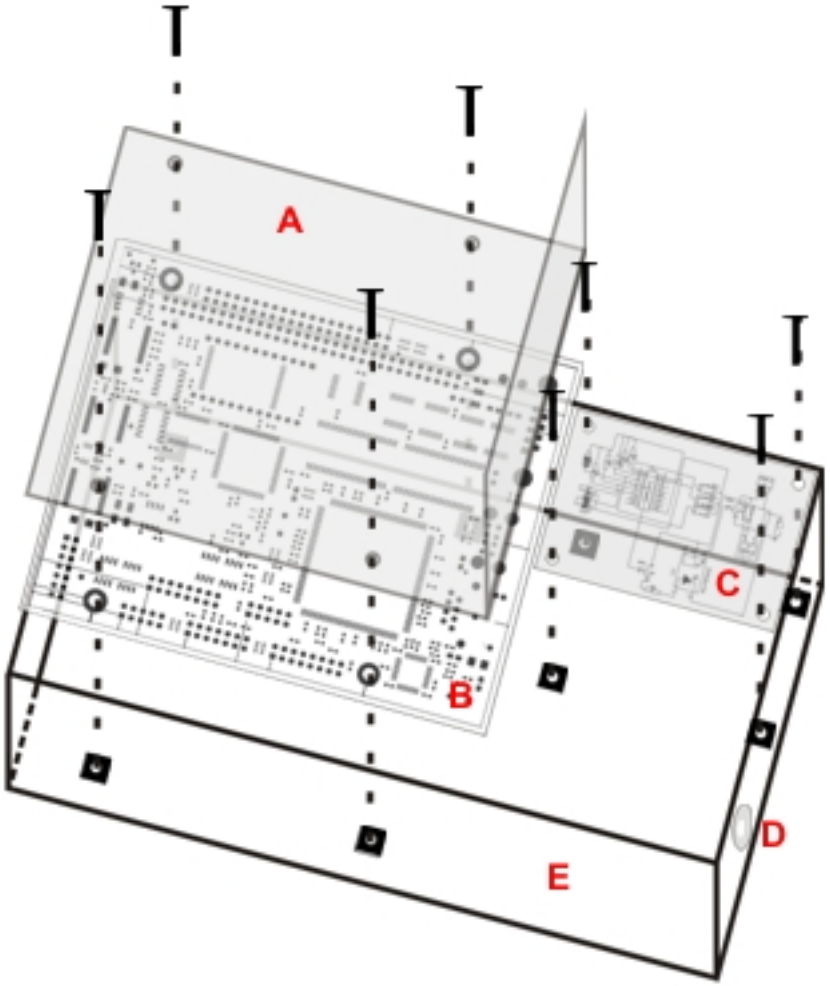


Figura 6.1: **Componentes dentro de la caja** A– Acrílico, B– SBC, C– Placa botones y leds, D– Pasacables, E– Caja

Figura 6.2: **Bozal** - La caja con el dispositivo se coloca en el cuello de la vaca mientras que el micrófono se coloca arriba del hocico.

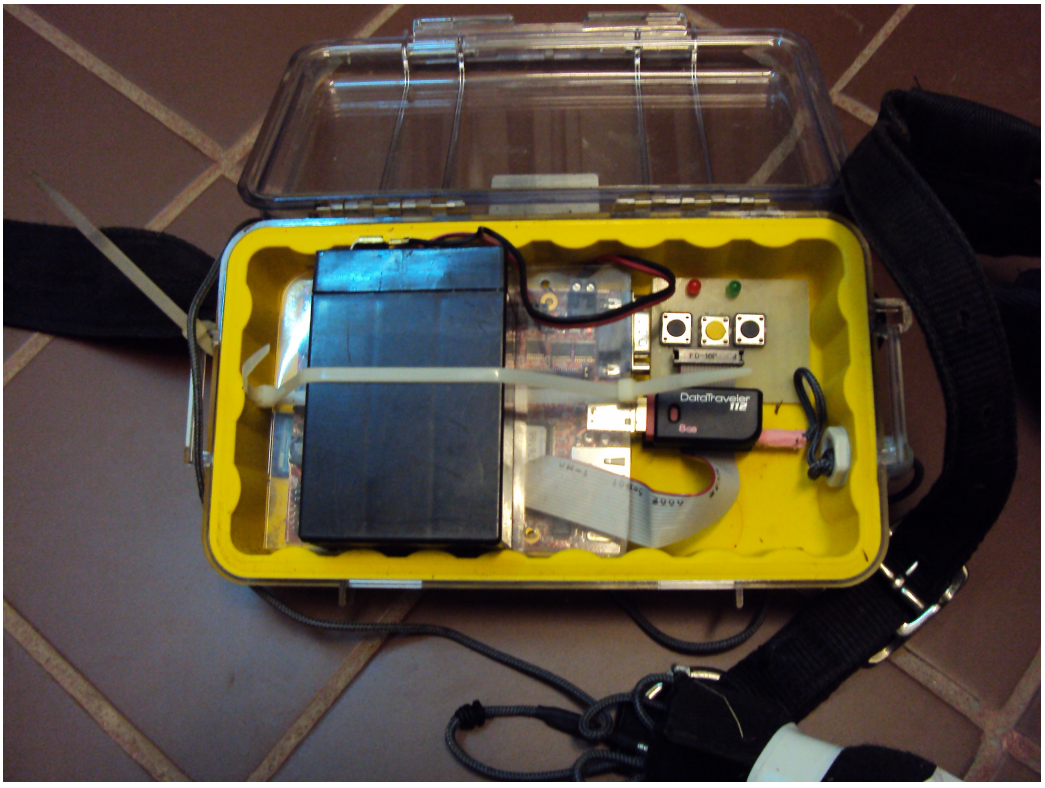


Figura 6.3: Foto de la caja con todos los componentes

Parte III

Sistema embebido

Capítulo 7

Especificación del software del sistema embebido

7.1. Introdución

A continuación se detalla el diseño del sistema embebido. El mismo se divide en los siguientes bloques:

- Inicialización del sistema.
- Adquisición y almacenamiendo de audio.
- Apagar el sistema.

Varias de las acciones que llevará a cabo el sistema embebido serán realizadas por programas y/o librerías de código abierto ya implementadas y testeadas por varias comunidades de desarrolladores. Además de avalar el correcto funcionamiento del mismo estas comunidades producen un software estable y de excelente calidad.

A continuación se detalla el funcionamiento de los mismos.

7.2. Inicialización del sistema

Luego de presionar el switch de encendido se ejecuta TS-SDBOOT e inicia el sistema operativo: configuración de autenticación de usuarios, ejecución inicioLeds,

instalación de módulos de audio y almacenamiento, configuración de la red, control de memoria libre, control de conexión del pendrive y configuración del mismo y control de pulsación de los botones.

7.2.1. Ejecutar TS-SDBOOT

El TS-SDBOOT es un bootloader para las placas TS-7260, TS-7300 y TS-7400. Al ejecutarlo (Ver Figura 7.1) se levanta la imágen del kernel que se encuentra en la primera partición de la SD para luego descomprimirla en RAM.

A continuación se ejecuta linuxrc, el cual es un archivo realizado en shell scripting que se encarga de configurar el sistema de archivos del GNU/Linux.

Después que todas estas acciones han finalizado, se levanta el sistema operativo Debian Sarge 3.1, el cual se encuentra en la tercer partición de la SD.

7.2.2. Programa linuxrc

Se ejecuta para poder configurar y trabajar desde la SD como se diseñó, modificar la configuración del sistema de archivos de la tercera partición de la SD y realizar el control de leds y switch para encender la placa.

7.2.3. Inicio del sistema operativo

Para que el sistema arranque automáticamente con nuestro software se modificó la autenticación de los usuarios y los demonios que se levantan al incio del sistema (ver Figura 7.2).

Instalación de módulos de audio y almacenamiento

Se instalaron los módulos necesarios para utilizar el hub USB de la SBC, la tarjeta de audio USB y unidad de almacenamiento en formato vfat (pendrive).

Configuración de la red

Para que sea posible interactuar con la SBC (actualizar software, modificar archivos del sistema, observar los recursos del sistema, etc.) se levanta la interfaz

de red (eth0), asignándole una ip, gateway y netmask fijos.

Arranque prog_ppal

El programa prog_ppal es el encargado de montar el pendrive y encender el led verde en el caso que el pendrive sea montado correctamente. En caso contrario enciende el led rojo y se mantiene en la espera de que el pendrive sea correctamente montado.

Además controla la existencia de la carpeta log que guarda los archivos .log que llevan el registro de las acciones realizadas por el sistema.

Controles de memoria y botones

Luego de finalizada la configuración del sistema a través de prog_ppal, se lanzan dos hilos de software que se van a encontrar activos mientras el sistema permanezca activo.

Estos hilos son el control de memoria y el control de pulsación de los botones.

El primero se realiza para evitar posibles errores de direccionamiento de memoria al llenarse el medio de almacenamiento. Al llegar al 99,5 % del uso de la memoria se manda a apagar el sistema para conservar los archivos ya almacenados.

El segundo controla las acciones que debe realizar el sistema cuando se presiona alguno de los botones.

7.3. Adquisición y almacenamiendo de audio

La adquisición del audio se realizará mediante el uso del software *Bplay*, el cual cuenta con una interfaz de grabación llamada brec. Se debe configurar el software para grabar a la frecuencia de muestreo y cantidad de bits deseados.

7.4. Apagar el sistema

Cuando se presiona el botón apagar, se enciende y apaga el led rojo a razón de una vez por segundo. La función de esta aplicación es permitirle al sistema operativo apagarse correctamente y luego quitarle la alimentación a la SBC.

En la figura 7.4 se muestra un diagrama que explica como interactúan algunos programas de el software del sistema embebido.

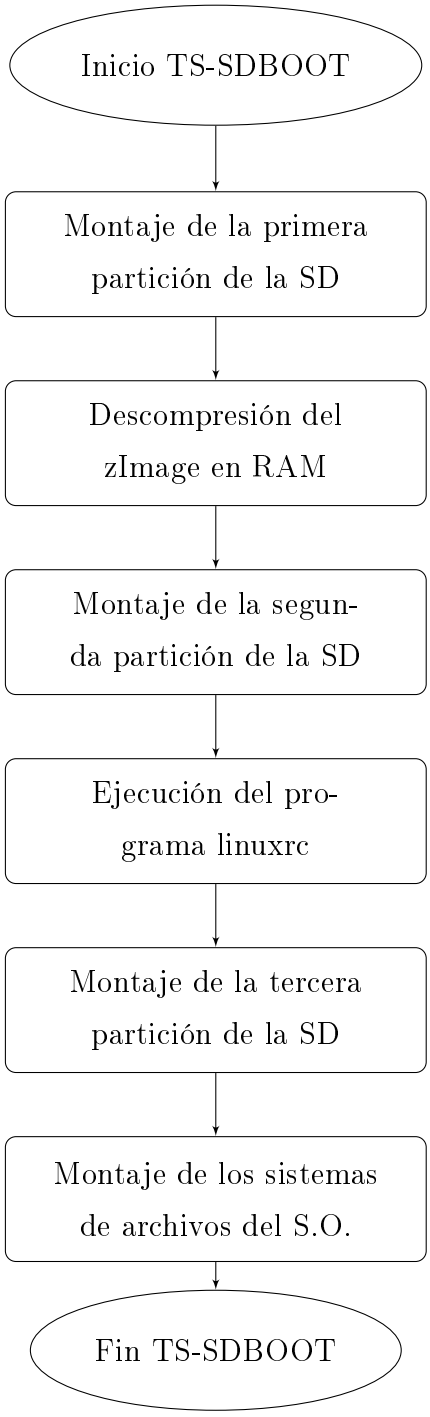


Figura 7.1: Diagrama con las tareas que se realizan al ejecutar el bootloader de la placa TS-7260

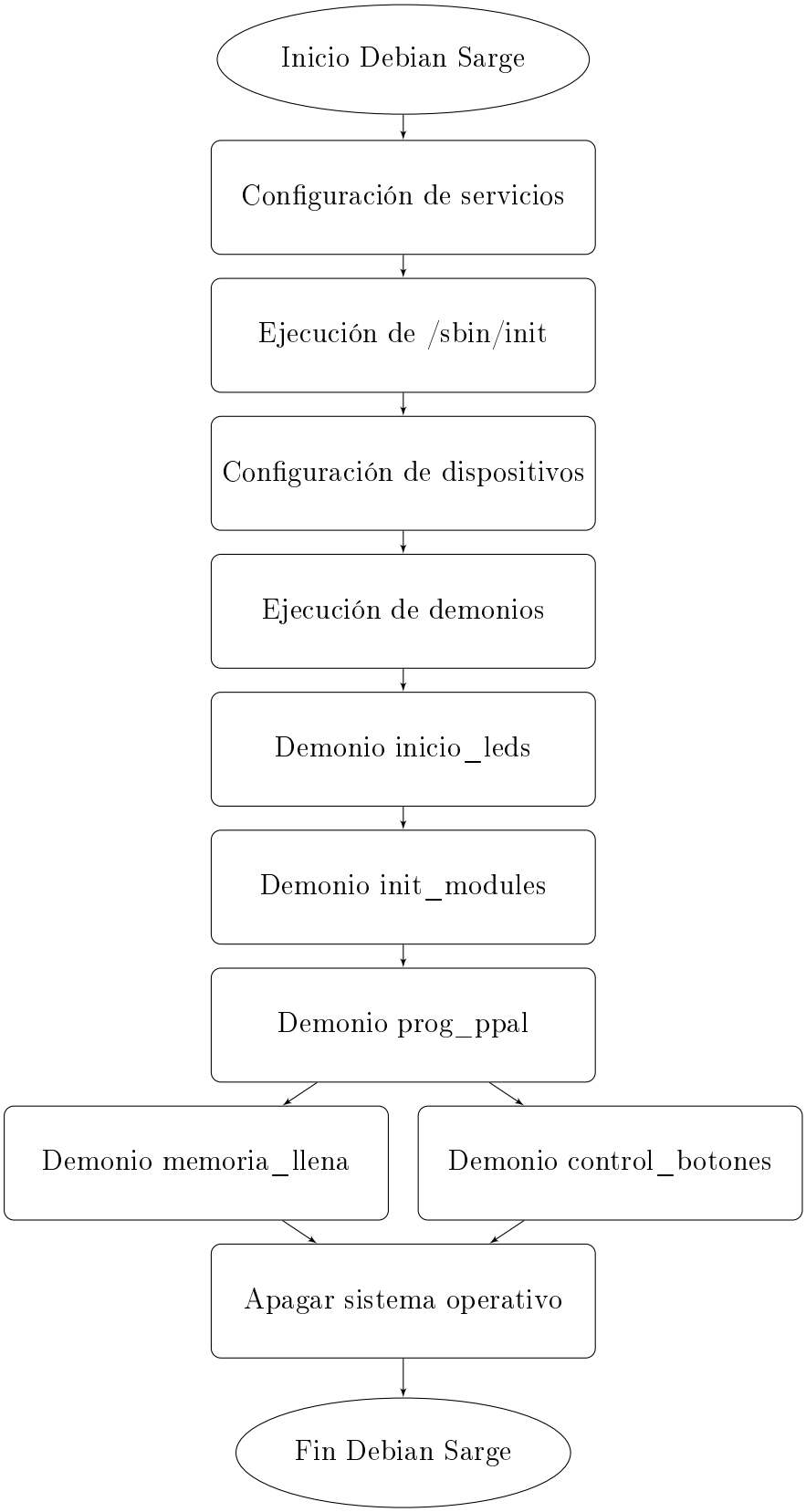


Figura 7.2: Diagrama de las tareas que se realizan al iniciar el sistema operativo

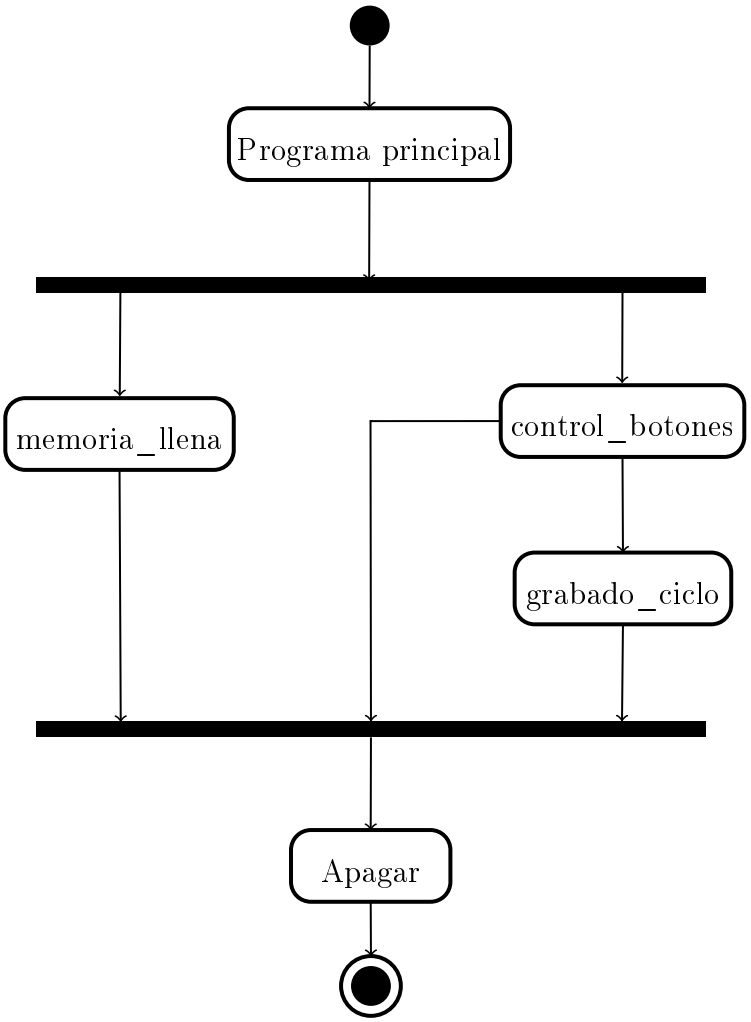


Figura 7.3: Diagrama de actividad que muestra la interacción de los distintos programas que componen el software sistema embebido.

Capítulo 8

Implementación del software del sistema embebido

En este capítulo se detalla como se comprobó el correcto funcionamiento de la SBC, la tareas que se realizaron y problemas que se presentaron para obtener el sistema operativo deseado. Finalmente se describen los archivos desarrollados y modificados.

Para tener la seguridad de que la SBC funcionara correctamente, se verificó el puerto serie, el puerto ethernet y algunos comandos básicos de GNU/Linux.

Luego se intentó bootear Debian Sarge 3.1 desde una tarjeta SD. Como el redboot que trae originalmente la placa TS-7260 no soportaba el booteo desde la tarjeta SD fue necesario cambiarlo por el TS-SDBOOT[17]. Al realizar esta acción no es posible volver a bootear desde la SDRAM a menos que se la envíe a la fábrica de origen sin embargo decidimos realizarla.

El kit de desarrollo de Technologic System incluía el código fuente de Debian Sarge 3.1, para el kernel 2.4.26-ts10 que se cross-compiló para crear la imagen comprimida `zImage` (Ver Anexo A.3). Para implementarlo exitosamente fue necesario modificar el código fuente proporcionado por Technologic System para obtener módulos de audio y tarjeta SD, direccionamientos de memoria correctos y comando de ejecución de inicio del kernel; solucionar errores de dependencia y código mal implementado. Dichos arreglos llevaron bastante tiempo debido a que se modificaron archivos desarollados en C del fuente del kernel.

Para obtener los errores del sistema operativo se utilizó la herramienta `strace`, la cual muestra las llamadas al sistema operativo y analizando los distintos errores que daba el mismo se pudo arreglar los defectos del código[1].

Posteriormente se cross compilaron los módulos y bibliotecas necesarias para controlar las distintas interfaces de la SBC que traía incluído el kit de desarrollo. También fue necesario modificar los códigos fuentes de los cuales dependía el módulo de audio, contenían diversos errores (no se podía grabar correctamente a ninguna frecuencia).

Además de utilizar el TS-SDBOOT fue necesario particionar en 3 la tarjeta SD y realizar los siguientes procedimientos:

1ra partición. Instalar el zImage del kernel de GNU/Linux deseado.

2da partición. Instalar linuxrc y modificar algunos de los parámetros de dicho programa.

3ra partición. Instalar Debian Sarge 3.1, nuevos módulos crosscompilados y archivos creados durante la realización del proyecto (Ver Sección 8.1) encargados de manejar la placa interfaz y grabar.

8.1. Descripción de los archivos desarrollados durante el proyecto

Los archivos que se desarrollaron a lo largo del proyecto para el correcto funcionamiento del sistema embebido se dividieron en distintas áreas: demonios, grabaciones, controles, comunicación. A continuación se detallan los mismos.

8.1.1. Demonios

Los Demonios, en inglés Daemon (Disk And Execution MONitor), son programas que corren en ciertos niveles de memoria interactuando directamente con el

[1]Los archivos correctos se encuentran en: mosobo.it.com.uy/mosobo_2.4.26-ts10.tar.gz. Las herramientas para cross compilar el kernel se encuentran en: mosobo.it.com.uy/toolchain.tar.gz.

kernel y no con el usuario[2].

prog_ppal

Este demonio se ejecuta al iniciar el sistema operativo. En caso que el pendrive no se encuentre correctamente montado enciende el led rojo esperando a que el usuario lo coloque correctamente dicha unidad de memoria.

Luego que el mismo es montado correctamente en `/mnt/flash`, se apaga el led rojo, se enciende el led verde, realiza el control de carpetas y registra en el log que se inició el programa principal.

control_botones

Ejecuta pulsa_botones (ver 8.1.2) al inicio del sistema operativo y es el encargado de finalizar dicho proceso al apagar el sistema.

init_modules

Se inicializan los módulos:

- pcipool
- usbcore
- usb−ohci
- usb−ohci−ep93xx
- soundcore
- audio
- fat
- vfat
- msdos
- scsi_mod
- sd_mod
- usb−storage
- zd
- zd1211
- zd1211b

Dichos modulos son los encargados de manejar el hub usb, pendrive y la tarjeta de audio.

[2] http://doc.ubuntu-es.org/Daemon - Junio 2010

memoria_llena

Ejecuta calcular_memoria_libre (ver 8.1.2). El objetivo de esta acción es no sobreescribir la memoria.

8.1.2. Controles

pulsa_botones

Se encarga de ejecutar los diferentes procesos de acuerdo al botón pulsado y a los procesos que se estén ejecutando en ese momento (es una máquina de estados).

Se mantiene en un estado de escucha permanente.

A continuación se detallan las acciones que realiza este programa una vez iniciado el sistema:

- Al presionar el botón de grabar se comienza con la grabación. En caso de que el sistema se encuentre grabando y se vuelve a presionar dicho botón no se realiza ninguna acción. La grabación se detiene únicamente al presionar el botón de apagar.
- Al presionar el botón de apagar se comienza el proceso de apagado del sistema. Se realiza independientemente del proceso que se esté ejecutando en ese momento.

apagar_sistema

Cuando el usuario presiona el botón de apagar, el sistema operativo llama al programa apagar_sistema, el cual registra en el log que se mandó apagar el sistema y realiza el proceso de apagado de la SBC.

El led rojo titila a una frecuencia de 1 Hz hasta que se apaga el sistema.

calcular_memoria_libre

Este programa analiza cada 10 segundos el estado de la memoria. En caso de que el 99.5 % o más de la misma esté ocupado, se registra en el log que la memoria está llena y se llama al programa apagar_sistema.

A este programa se le asigna la menor prioridad.

control_carpeta

Los archivos de audio son almacenados en una carpeta ubicada en el pendrive, cuyo nombre es la fecha en que se realiza la grabación.

Este programa verifica la existencia de dicha carpeta en la dirección en donde se encuentra montado el pendrive y se registra en el log la existencia o no de la misma. En caso de no existir la carpeta, se crea la misma y se registra esta acción el log.

grabar_log

Se encarga de registrar en el archivo log todas las acciones que se realizan en el sistema embebido. Este archivo es creado diariamente y se almacena en la carpeta log del pendrive.

8.1.3. Grabado

grabado_ciclo

Cuando el usuario presiona el botón de grabar, el sistema operativo llama al programa grabado_ciclo el cual define la frecuencia de muestreo, cantidad de bits y duración del archivo de audio en segundos.

Los led rojo y verde titilarán durante 10 segundos indicando al usuario que se comenzó la grabación.

El nombre del archivo de audio a grabar se determina en base a la hora de la SBC.

Se registra en el log el inicio de la grabación y finalmente se comienza la misma utilizando el software `brec` con los parámetros definidos previamente y asignándole máxima prioridad mediante el comando `nice`[3].

[3] http://linux.about.com/library/cmd/blcmdl1_nice.htm - Junio 2010

8.1.4. Archivos originales modificados

Interfaces

En el archivo `/etc/network/interfaces` se definió la ip, gateway y netmask fijos de la interfaz eth0 para poder conectarse con la SBC mediante ethernet.

resolv.conf

En el archivo `/etc/resolv.conf` se le asignó a la ip del DNS el gateway de la SBC.

linuxrc

Este programa, realizado en Shell scripting, es el encargado de montar todo el sistema operativo de la SD. Al inicio del mismo se manda ejecutar inicioLeds y alimentar_placa. El primero hace titilar el led verde de la placa a 1 Hz durante el arranque del sistema y el segundo es el encargado de alimentar la placa hasta que se mande apagar.

mountnfs

Es el demonio encargado de montar y desmontar los sistemas de archivos del sistema operativo. Está desarrollado en Shell scripting y además de las funciones que trae por defecto se le agregaron las funciones de apagar los leds y quitarle la alimentación a la placa al desmontar el último sistema de archivos.

Parte IV

Tratamiento de las señales

Capítulo 9

Consideraciones previas

9.1. Introducción

Para realizar el análisis de las señales adquiridas nos vamos a basar en el artículo *"Computational method for segmentation and classification of ingestive sounds in sheep"*[7]. El mismo explica un método para el análisis y reconocimiento automático del sonido ingestivo de las ovejas y podemos utilizarlo también con vacas dado que ambos son rumiantes y el sonido emitido posee las mismas características. Para realizar este reconocimiento, vamos a utilizar una representación apropiada de las señales acústicas y modelado estadístico que tiene su base en los modelos ocultos de Markov.

Dado que se va a trabajar con un modelo estadístico es de vital importancia realizar un buen entrenamiento de este modelo, por lo que debemos contar con un gran número de eventos, utilizando varios tipos de pastura con la finalidad de construir un modelo que sea independiente de la misma.

Cabe destacar que todo el análisis de las señales va a ser realizado desde un PC.

En este capítulo se detalla cómo se modelan las actividades pastoreo, rumia y descanso; y se da una breve introducción a la teoría de modelos ocultos de Markov y tipos de caracterización de señales consideradas.

También se presentan las herramientas que se utilizaron para implementar el sistema de reconocimiento.

9.2. Modelado de actividades y eventos

Uno de los objetivos del proyecto es poder identificar 3 actividades bien diferenciadas:

1. *Pastoreo*
2. *Rumia*
3. *Descanso*

A partir de observaciones del espectrograma de las señales de prueba se determinó que cada una de las actividades estaba formada por sucesiones de eventos:

1. *Pastoreo* - Eventos de arranque de pasto (A), silencios entre arranques (silA) y masticaciones entre arranques (MA).
2. *Rumia* - Eventos de masticaciones de rumia (R) y silencios entre masticaciones de rumia (silR).
3. *Descanso* - Eventos de descanso (D).

Cada uno de estos **eventos** se modela como un **modelo oculto de Markov** (HMM)[18] y lo que se consideró como salida observable de los HMMs fueron las características de las señales de audio.

La Figura 9.1 muestra un diagrama donde se detalla como intercatúan los eventos entre sí.

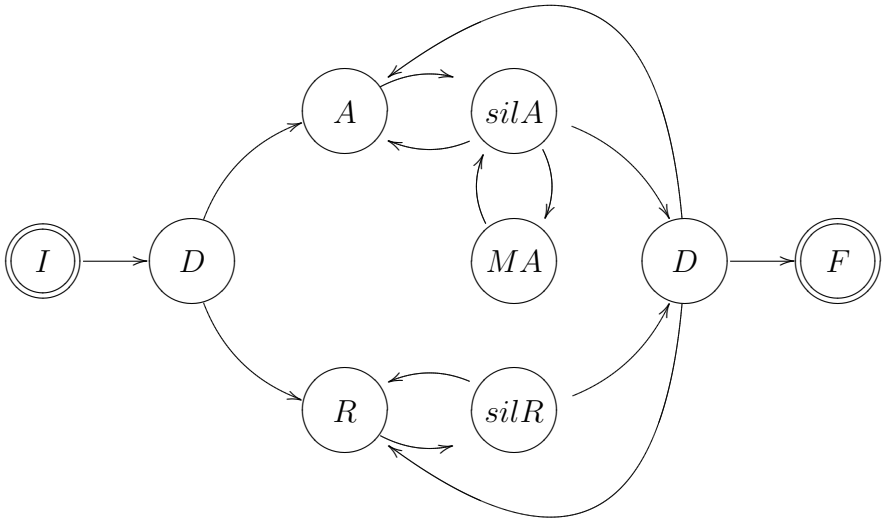


Figura 9.1: Diagrama general del lenguaje modelo utilizado para los experimentos. I significa inicio, A arranque de pasto, silA silencios entre arranques, MA masticaciones entre arranques, R masticación de rumia, silM silencio entre masticaciones de rumia y F fin.

9.3. Modelos ocultos de Markov

Una cadena de Markov es una máquina de estados finita con probabilidades de transición de un estado a otro. Un **HMM** es una máquina de estados finita, al igual que la cadena de Markov, con probabilidades determinadas para cada transisión. La diferencia con las cadenas de Markov es que los estados no son observables directamente, sino que cada estado produce una salida observable con una determinada probabilidad. El objetivo es inferir la secuencia de estados oculta.

Un HMM es la composición de dos procesos estocásticos: una cadena oculta de Markov y procesos observables. Algunos ejemplos de aplicaciones de HMM los podemos encontrar, por ejemplo, en las áreas de robótica, secuenciamiento biológico y reconocimiento de voz[19].

9.4. Caracterización de las señales

9.4.1. Introducción

Para poder realizar el modelado es necesario poder relacionar el estado oculto con la salida observable del mismo. Dicha salida observable es el conjunto de características de la señal, como por ejemplo la energía de la señal y sus propiedades espectrales. A continuación se describen dos opciones que fueron consideradas en la implementación del software de análisis.

9.4.2. LPC - Linear Predictive Coding

Es una herramienta utilizada para el procesamiento de señales de audio con la cual podemos representar las propiedades espectrales de una señal, utilizando la información de un modelo predictivo lineal[20].

9.4.3. MFCC - Mel Frequency Cepstral Coefficients

Se calculan los coeficientes cepstrales en las frecuencias de Mel, los cuales están basados en la percepción auditiva humana ya que las bandas de frecuencia están situadas logarítmicamente[21].

9.5. Elección del software para la determinación del modelo

En la actualidad existen dos herramientas de trabajo con las que podríamos generar el modelo y aplicarlo a las señales de interés:

- Matlab - Statistics Toolbox.
- HTK - Hidden Markov Model Toolkit.

9.5.1. Matlab - Statistics Toolbox

Brinda la posibilidad de estimar los parámetros del modelo utilizando el algoritmo de Baum-Welch, calcular la secuencia más probable de un modelo con el algoritmo de Viterbi y generar secuencias aleatorias a partir de un modelo dado.

Ventajas y desventajas de Matlab

Ventajas:

- Estamos habituados a utilizar Matlab y sus diferentes toolboxes.
- Implementa funciones que necesitamos.

Desventajas:

- No permite realizar el etiquetado de las señales.
- Algunas de las funciones utilizan como parámetro de entrada las probabilidades de transición de estados, parámetro que a priori desconocemos.
- Es necesario adquirir una licencia para utilizar Matlab.
- Utiliza muchos recursos del CPU y el tiempo de análisis es demasiado extenso.

9.5.2. HTK - Hidden Markov Model Toolkit

Es un conjunto de herramientas de software utilizado específicamente para construir y manipular HMMs. Aunque originalmente se creó para aplicarlo al desarrollo de sistemas de reconociemiento automático del habla, también puede utilizarse en cualquier área en la cual se tenga que resolver un problema que pueda ser enfocado desde el punto de vista de los modelos de Markov [22].

HTK es adaptable al tipo y formato de dato que queramos modelar lo que nos da una gran flexibilidad y permite el diseño de distintos tipos de reconocedores.

Es controlado por módulos de librerías. Cada una de estas librerías es un conjunto de instrucciones utilizado para realizar funciones específicas de las herramientas disponibles.

Ventajas y desventajas del HTK

Ventajas:

- Presenta una arquitectura flexible y autosuficiente.
- Se encuentra disponible para utilizarlo en diversas plataformas o sistemas operativos (Unix, Linux, Windows XP y DOS).
- Tiene una gran cantidad de algoritmos y tipos de análisis disponibles.
- Permite probar con diferentes tipos de entrenamiento del modelo.

Desventajas:

- Debe manejar y controlar un gran volumen de archivos.
- Poco amigable para realizar las llamadas a las librerías utilizadas y al ingreso de los parámetros del sistema.

9.5.3. Elección final

Luego de analizar las ventajas y desventajas de las herramientas antes descriptas, se llegó a la conclusión que el **HTK** era la mejor opción ya que si bien debíamos dedicar cierto tiempo para aprender a utilizarlo, presentaba posibilidades de probar fácilmente varios tipos de análisis sin la necesidad de introducirnos demasiado en la teoría.

9.6. HTK - Hidden Markov Model Toolkit

9.6.1. Introducción

En esta sección se detallan algunas generalidades y funciones del HTK que son utlizadas para implementar el software de análisis.

9.6.2. Organización del espacio de trabajo

Se crea una estructura de directorios de la siguiente forma:

- `data/` - Para almacenar los datos de entrenamiento y testeo. A su vez se crean 2 subdirectorios: `data/train/` y `data/test/` para separar los datos utilizados para entrenar a reconocedor de los utilizados para realizar la evaluación de performance.
- `analysis/` - Para almacenar los archivos que tienen que ver con los pasos del análisis acústico.
- `training/` - Para almacenar los archivos que tienen que ver con la inicialización y los pasos del entrenamiento.
- `model/` - Para almacenar el modelo de reconocimiento de eventos.
- `test/` - Para almacenar los archivos que tienen que ver con el testeo.

9.6.3. Creación de la base de datos de entrenamiento

Se procede a grabar señales de audio en las mismas condiciones en que va a ser utilizado el dispositivo para luego ser etiquetadas, es decir, asociar cada señal con un texto (etiqueta) que describe el contenido de la misma.

El etiquetado de las señales puede ser realizado utilizando la función `HSLab` (ver Anexo C) de HTK, sin embargo se decidió utilizar el software *Wavesurfer* con el cual estamos familiarizados, ya que también permite la función de etiquetado.

Las señales de entrenamiento las almacenamos en el directorio `data/train/wav/` y las etiquetas en `data/train/lab/`

9.6.4. Análisis acústico

Las herrramientas de reconocimientos de eventos no pueden ser procesadas directamente en las señales de audio, dichas señales primero deben ser representadas como una serie de vectores de coeficientes[1].

El análisis acústico consiste en:

[1]Es lo que va a ser tomado como la salida observable de los HMMs

- Segmentar las señales de audio en cuadros sucesivos, cuyo largo lo hicimos variar entre 25 y 100 ms, superponiéndose entre sí.
- Multiplicar cada cuadro por una función de enventanado, por ejemplo podría ser Hamming o Hanning.
- Caracterizar cada uno de los cuadros enventanados, lo que nos proporciona un vector de coeficientes acústicos brindando una representación compacta y propiedades espectrales de los mismos. Vamos a probar 2 caracterizaciones: LPC y MFCC.
- Determinar el tipo de coeficientes que vamos a utilizar en función de los resultados obtenidos del reconocimiento de eventos.

Para realizar la conversión de las formas de onda originales a la serie de vectores acústicos utilizamos la función `HCopy` (ver Anexo C). Esta función se encarga de realizar los pasos antes descriptos de acuerdo con su archivos de configuración.

```
HCopy -A -D -C analysis.conf -S targetlist.txt
```

analysis.conf es el archivo de configuración (archivo de texto) en donde se establecen los parámetros de la extracción de los coeficientes acústicos.

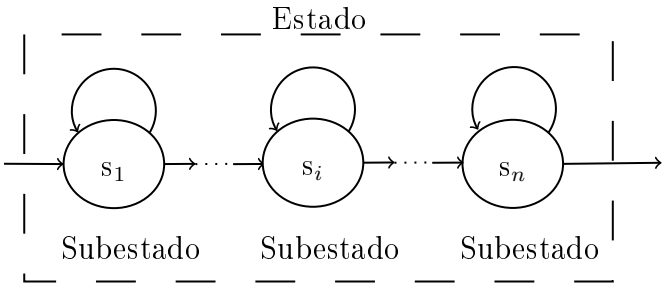
targetlist.txt especifica los nombres y ubicaciones de todas las formas de onda que queremos procesar, junto con el nombre y ubicación de los archivos de coeficientes (destino).

9.6.5. Definición de los modelos

Cada uno de los **eventos** que queremos modelar van a ser tratados como **estados** y a su vez cada uno de estos estados se modela con **subestados**.

Para cada uno de los eventos que queremos modelar se eligió una topología en función del número de subestados, la forma de las funciones de observación (asociada a cada estado) y la forma de las transiciones entre subestados.

Topología de los eventos:



En HTK, un prototipo HMM se escribe en un archivo de texto de descripción de la forma:

```
~o <VecSize> K <COEFF_TYPE>
~h "name_of_model"
<BeginHMM>
<NumStates> N
<State> 2
<Mean> K
0.0 (...) 0.0
<Variance> K
1.0 (...) 1.0
<State> 3
<Mean> K
0.0 (...) 0.0
<Variance> K
1.0 (...) 1.0

.....

<State> N-1
<Mean> K
0.0 (...) 0.0
<Variance> K
1.0 (...) 1.0
<TransP> N
a11 a12 ... a1N
a21 a22 ... a2N
..............
aN1 aN2 ... aNN
<EndHMM>
```

~o <VecSize> K <COEFF_TYPE> es el encabezado del archivo, K es el largo del vector de coeficientes y `COEFF_TYPE` el tipo de coeficiente.

~h "name_of_model" <BeginHMM> (...) <EndHMM> encierra la descripción del modelo `name_of_model`.

<NumStates> N define el número total de estados del HMM, incluyendo los 2 estados de no emisión (1 y N).

<State> 2 introduce la descripción de la función de observación del estado 2. En este caso se utiliza una función de observación gaussiana, la cual queda completamente determinada por la media y la varianza. Los estados 1 y N no se describen porque no tienen función de observación.

<Mean> K define el vector de la media (largo K) de la función de observación del estado que se está describiendo. Cada elemento se inicializa arbitrariamente en 0.0 ya que luego serán entrenados.

<Variance> K define el vector de la varianza (largo K) de la función de observación del estado que se está describiendo. Cada elemento se inicializa arbitrariamente en 1.0.

<TransP> N define la matríz de transición de estados $N \times N$ del HMM:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix}$$

donde a_{ij} es la probabilidad de transición del estado i al estado j. Los valores que son nulos indican que esas transiciones no están permitidas. Los otros valores se inicializan arbitrariamente, con la condición de que las filas de la matriz tienen que sumar 1. Al igual que la media y la varianza, estos valores también van a ser entrenados.

Para cada evento del modelo es necesario generar un prototipo. Estos prototipos son almacenados en el directorio `model/proto/`.

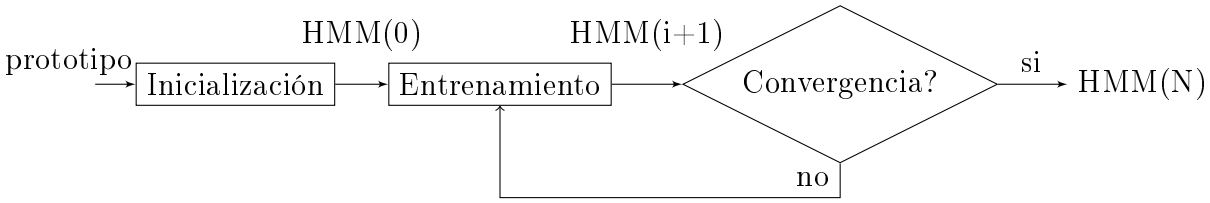


Figura 9.2: Procedimiento de inicialización y entrenamiento de los HMM

9.6.6. Entrenamiento del modelo

El procedimiento de inicialización y entrenamiento de los modelos se describe en la figura 9.6.6.

Inicialización

Antes de comenzar con el proceso de entrenamiento, los parámetros del HMM deben inicializarse con los datos de entrenamiento para tener una convergencia rápida y precisa del algoritmo de entrenamiento.

Para la inicialización de parámetros HTK ofrece 2 herramientas de inicialización: `Hinit` y `HCompV` (ver Anexo C); sin embargo en nuestro caso solo vamos a utilizar `Hinit`.

Su función es inicializar el HMM utilizando el algoritmo de Viterbi. La llamada a la función es:

```
HInit -A -D -T 1 -S training/trainlist.txt \
    -M model/hmm0 -H model/proto/hmmfile \
    -l label -L label_dir name_of_model
```

name_of_model es el nombre del HMM que queremos inicializar.

hmmfile nombre del archivo de texto de descripción que contiene el prototipo del HMM `name_of_model`.

trainlist.txt archivo de texto que detalla el nombre y la ubicación de los archivos de coeficientes de las señales de audio.

label_dir es el directorio donde se encuentran los archivos de etiquetado (.lab). En nuestro caso es `data/train/lab/`.

label indica cuál segmento etiquetado es el que debe utilizarse (nombre que se puso a la etiqueta).

model/hmm0 es el nombre del directorio donde se almacena el resultado de la inicialización del HMM. Debe ser creado previamente. El nombre del archivo de salida tiene el mismo nombre que el prototipo.

El procedimiento antes desripto tiene que repetirse para cada modelo.

Entrenamiento

La herramienta `HRest` (ver Anexo C) de HTK realiza una iteración para re-estimar los parámetros del modelo. Cada llamada a la función `HRest` realiza una iteración. Es necesario repetir el procedimiento de llamada a la función hasta que la variable de salida que brinda información del cambio producido entre iteraciones (*change*) no decrezca (en valor absoluto).

Este procedimiento se realiza para cada uno de los eventos que queremos modelar:

```
HRest -A -D -T 1 -i N -S training/trainlist.txt \
    -M model/hmmik1 -H model/hmmk0/hmmfile \
    -l label -L label_dir nameofhmm
```

N es el número de iteraciones máximas.

trainlist.txt archivo de texto que detalla el nombre y la ubicación de los archivos de coeficientes de las señales de audio.

hmmfile nombre del archivo del modelo que queremos entrenar.

hmmk0 carpeta en la que se encuentra el modelo que queremos entrenar.

hmmk1 carpeta destino en donde se va a guardar el resultado de el entrenamiento. Debe ser creada antes de ejecutar el comando. El nombre del archivo nuevo es `hmmfile`.

label nombre de la etiqueta que va a ser tenida en cuenta para el entrenamiento.

label_dir es el directorio donde se encuentran los archivos de etiquetado (.lab). En nuestro caso es `data/train/lab/`.

nameofhmm nombre del modelo dentro del archivo `hmmfile`.

9.6.7. Gramática y red

Para poder llevar a cabo el reconocimiento de eventos, es necesario definir previamente la gramática y la red que van a tener las señales que queremos analizar, la cual determinamos a partir del estudio previo de las señales de audio.

La red consiste en una lista de nodos y arcos. Los nodos representan eventos y los arcos representar transiciones entre eventos. Si bien la construcción de este tipo de archivo a mano no presenta gran dificultad, es mucho más sencillo si utilizamos la función `HParse` (ver Anexo C) para crearlo a partir de un archivo que tiene una notación de más alto nivel, el archivo de la gramática.

Los archivos de gramática se generan utilizando una extensión de la notación de Backus-Naur[23], lo que permite describir el lenguaje que se quiere utilizar a partir de expresiones regulares. Las expresiones se construyen a partir de palabras y caracteres de la forma:

| denota alternativa.

[ ] encierra opciones.

{ } denota cero o más repeticiones.

< > denota una o más repeticiones.

El archivo que define la red, `net.slf`, se obtiene a partir del archivo que define la gramática del modelo, `gram.txt` y con la función `HParse` definimos la red `net.slf`:

```
HParse -A -D -T 1 test/gram.txt test/net.slf
```

Ejemplo de archivo `gram.txt`:

```
( inicio < ej1 | ej2 > fin)
```

Ejemplo de archivo `net.slf` asociado al archivo de ejemplo `gram.txt`:

```
# Definir tamaño de la red: N=num nodos y L=num arcos
N=7    L=9
# Lista de nodos: I=num-nodo, W=palabra
I=0    W=!NULL
I=1    W=!NULL
I=2    W=inicio
I=3    W=ej1
I=4    W=!NULL
I=5    W=ej2
I=6    W=fin
# Lista de arcos: J=num-arco, S=nodo-inicio, E=nodo-fin
J=0     S=6    E=1
J=1     S=0    E=2
J=2     S=2    E=3
J=3     S=4    E=3
J=4     S=3    E=4
J=5     S=5    E=4
J=6     S=2    E=5
J=7     S=4    E=5
J=8     S=4    E=6
```

9.6.8. Reconocimiento de eventos

Para realizar el reconocimiento de eventos de una señal de audio necesitamos previamente ejecutar el comando `HCopy` (ver Anexo C) para obtener los coeficientes acústicos de la misma.

Finalmente ejecutamos:

```
HVite -A -D -T 1 -o S -H model/model1 -H model/model2 ... \
   -H model/modeln -i reco.txt -w test/net.slf \
   test/dict.txt test/hmmlist.txt test/input.txt
```

modeli con i de 1 .. n es el nombre de cada uno de los modelos que queremos incluir para hacer el reconocimiento.

reco.txt es el nombre del archivo de salida en el que se especifica tiempo de inicio y fin de la secuencia de eventos.

net.slf archivo de la red.

dict.txt es el archivo que define la correspondencia entre el nombre del modelo y el nombre que le pone como etiqueta.

hmmlist.txt es el archivo de texto que especifica todos los modelos

input.txt es el archivo que detalla el nombre y ubicación de todos los archivos de coeficientes que se quiere realizar el reconocimiento.

Capítulo 10

Implementación del Software de Análisis

10.1. Introducción

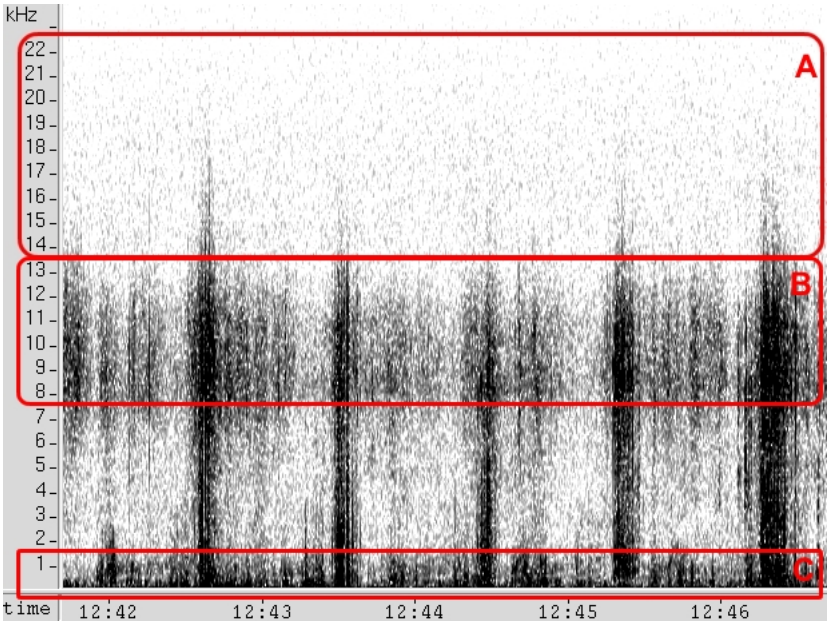
Para poder implementar el software de análisis fue necesario contar con señales obtenidas del dispositivo que se diseñó y en las mismas condiciones que va a ser utilizado. Para ello concurrimos a la Estación Experimental Prof. Bernardo Rosengurtt (**EEBR**[1]) ubicada en el departamento de Cerro Largo.

Colocamos 2 dispositivos, cada uno en animales distintos. Finalmente logramos obtener aproximadamente 11 hs de grabación en cada uno de los dispositivos.

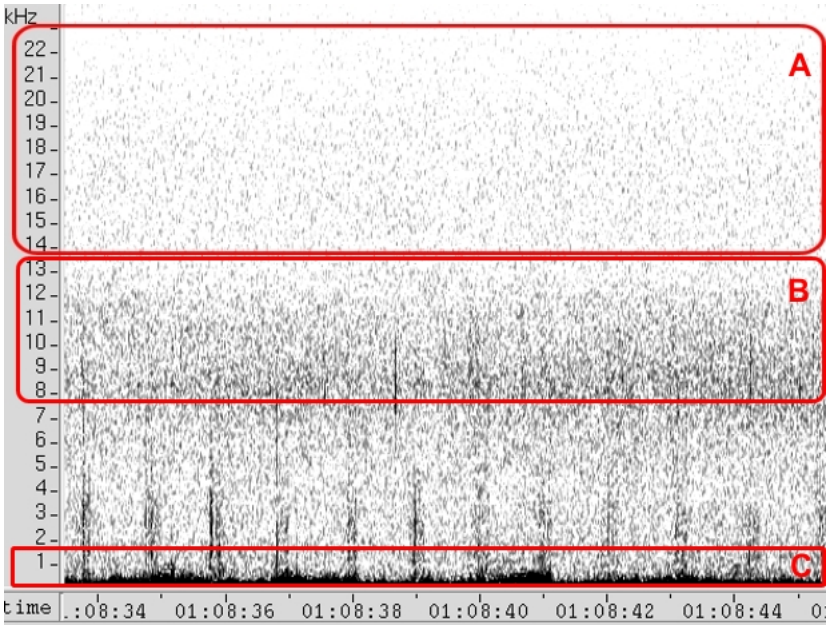
10.2. Acondicionamiento de las señales

En la Figura 10.1 se puede observar 2 señales obtenidas durante la grabación en la EEBR y algunas bandas de frecuencias marcadas en donde observamos características particulares.

[1] http://www.fagro.edu.uy/eebm.html



(a) Ejemplo de señal de pastoreo



(b) Ejemplo de señal de rumia

Figura 10.1: Ejemplos de señales obtenidas en la EEBR

A grandes rasgos notamos:

- En la zona A, que va de 14 a 24 kHz prácticamente no se tiene componentes de señal.

- En la zona B, de 8 a 14 kHz, las componentes del ruido son comparables a las de las señales de interés. En esta banda de frecuencias se ve disminuida la relación señal a ruido.
- En la zona C se encuentra principalmente el ruido introducido por el viento que tiene componentes de hasta 1 kHz.

A partir de estas observaciones resolvimos:

- Realizar un submuestreo a 16 kHz obteniendo señales que tienen componentes hasta 8kHz.
- Aplicar un filtro pasa-alto con frecuencia de corte 1 kHz.

Las señales acondicionadas se guardan en el directorio `data/train/wav`.

10.3. Etiquetado

Se etiquetaron 2 señales de audio obtenidas a partir de la prueba que se realizó en Cerro Largo. Una de ellas tenía información principalmente relacionada con eventos de rumia y la otra con eventos de arranque.

Se etiquetaron aproximadamente 15 minutos en cada una de ellas. Los archivos de etiquetas, tienen extensión `.lab` y se guardan en el directorio `data/train/lab`.

10.4. Análisis acústico

En un principio esperábamos obtener un mejor resultado al caracterizar las señales de audio utilizando LPC, tal como se realiza en el artículo *"Computational method for segmentation and classification of ingestive sounds in sheep"*[7]. Sin embargo, como en nuestro caso obtuvimos mejores resultados al aplicar MFCC, es la caracterización que vamos a utilizar.

Se decidió caracterizar las señales de audio utilizando MFCC de 13 coeficientes y el coeficiente de energía de la señal; utilizando 26 canales para el banco de filtros y un reescalamiento de los coeficientes con el valor 22. Se eligió un ancho de ventana de 50 ms con una superposición de 25 ms. En función del estudio de las

características de las señales se agregó el parámetro que tiene en cuenta la potencia del silencio y su valor es de 32 dB. Se aplicó el enventanado de Hamming ya que HTK no ofrece la posibilidad de utilizar el enventanado de Hanning, que brinda mejores resultados.

Finalmente el archivo `analysis.conf` resultó en:

```
#
# Analysis configuration file
#
SOURCEFORMAT = WAV            # Gives the format of the input files
TARGETKIND = MFCC_0_E         # Identifier of the coefficients to use
# Unit = 0.1 micro-second :
SOURCERATE = 625.00       # 1/frecuency of input files
WINDOWSIZE = 500000.0         # = 25 ms = length of a time frame
TARGETRATE = 250000.0         # = 10 ms = frame periodicity
NUMCEPS = 12                  # Number of MFCC coeffs (here from c1 to c12)
USEHAMMING = T                # Use of Hamming function for windowing frames
PREEMCOEF = 0                 # Pre-emphasis coefficient
NUMCHANS = 26                 # Number of filterbank channels
CEPLIFTER = 22                # Length of cepstral liftering
SILENERGY = 32.0              # Energy of the silence in dB
# The End
```

El resultado del análisis acústico son las características de las señales de audio (salidas observables de los HMMs) y se almacenan en archivos con extensión `.mfcc` en el directorio `data/train/mfcc`, por lo que el archivo `targetlist.txt` es de la forma:

```
data/train/wav/audio_0.wav data/train/mfcc/audio_0.mfcc
data/train/wav/audio_1.wav data/train/mfcc/audio_1.mfcc
data/train/wav/audio_2.wav data/train/mfcc/audio_2.mfcc
...
data/train/wav/audio_n.wav data/train/mfcc/audio_n.mfcc
```

Teniendo los archivo `analysis.conf` y `targetlist.txt` ejecutamos la función `HCopy`:

```
HCopy -A -D -C analysis/analysis.conf -S analysis/targetlist.txt
```

10.5. Definición de los modelos

Para todos los eventos de todas las actividades elegimos que el número de subeventos debía ser 3.

En algunos eventos, como por ejemplo el arranque de pasto, los 3 subeventos son notorios a simple vista; pero en el caso de otros eventos, como por ejemplo los silencios, podría ser discutible poner en 2 la cantidad de subeventos. Sin embargo, como es frecuente que existan ruidos en el ambiente que pueden afectar las características de estas señales decidimos agregar 1 subevento más de forma de que el modelo fuera un poco más robusto ya que es el programa mismo determina, en función de las características de las señales, si un subevento en particular se recorre o no.

Finalmente el archivo de texto que define cada uno de nuestros eventos es:

```
~o <VecSize> 14 <MFCC_0_E>
~h "nameofhmm"
<BeginHMM>
      <NumStates> 5
      <State> 2
         <Mean> 14
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
         <Variance> 14
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <State> 3
         <Mean> 14
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
         <Variance> 14
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <State> 4
         <Mean> 14
   0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
         <Variance> 14
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <TransP> 5
           0.0 0.5 0.2 0.1 0.1
           0.0 0.4 0.3 0.2 0.1
           0.0 0.0 0.4 0.3 0.3
           0.0 0.0 0.0 0.5 0.5
           0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

Los nombres de los modelos de cada evento fueron:

- Arranque de pasto - A.
- Masticación entre arranques - MA.
- Silencio de arranques - silA.
- Rumia - R.
- Silencio de rumia - silR.
- Descanso - D.

Los archivos de texto que definen cada uno de los modelos se guardan en `model/proto`.

10.6. Inicialización de los modelos

Optamos por inicializar el modelo utilizando la función `HInit` y se ejecutó el comando descripto en la especificación del software de análisis para cada uno de los modelos.

```
HInit -A -D -T 1 -S training/trainlist.txt -M model/hmm0 \
    -H model/proto/hmmfile -l label -L data/train/lab name_of_model
```

10.7. Entrenamiento de los modelos

Se obtuvo la convergencia del modelo en las siguientes cantidades de iteraciones:

- A - iteración número 7
- MA - iteración número 7
- silA - iteración número 7
- R - iteración número 8
- silR - iteración número 7
- D - no fue posible lograr la convergencia del modelo, por lo que se optó por elegirlo igual a silR.

10.8. Gramática y red

Inicialmente definimos la gramática de la seguiente forma:

```
/*
 * Task grammar - gram.txt
 */
$A0 = A silA {MA silA};
$R0 = R silR;
({[D] [{$A0} | {$R0}]} {D})
```

Al definirlo de esta manera, cualquier ruido que tuviera características similares a las de un arranque o rumia era tomado como tal. Por esta razón decidimos definir la gramática de forma de que se detecten varios eventos de arranque o rumia consecutivos. De esta forma evitamos la aparición de arranques o rumias esporádicas.

Finalmente la gramática se definió:

```
/*
 * Task grammar - gram.txt
 */
$A0 = A silA { MA silA };
$A2 = $A0 $A0;
$A4 = $A2 $A2;
$A8 = $A4 $A4;
$A16 = $A8 $A8;
$A32 = $A16 $A16;
$A64 = $A32 $A32;
$A128 = $A64 $A64 {$A0};


$M0 = M silM;
$R2 = $R $R;
$R4 = $R2 $R2;
$R8 = $R4 $R4;
$R16 = $R8 $R8;
$R32 = $R16 $R16;
$R64 = $R32 $R32;
$R128 = $R64 $R64 {$R0};

$D2 = D D;
$D4 = $D2 $D2;
$D8 = $D4 $D4;
```

```
$D16 = $D8 $D8;
$D32 = $D16 $D16;
$D64 = $D32 $D32 {D};

({$D64} {[$A128 | $R128] [$D64]})
```

La red (`net.slf`) la definimos ejecutando:

```
HParse -A -D -T 1 test/gram.txt test/net.slf
```

10.9. Reconocimiento de eventos

Para obtener el archivo `reco.txt` ejecutamos:

```
 HVite -A -D -T 1 -o S -H model/model1 -H model/model2 ... \
    -H model/modeln -i test/reco.txt -w test/net.slf
    test/dict.txt test/hmmlist.txt test/testlist.txt
```

El archivo `dict.txt` resultó en:

```
silR [0] silR
silA [1] silA
D [2] D
A [3] A
MA [4] MA
R [5] R
```

La salida de este comando nos proporciona información del inicio y fin de cada uno de los eventos que se reconocen a lo largo de las señales analizadas; es decir, los eventos A, MA, silA, R, silR y D que se encuentran en nuestra señal.

Como el objetivo de este análisis es obtener intervalos de actividades de pastoreo, rumia y descanso es necesario procesar los datos obtenidos de la función `HVite`.

10.10. Reconocimiento de actividades

A partir del reconocimiento de eventos obtenemos la sucesión de eventos que se observan en nuestra señal; pero a partir de dicha sucesión se deben determinar las actividades que se realizaron.

Se determina el comienzo de la actividad de pastoreo cuando se observa un evento de arranque (A) y finaliza con un evento de rumia (R) o descanso (D). De la misma forma se determina la actividad rumia que comienza con un evento de rumia (R) y finaliza con un evento de arranque (A) o descanso (D). La actividad descanso resulta de la sucesión continua de eventos de descanso.

Parte V

Aplicación cliente

Capítulo 11

Aplicación cliente

11.1. Introducción

La aplicación cliente es la encargada de interactuar con el usuario la cual le permitirá copiar y/o borrar los datos de un pendrive, analizar las señales y observarlas. Además, el usuario mediante esta aplicación podrá realizar un formateo y renombre de la unidad (permitiendo identificar el bovino que se desea estudiar).

La aplicación cliente es quien integra todas las etapas descriptas en los capitulos anteriores. Permite obtener los datos adquiridos por el dispositivo el cual se coloca en el rumiante durante todo el día y luego procesarlos obteniendo las actividades realizadas.

11.2. Análisis de requerimientos

11.2.1. Requerimientos funcionales

Las funciones que debe realizar la aplicación cliente son:

Copiar datos - Selecciona la unidad de la cual se quiere obtener las señales de audio y realiza la copia de las señales de audio que se encuentran en dicha unidad hacia la PC.

Borrar datos - Selecciona la unidad de la cual se quiere borrar los datos y elimina los datos que contiene dicha unidad.

Analizar datos - Es el encargado de realizar las llamadas funcionales necesarias para realizar el análisis de las señales de audio deseadas: decimarlas, filtrarlas, reconocimiento de eventos y reconocimiento de actividades.

Observar datos analizados - Muestra en pantalla el resultado del análisis de las señales de audio.

Formateo y/o cambio de nombre de pendrive - Formatea y/o renombra la unidad seleccionada.

Los detalles de los cursos básicos y alternativos de los casos de uso se encuentran en el Anexo D.1

11.2.2. Requerimientos no funcionales

La aplicación cliente deberá ser portable, dado que en primera instancia se usará en un sistema operativo GNU/Linux pero no se descarta que en un futuro se utilice en otras plataformas, por ejemplo Windows.

11.2.3. Asunciones realizadas

Dado que varios comandos deben ser ejecutados con permisos de super usuario, se asume que el usuario iniciará la aplicación cliente como root.

11.3. Elección del software

Los datos son almacenados por el dispositivo en un pendrive y el usuario debe retirar el mismo y colocarlo en un PC.

Como el análisis de las señales obtenidas se realiza en un PC (con cualquier sistema operativo), la aplicación cliente deberá ser portable (que se pueda ejecutar en cualquier terminal).

A su vez debe poder interactuar con otros lenguajes de programación, ser capaz que llamar a procedimientos o funciones desarrollados en otros lenguajes (C, Octave, HTK).

Por estas razones es que se decidió implementar la aplicación cliente en **Java**, que a su vez posee muchas librerías disponibles y útiles para desarrollar nuestra aplicación.

11.4. Implementación de la aplicación cliente

11.4.1. Interfaz

La interfaz de la aplicación cliente se desarrolló utilizando la librería *Java Swing* que es la librería de interfaces de Java más utilizada. Algunas ventajas de esta librería son:

- Swing está basado en la arquitectura MVC (Model View Controller) y por ellos es independiente del hardware y sistema operativo.
- Los componentes de Swing soportan más características y son más livianos.
- Funciona en distintas versiones de Java.

La interfaz muestra en pantalla los menú y botones que permite interactuar al usuario con las diferentes funciones de la aplicación cliente. La Figura 11.1 muestra la interfaz gráfica.

Figura 11.1: Interfaz gráfica de la aplicación cliente.

11.4.2. Menú

El menú está formado por las siguientes opciones:

- Archivo
  - Salir
- Editar
  - Borrar datos - realiza la misma función que el botón *Borrar datos* (Ver Sección 11.4.3).
  - Copiar datos - realiza la misma función que el botón *Copiar datos* (Ver Sección 11.4.3).

- Ver
  - Analizar datos - realiza la misma función que el botón *Analizar datos* (Ver Sección 11.4.3).
  - Obervar datos - realiza la misma función que el botón *Observar datos* (Ver Sección 11.4.3).
- Herramientas
  - Cambiar nombre - al seleccionar esta opción, la aplicación le permite al usuario seleccionar la unidad a la que se desea cambiar el nombre y se le pide que ingrese el nuevo nombre. Se ejecutan los comandos `umount` para desmontar la unidad, y con el comando `mlabel` se le asigna el nuevo nombre.
  - Formatear pendrive - al seleccionar esta opción, la aplicación le permite al usuario formatear la unidad deseada. Se ejecutan los comandos `umount` para desmontar la unidad, y con el comando `mkfs` formatea la misma.
- Ayuda
  - Sobre Mosobo - muestra la versión del software, el logo y el sitio web del proyecto.

11.4.3. Botones

Copiar datos / Borrar datos

En ambos casos se utilizó la clase *File* la cual permite realizar las acciones de copiar, mover y borrar archivos y directorios independientemente del sistema operativo en el cual se ejecute la aplicación.

Analizar datos

Al seleccionar la opción *Analizar Datos*, se despliega en pantalla un selector de directorios y el usuario debe seleccionar el directorio que contiene los archivos que

se desean analizar. Los nombres de estos directorios contienen la fecha en que se realizó la grabación.

Luego de este paso se procede a realizar las actividades de (ver Figura 11.2):

Preparación de las señales: decimar, cortar, filtrar y convertir a .mfcc.

Reconocimiento: de eventos y actividades

Post-reconocimiento: elimina períodos cortos de actividades.

Preparación de las señales

Introducción Para ejecutar todos los comandos en Java como si se estuvieran ejecutando en la consola, se utilizó la clase *Runtime* la cual permite ejecutar comandos del sistema operativo en el que se está ejecutando la aplicación cliente.

Decimar La frecuencia de muestreo de los archivos de audio que se obtienen con el dispositivo es de 48 kHz, por esta razón se implementó un decimador en C. Para ejecutar el mismo es necesario pasarle como parámetros el archivo original (a 48kHz), el nombre del archivo que se desea generar y finalmente la frecuencia de muestreo del nuevo archivo (en Hz). A partir de las resoluciones tomadas para el acondicionamiento de las señales, se determinó trabajar con archivos de audio .wav de 16 kHz, por lo que el tercer parámetro es 16000.

El comando que se ejecuta para decimar los archivos es:

```
./decimar wav48.wav wav16.wav 16000
```

Cortar Este requerimiento surge a partir de la imposibilidad de aplicarle un filtro en Octave a una señal de 4 hs de duración muestreada a 16 kHz. Por ello se decidió cortar los arhivos cada 1 hora y así aplicar el filtro y reconocer las señales sin problemas.

Para cortar los archivos se desarrolló una aplicación en C que permite cortar el archivo deseado en períodos de tiempo deseados. Para ejecutar el mismo es necesario pasarle 3 parámetros: cantidad de minutos de los archivos de salida, el archivo a cortar y el nombre del archivo de salida. El nombre de los archivos creados

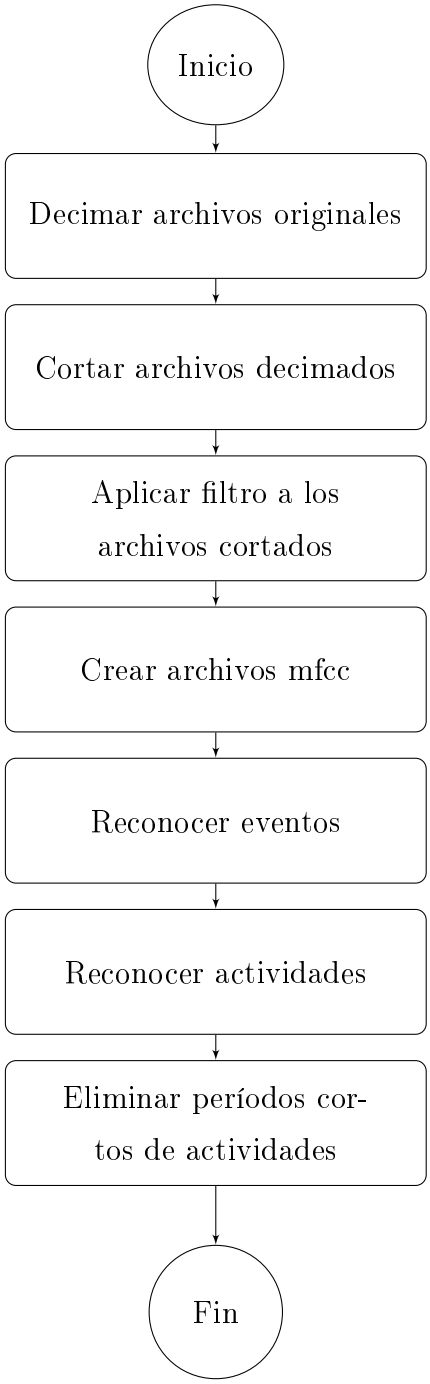


Figura 11.2: Diagrama de las tareas que se deben realizar para reconocer las actividades

están formardos por el nombre del archivo de salida que se pasó como parámetro seguido por un número que indica el orden en el que fueron creados.

El comando que se ejecuta para cortar los archivos es:

```
./cortar 60 wav4hs.wav wav1hs
```

La salida a este comando resulta en los archivos `wav1hs1.wav`, `wav1hs2.wav`, `wav1hs3.wav` y `wav1hs4.wav`.

Filtrar Para el correcto reconocimiento de los eventos y actividades los archivos de audio son filtrados utilizando un filtro pasa-altos de Butterworth de orden 6 y frecuencia de corte 1 kHz implementado en Octave. Para ello se realizó una función en Octave que toma como parámetro de entrada el archivo .wav que se quiere filtrar y el nombre con que se quiere guardar la señal filtrada.

Para ejecutar dicha función en java se utilizó la librería *joPAS*, la cual permite ejecutar comandos de octave en Java.

El comando que se ejecuta para realizar el filtrado es:

```
filtro(wav_origen.wav,wav_destino.wav)
```

Crear archivos .mfcc Para crear los archivos .mfcc se ejecuta:

```
HCopy -A -D -C analysis.conf -S targetlist.txt
```

Reconocimiento

Reconocimiento de eventos Para llamar a la función de HTK que realiza el reconocimiento de eventos ejecutamos:

```
HVite -o S -H modelos/A -H modelos/D modelos/M -H modelos/MA \
-H modelos/silA -H modelos/silM -i reco_i.txt -w net.slf dict.txt \
hmmlist.txt archivo.mfcc
```

Reconocimiento de actividades Al ejecutar el comando `HVite` se obtiene el reconocimiento de eventos; luego es necesario reconocer las actividades. Para esto creamos un nuevo archivo el cual indicamos el tiempo inicial, final y actividad. El principio y fin del pastoreo se determina a partir del primer arranque (A) hasta que

se detecte un descanso (D) o rumia (R). El principio y fin de la rumia se determina a partir del primer evento de rumia (R) hasta que se detecte descanso (D) o un arranque (A). Este procedimiento fue implementado en Octave.

Post-reconocimiento Debido a la presencia de ruidos en las señales de audio (como por ejemplo sonidos de pájaros) el proceso de reconocimiento de eventos realiza una detección errónea durante un corto período de tiempo y estos errores se ven reflejados después en el reconocimiento de actividades; por lo tanto se presentan actividades de pequeña duración en lugares que no deberían estar ya que esta etapa no tiene en cuenta la duración de las mismas.

Además desde el punto de vista del comportamiento ingestivo no tiene sentido que el animal realice una actividad por unos pocos segundos o minutos y luego cambie a otra.

Por estas razones es que se implementó una etapa de post-reconocimiento que mejora este problema ya que sí tiene en cuenta la duración de las actividades. Este post-reconocimiento está implementado en Java.

El algoritmo desarrollado consiste en seleccionar una ventana de tiempo fijo e ir trasladando la misma a lo largo de todo el tiempo de grabación de cada día. En cada intervalo se evalúa que actividad se realizó durante más tiempo y se le asigna a dicho intervalo de tiempo la actividad que se haya realizado durante más tiempo.

Ancho de la ventana del algoritmo de entrenamiento Para determinar cuál era el ancho de la ventana del algoritmo de entrenamiento comparamos las gráficas obtenidas cuando aplicamos el algoritmo de post-procesamiento tomando varios anchos de ventana de tiempo: 30 segundos, 1 minuto, 2 minutos, etc.

El ancho de ventana eligido es aquel que maximiza el porcentaje de aciertos de las actividades y se determinó que fuera de 1 minuto. Para este ancho de ventana el error máximo sería de 30 segundos.

En la Figura 11.4.3 se muestra un ejemplo de la gráfica obtenida luego del análisis de las señales obtenidas en Cerro Largo.

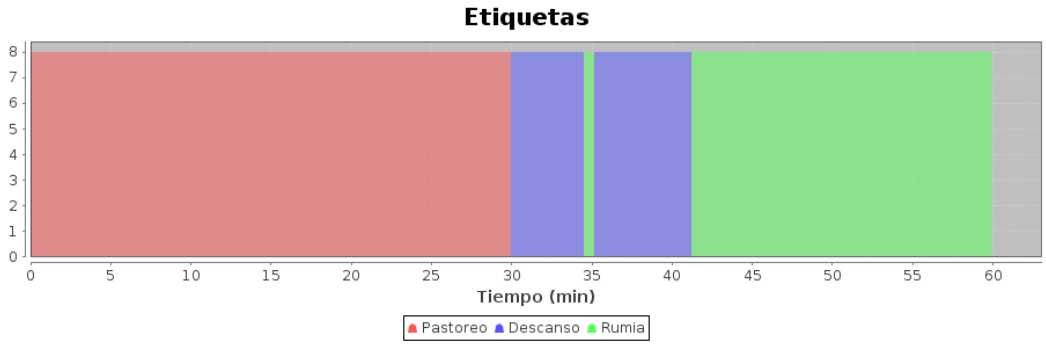
Observar datos Al seleccionar *Observar datos* se despliega en pantalla un selector de directorios y el usuario debe seleccionar el directorio que contiene los

archivos que desea observar[1].

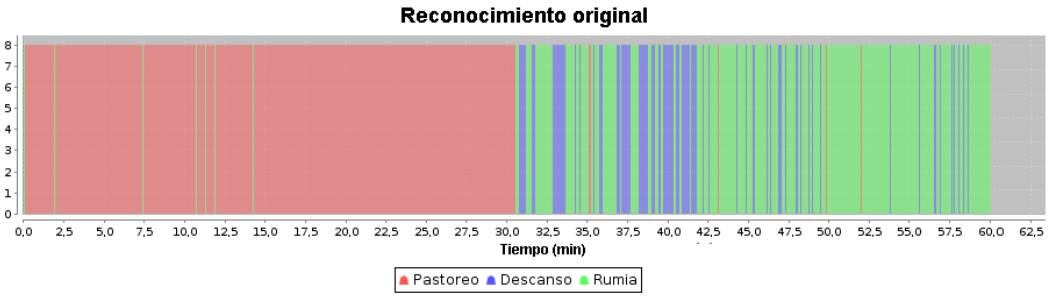
Luego se muestra en pantalla una gráfica indicando que actividad realizó el bovino a lo largo del día. Además se informa la cantidad de tiempo que realiza cada actividad.

Para mostrar los datos en pantalla se utilizó la librería *JFreeChart* de Java.

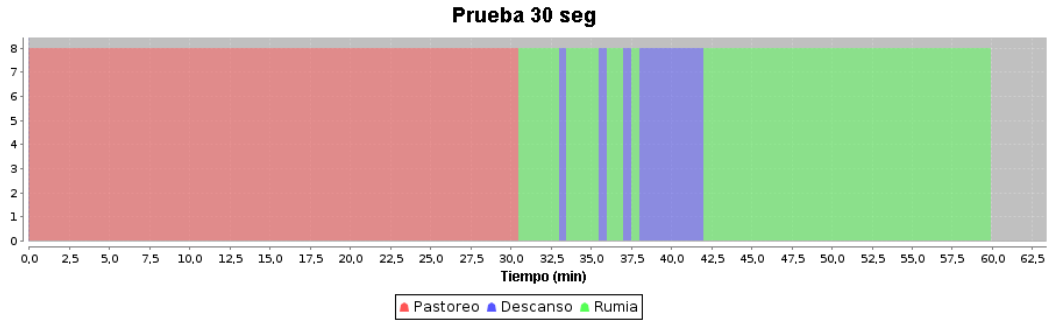
[1]Para poder observar los datos, previamente debe analizarlos seleccionando la opción analizar datos.



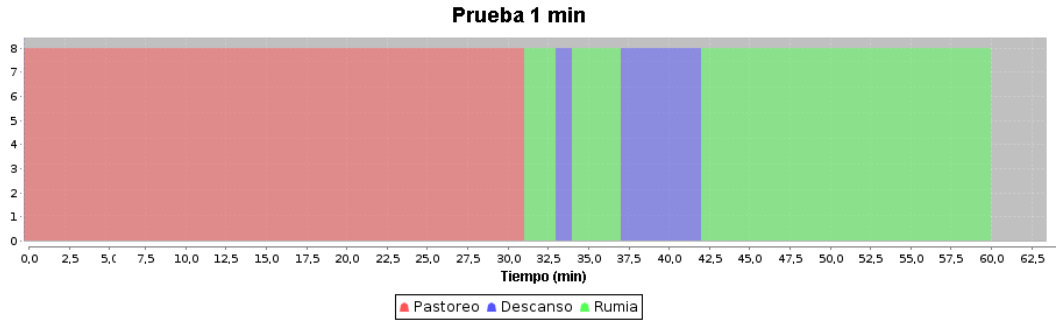
(a) Reconocimiento visual y auditivo.



(b) Resultado original del reconocimiento de actividades.



(c) Resultado del reconocimiento de actividades al elegir una ventana de 30 segundos.



(d) Resultado del reconocimiento de actividades al elegir una ventana de 1 minuto.

Figura 11.3: Comparación entre el resultado obtenido del reconocimiento de actividades original y el reconococimiento de actividades aplicando diferentes anchos de ventana.

Parte VI

Resultados

Capítulo 12

Resultados

12.1. Introducción

Luego de entrenar los modelos de eventos y utilizarlos para realizar el reconocimiento de eventos y actividades evaluamos cuantitativamente el resultado obtenido. Esta evaluación se realiza en función del resultado del reconocimiento de actividades.

Realizamos una inspeccón visual y auditiva de una señal tomada a partir de los ensayos realizados en Cerro Largo y a partir de esto recurrimos a hallar el **porcentaje de acierto** y **porcentaje de error** de cada una de las actividades.

A continuación se define el porecentaje de acierto y error de cada actividad y finalmente se presentan los resultados.

12.2. Porcentaje de acierto

En la Ecuación 12.1 definimos el porcentaje de acierto de la actividad A (simbolizando pastoreo, rumia o descanso). Primero se etiqueta la señal a evaluar para determinar todos los intervalos de la actividad A presentes en dicha señal. Para cada uno de los intervalos determinados en la señal se realiza una sumatoria de los tiempos de esa actividad detectados en un intervalo. Finalmente se dividen las

sumatorias entre el tiempo total etiquetado de la actividad a evaluar.

$$Pa_A = \frac{100}{t_A} \sum_{i=1}^{K_A} \left(\sum_{j=1}^{N_i} t_{Aij} \right) \tag{12.1}$$

En la Ecuación 12.1:

A simboliza la actividad (pastoreo, rumia o descanso).

$\mathbf{K}_A$ es la cantidad de intervalos de la actividad A en la señal a evaluar.

$\mathbf{t}_A$ es el tiempo total etiquetado de la actividad a evaluar A.

$\mathbf{N}_i$ es la cantidad de intervalos de la actividad A en t_{Ai} detectados en el reconocimiento.

$\mathbf{t}_{Aij}$ es el intervalo de tiempo j de la actividad A en t_{Ai} en el reconocimiento.

12.3. Porcentaje de error

En la Ecuación 12.2 se define el porcentaje de error $Pe_{A_1A_2}$ como el porcentaje de error de reconocer la actividad A_1 cuando la actividad real era A_2.

$$Pe_{A_1A_2} = \frac{100}{t_{A_2}} \sum_{i=1}^{K_{A_2}} \left(\sum_{j=1}^{N_{A_1 i}} t_{A_1 ij} \right) \tag{12.2}$$

En la Ecuación 12.2:

$\mathbf{A}_1$ simboliza la actividad (pastoreo, rumia o descanso).

$\mathbf{A}_2$ simboliza la actividad (pastoreo, rumia o descanso).

$\mathbf{K}_{A_2}$ es la cantidad de intervalos de la actividad A_2 en la señal a evaluar.

$\mathbf{t}_{A_2}$ es el tiempo total etiquetado de la actividad A_2 en la señal a evaluar.

$\mathbf{N}_{A_1 i}$ es la cantidad de intervalos detectados de la actividad A_1 en el intervalo i de la actividad A_2 en el reconocimiento.

$\mathbf{t}_{A_1ij}$ es el intervalo de tiempo j de la actividad A_1 en el intervalo i de la actividad A_2 en el reconocimiento.

Finalmente en la Ecuación 12.3 definimos Pe_{A_1}, el porcentaje de error de la actividad A_1:

$$Pe_{A_1} = \frac{t_{A_2}.Pe_{A_1A_2} + t_{A_3}Pe_{A_1A_3}}{t_{A_2} + t_{A_3}} \tag{12.3}$$

Donde:

$$t_{A_2} = \sum_{i=1}^{K_{A_2}} t_{A_2i}$$

$$t_{A_3} = \sum_{i=1}^{K_{A_3}} t_{A_3i}$$

12.4. Resultados

Los resultados obtenidos de los porcentajes de acierto a partir de una señal de 1 hora de duración son:

$$\boxed{\begin{array}{c} Pa_P = 100\,\% \\ Pa_R = 100\,\% \\ Pa_D = 52\,\% \end{array}}$$

Los porcentajes de error son:

$$\left.\begin{array}{c} Pe_{PR} = 0\,\% \\ Pe_{PD} = 4.3\,\% \end{array}\right\} \Rightarrow \boxed{Pe_P = 1.2\,\%}$$

$$\left.\begin{array}{c} Pe_{RP} = 0\,\% \\ Pe_{RD} = 61\,\% \end{array}\right\} \Rightarrow \boxed{Pe_R = 16\,\%}$$

$$\left.\begin{array}{c} Pe_{DP} = 0\,\% \\ Pe_{DR} = 0\,\% \end{array}\right\} \Rightarrow \boxed{Pe_D = 0\,\%}$$

El bajo porcentaje de acierto de la actividad descanso se debe principalmente a la existencia de ruidos en el ambiente que hacen que las señales tengan características de eventos que se producen durante la actividad de rumia. Y de acuerdo a

los cálculos realizados se llega a la conclusión de que el 16 % del tiempo reconocido como actividad de rumia corresponde en realidad a la actividad descanso.

Una posible solución a este problema es colocar el micrófono pegado en la frente del animal. De esta forma se aprecian componentes de mayor frecuencia en los eventos de rumia.

Si bien el peor resultado se obtuvo en la actividad descanso, la actividad que más les interesa a los investigadores que se detecte correctamente es el pastoreo. Debido a las características de las señales de audio durante el pastoreo solo el 1.2 % del tiempo detectado por el reconocimiento corresponde a otra actividad, lo cual consideramos un muy buen resultado.

Parte VII

Conclusiones y perspectivas

Capítulo 13

Conclusiones

Se desarrolló a nivel de primer prototipo un dispositivo capaz de grabar y almacenar correctamente los sonidos emitidos por bovinos y que puede permanecer a la intemperie junto con los mismos sin presentar problemas de funcionamiento. El dispositivo es de fácil colocación, no interfiere con el comportamiento de los animales, no les genera lesiones y además su costo, que ronda los 550 USD en materiales, es relativamente bajo en comparación con otros dispositivos que se utilizan para este fin.

Se logró una independencia energética de 24 horas comprometiendo el peso del dispositivo que se encuentra en el entorno de los 1700 gramos sin bozal.

Para determinar qué se iba a utilizar para realizar las grabaciones se evaluaron varias opciones. Se decidió utilizar una single board computer para facilitar el desarrollo de las aplicaciones (el sistema operativo trae implementado módulos de audio y almacenamiento). Los factores más importantes que se tuvieron en cuenta para la selección de la SBC fueron: consumo, cantidad de puertos USB, soporte de tarjetas SDHC, RTC, sistema operativo que soporta, factor de forma y costo. En función de estos factores se decidió utilizar la placa TS-7260 de ARM junto con la tarjeta de sonido USB 3D Sound modelo HY554.

Si bien el consumo de esta placa fue el factor que más se tuvo en cuenta para su selección, el uso de periféricos (necesarios para la implementación) aumentó dicho consumo significativamente. En particular, al iniciar la placa aumentó un 60 % y al grabar y almacenar aumentó un 10 % con respecto al consumo en el estado de

espera.

Para que el usuario interactúe con el sistema embebido se diseñó una interfaz simple que consta de 3 pulsadores y 2 leds. Al pulsar los distintos botones se puede iniciar el sistema, comenzar a grabar y apagar el sistema. Con distintas configuraciones de secuencias en los leds se indica que acción esta realizando el sistema.

También se estudiaron distintas alternativas para encender y apagar el sistema embebido correctemente. Por un lado se buscaron en el mercado soluciones ya implementadas. Las mismas no resultaban convenientes debido a su elevado costo y tamaño; y además era necesario hacer una importación porque no se encontraban en el mercado local. Por esta razón se optó por desarrollar un diseño propio. Se diseñaron y verificaron varias opciones eligiendo el diseño de menor tamaño y consumo.

Se seleccionó la caja Pelican 1060 Micro Case, a prueba de golpes, polvo y agua, para almacenar la SBC junto con periféricos, la placa interfaz y baterías. Otra de las características de la caja seleccionada es el sistema de cierre robusto y simultáneamente permite retirar y colocar la unidad de almacenamiento y baterías; y pulsar los botones para controlar el sistema embebido.

Se diseñó un bozal que sujeta firmemente la caja con todos los componentes contra el animal y permite que el micrófono se ubique en la zona del hocico.

Es importante destacar que el costo del diseño industrial (caja, importación y bozal) es inferior a 60 dólares.

A pesar de haber tenido varios inconvenientes con las herramientas de desarrollo brindadas por el fabricante de la SBC (código fuente mal implementado, errores de direccionamiento de memoria, dependencias, etc), se logró implementar exitosamente el software y sistema operativo del sistma embebido, cumpliendo con todos los objetivos planteados. Estos inconvenientes generaron un retraso significativo durante el desarrollo del proyecto.

Partiendo de los ensayos realizados con el prototipo desarrollado, se extrajeron las características de los sonidos emitidos por los bovinos y se determinó como se deben tratar las señales obtenidas.

Para que el sonido ambiente interfiera lo menos posible con los resultados del

reconocimiento, se implementó un filtro pasa-altos que elimina algunas distorsiones provocadas por el viento e insectos. Otros sonidos que pueden afectar el reconocimiento, pero que no se pudieron eliminar en su totalidad son: la lluvia, cantos de pájaros, sonidos de terneros, voz humana, tractores, etc. En el caso particular de la lluvia, los investigadores del área no están interesados en analizar el comportamiento ingestivo de los bovinos bajo esta condición debido a modificaciones en el comportamiento del animal.

Utilizando las herramientas de desarrollo HTK se implementó un sistema de reconocimiento de eventos de las actividades que realizan los rumiantes. El mismo está basado en modelos estadísticos y para determinarlo se probaron y variaron parámetros como: la caracterización de las señales de audio (linear predictive coding y mel frecuency cepstral coefficients), ancho de la ventana de análisis, cantidad de estados de cada modelo, máquina de estados que determina la interacción entre los modelos y cantidad mínima de repetición de cada evento.

A partir del sistema de reconocimiento de eventos se implementó un sistema de reconocimiento de actividades. También se implementó un algoritmo que mejora el resultado del sistema de reconocimiento de actividades eliminando períodos cortos de determinada actividad (de unos pocos segundos de duración) que se deben a resultados erróneos del sistema de reconocimiento de eventos o no son relevantes para los investigadores. Se logró obtener un buen sistema de reconocimiento para las actividades de pastoreo y rumia pero no así para la actividad descanso. Teniendo en cuenta el alcance del proyecto y las condiciones en que fueron realizadas las grabaciones consideramos que el resultado obtenido es satisfactorio (porcentajes de acierto del 100 % para pastoreo y rumia; y 52 % para descanso). Se podrían obtener resultados más confiables modificando la ubicación del micrófono y realizando un nuevo modelo (etiquetado de las señales, identificación de cantidad de subestados por eventos e inicialización y entrenamiento) para todos los eventos.

Se logró implementar exitosamente un paquete de software que se ejecuta en un PC y permite copiar y borrar datos de una unidad de almacenamiento, seleccionar los archivos de audio del bovino y día que se desea analizar, reconocer las actividades realizadas por el mismo (pastoreo, rumia y descanso); y observar el resultado del análisis deseado. Este programa permite además cambiar el nombre

y/o formatear cualquier unidad de almacenamiento.

Para el desarrollo del proyecto se utilizaron herramientas de cross compilación para generar el SO y los ejecutables. Además fue necesario aprender a programar en diversos lenguajes tales como shell scripting, Ansi C, Java y Octave; y familiarizarse con el funcionamiento del HTK.

Cabe destacar que se trató de un proyecto multidisciplinario que brinda una solución integral al problema planteado por los investigadores del área de producción animal y pasturas.

La correcta finalización de este proyecto no hubiera sido posible de no haber trabajado en equipo y de no haber dividido y delegado tareas. Si bien los criterios que se tomaron para asignar las tareas realizadas por cada integrante del grupo fueron los conocimientos y afinidad de cada uno; todos nos interiorizamos con las tareas que estaba realizando el resto del grupo.

Capítulo 14

Perspectivas

A continuación se describen mejoras que se le podrían realizar tanto al dispositivo, al análisis de las señales y a la aplicación cliente.

También se listan posibles agregados al sistema para aumentar las funcionalidades del mismo.

14.1. Sistema embebido

14.1.1. Single Board Computer

Debido a que el desarrollo de las SBC se encuentra en pleno auge y han pasado varios meses desde que se seleccionó la SBC que utilizamos para la realización de este proyecto, ya se encuentran a la venta otras soluciones de menor consumo, tamaño y precio. La gran candidata a sustituir la TS-7260 es la placa **FOX Board G20 de ACME Systems**. Las características más destacables de la FOX Board G20 son su consumo (60mA @ 5V) y tamaño (66 x 72 mm); y además su diseño está basado en la arquitectura ARM al igual que la placa TS-7260, lo cual facilita bastante la adaptación del software del sistema embebido.

14.1.2. Baterías

Sin lugar a dudas las baterías son los elementos que más peso aportan al dispositivo. Una de las mejoras posibles es sustituir las mismas por otras que utilicen un

principio químico distinto que les permite aumentar la relación potencia entregada por kg.

En la Tabla 14.1.2 se muestran algunas características de las baterías que existen en el mercado.

Tipo	Energía/peso (Wh/kg)	Tensión por celda (V)	Tiempo de carga (h)	Auto-descarga por mes (% del total)
Gel	30-40	2,1	2-4	3-20
Plomo	30-50	2	8-16	5
Ni-Cd	48-80	1,25	10-14	30
Ni-Mh	60-120	1,25	2-4	20
Li-ion	110-160	3,16	2-4	25
Li-Po	100-130	3,7	1-1,5	10

Cuadro 14.1: Tabla comparativa de características de las baterías según el principio químico que utilizan.

14.1.3. Micrófono

Ubicación

Actualmente el micrófono está ubicado por encima del hocico del animal, sin embargo se podría pensar en colocarlo en la frente del mismo. En lugar de colocarlo en el bozal, pegarlo directamente al animal ya que además de registrar los sonidos emitidos también se registrarían las vibraciones a través del hueso.

Hay que tener en cuenta que en este caso se debería realizar nuevamente el proceso de determinación del sistema de reconocimiento de eventos.

Modelo

En este proyecto se decidió utilizar el micrófono **Ht-101** de la marca Xtreme que se encuentra en el mercado local a un precio accesible (5 USD). El mismo fue adaptado para permanecer a la intemperie sin problemas, se integró un pasa cable estanco a la caja y se recubre el micrófono con latex para protegerlo del

agua y polvo. A pesar de que el micrófono seleccionado funciona perfectamente creemos que existen mejores opciones, como por ejemplo el micrófono **AV-JEFES AVL63435**[24]. El mismo no se encuentra en el mercado local y su costo es elevado (94 USD en origen). Las principales diferencia entre el micrófono Xtreme Ht-101 y el AV-JEFES AVL63435 son que el último posee una mayor sensibilidad, trabaja en un mayor rango de operación y es a prueba de agua y polvo tanto para interior como exterior.

En caso de modificar el modelo del micrófono también es necesario realizar nuevamente el proceso de determinación del sistema de reconocimiento de eventos y evaluar si es signifactiva la mejora obtenida utilizando un micrófono de un costo tan elevado.

14.1.4. Técnicas de bajo consumo

Para reducir el consumo de la SBC se podrían aplicar técnicas de bajo consumo como son: cambiar procesos que utilizan polling por manejo de interrupciones, reducción de la frecuencia del procesador y utilizar modo sleep del procesador.

14.2. Aplicación cliente

14.2.1. Observación de datos

Actualmente solamente se muestran las actividades realizadas y en qué momento se realizaron. Para comprobar que el análisis de las señales fue realizado correctamente sería de gran utilidad que se permita reproducir el audio y se muestre su espectrograma.

Habría que estudiar otras técnicas de manejo de datos para poder implementar una solución e incluso existe la posibilidad de utilizar otro lenguaje de programación.

14.2.2. Sonidos indeseados

Al día de hoy no es posible filtrar los sonidos de pájaros, terneros, voz humana, tractores, etc. Sería muy útil que la aplicación cliente permita eliminar sectores de

las señales que no brindan información para los investigadores.

14.3. Extras

14.3.1. Evaluación del desempeño

La mejor forma de evaluar el desempeño de nuestro prototipo y validarlo, es mediante la comparación del mismo contra el dispositivo IGER y la inspección visual del animal.

14.3.2. Estimación de la materia seca consumida

Para estimar la materia seca consumida por el animal existen varias opciones:

- Analizando la energía de la señal. Cuanto más seca es la pastura mayor energía tiene la señal al realizar la actividad de pastoreo.
- Mediante el uso de GPS. En caso de que agregáramos un GPS a nuestro diseño se podría determinar la ubicación del animal al realizar cada actividad. Además, mediante el uso de mapas se podría determinar qué tipo de pastura ingirió el animal y estimar la cantidad de materia seca consumida.

14.3.3. Análisis video-acústico

Se podría agregar al dispositivo una cámara que registrara los movimientos realizados por el animal durante la ingesta de pastura. Haciendo corresponder el video y sonido obtenido se podría analizar:

1. Si el animal muerde en determinado lugar de manera aleatoria o sistemática, y qué tan lejos se encuentra una mordida de otra[25][26].
2. Cómo están sincronizados los movimientos y mordidas y cómo influye la distancia entre pequeños parches de pasturas[25][26].
3. Cómo se sincronizan los pasos y las mordidas y la cantidad de mordidas por paso en pasturas continuas[25][26].

Parte VIII

Anexos

Apéndice A

Sistema Embebido

A.1. Acciones TS-7260

Esto se debe realizar sino se corre GNU/Linux desde la tarjeta SD.

1. Para montar los USB's se debe realizar las siguientes secuencias.
   a) `/usr/bin/loadUSBModules.sh`
   b) Montar USB: `mount /dev/scsi/host0/bus0/target0/lun0/disc /mnt`
2. Montar SD Card
   a) `mount /dev/sdcard1/disc0/disc /mnt/`
   b) De ser necesario instalar el modulo `sdcard.o`
   c) insmod `/lib/modules/2.4.26-ts11/kernel/drivers/block/sdcard.o`
3. Para hacer andar las tarjetas SD HC se debe realizar la siguiente secuencia.
   a) `rmmod /lib/modules/2.4.26-ts11/kernel/drivers/block/sdcard.o`
   b) `wget ftp.embeddedarm.com/ts-arm-sbc/ts-7260-linux/binaries/ts-modules/sdcard-sdhc.o`
   c) `mv sdcard-sdhc.o /lib/modules/`
   d) `insmod /lib/modules/sdcard-sdhc.o`

A.2. Crear SD booteable

1. Se inicia dandole formato ext2 a la tarjeta sd.
   a) Ejecutamos `mkfs.ext2 /dev/mmcblk`
   b) De no poderse debido a que esta montado ejecutar `umount /dev/mmcblk*` (Desmonta todas las particiones)
2. Copiamos la imágen que nos brindan en la página [17].
   a) Ejecutamos `dd if=sdimage.dd of=/dev/mmcblk`
3. Copiamos la imagen del disco de arranque.
   a) Ejecutamos `dd if=ts7260-fb-default of=/dev/mmcblkp1`
4. Para redimencionar la tercera partición realizamos la siguiente secuencia.
   a) `fdisk /dev/mmcblk`
   b) Tecleamos `p`, nos muestra las particiones de ls SD.
   c) Para borrar la tercera partición digitamos `d`.
   d) Nos va a pedir el número de partición, ingresamos `3`.
   e) Volvemos a digitar `p`, vemos que ya no tenemos la partición 3.
   f) Para crear la partición tecleamos `n`.
   g) Luego digitamos `p` para que sea una partición primaria.
   h) Nos va a dar a elejir cual es el primer cilindro, dejamos el primero.
   i) Seguido nos pide el último elegimos el último cilindro.
   j) Digitamos `p`, para ver las particiones de ser la que deseamos pasamos al siguiente paso sino volver al paso 3.
   k) Tecleamos `w`, para grabar las particiones en la SD.
5. Por último instalaremos el tsbootrom-update.

a) Copiar `tsbootrom-update` a una SD card, montarla y copiar tsbootrom-update en algún directorio (ej.: /tmp)

b) Nos ubicamos en el directorio donde copiamos tsbootrom-update.

c) Ejectamos `ifconfig`, y copiamos en número de MAC. Ellos dan la siguiente opción |mac=‘ifconfig eth0 | grep HWaddr | cut -d’ ’ -f11‘| (la cual tiene en la placa que usamos daba un error al buscar la mac).

d) Ejecutamos `./tsbootrom-update -s -m MAC` (MAC la copiamos en el paso anterior), sino ellos recomiendar `tsbootrom-update -s -m $mac` (esto no se ejecuta; se deberia hacer `./tsbootrom-update -s -m $mac`).

A.3. Cross compilar kernel

Para cross compilar se necesitan los fuentes del kernel y los fuentes de cross compilador. En nuestro caso vamos a usar el `tskernel-2.4.26-ts10-src.tar.gz` y la herramienta de cross compilación `arm-2009q3-67-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2`.
Ambos se encuentran disponibles en el ftp de TS.

Pasos a seguir:

1. Descomprimimos los fuentes del kernel `$ tar xvfz tskernel-2.4.26-ts10-src.tar.gz`

2. Descomprimimos los fuentes de la herramienta de cross compilación `$ tar xvfj arm-2009q3-67-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2`

3. Vamos a modificar el archivo Makefile que se encuentra en la carpeta que descomprimimos el kernel, en este caso linux24. La herramienta de cross compilación se descomprimio en la carpeta `usr`.

   Ejecutamos: `$ cd linux24` luego ejecutamos `$ gedit Makefile`

4. Cambiamos la siguiente variable `CROSS_COMPILE` la cual tiene que apuntar al directorio de cross compilación.

En este caso la cambiamos de la siguiente forma: `CROSS_COMPILE = $(shell cd $(TOPDIR)/../usr/local/opt/crosstool/arm-linux/gcc-3.3.4-glibc-2.3.2/bin/ && pwd)/arm-linux-`

5. Guardamos los cambios y pasamos a ejecutar la siguiente secuencia:

   a) `$ make mrproper`

   b) `$ make ts7300_config` [1]

   c) `$ make menuconfig`[2]

   d) `$ make dep`

   e) `$ make zImage`

   f) `$ make modules`

   g) `$ make modules_install`[3]

6. Pasaremos a instalar lo que hemos creado en la tarjeta SD.

   a) `$ cd arch/arm/boot/`

   b) `dd if=zImage of=/dev/mmcblkp1` [4]

   c) `$ cd /lib/modules/`

   d) `$ cp -r 2.4.26-ts10 /media/disk-1/lib/modules/`[5]

[1] No elegir ts7260_config, debido a que no bootea desde la SD; hay que realizarle cambios en el archivo .config)

[2] Elegimos los paquetes, módulos, librerias, programas, etc que se necesiten.

[3] Deber ser ejecutado como root.

[4] Grabamos el la imágen comprimida en la primera partición de la tarjeta SD

[5] La tarjeta puede ser montanda en otra dirección, hay que verificar donde se monto la tercera partición.

A.4. Implementación radio frecuencia

Las características que se tuvieron en cuenta para la elección de la tarjeta inalámbrica fueron que:

- La velocidad de transmisión sea mayor a 54 Mbps.
- Soporte modalidad Ad-hoc.
- Sea compatible con GNU/Linux 2.4.x

La primera opción que se consideró fue la tarjeta **ENCORE ENUWI-SG**[27], que además de cumplir con las condiciones antes mencionadas se encuentra en las listas de tarjetas Wireless se compatible con GNU/Linux. Además dicha tarjeta se encuentra en el mercado uruguayo a precios razonables.

Se intentó instalar esta tarjeta en la SBC y una computadora con GNU/Linux de las siguientes formas:

- Emulando los drivers de Windows mediante la aplicación ndiswrapper. Esta solución no es viable para una arquitectura distinta de la X86.
- Utilizando madwifi la cual es soportada por las arquitecturas ARM, MIPS y X86. La misma implementa un driver para el chipset AR5523, el cual posee la tarjeta wireless previamente mencionada. El problema de esta solución es que se implementa para interfaz PCI, no USB.
- Se compiló correctamente para nuestra arquitectura el driver suministrado por el proyecto ar5523libusb[28] (que utiliza interfaz USB mediante el uso de la biblioteca libusb[29] y el chipset AR5523). El problema fue que el driver compilado generaba errores de direccionamiento de memoria debido a que para utilizar el mismo eran necesarios 512 MBytes de memoria.

Por estos motivos se descartó la utilización de la tarjeta Wireless ENCORE ENUWI-SG.

La segunda opción considerada se seleccionó en base a los requisitos mencionados previamente, que sea de bajo consumo, que sea compatible con un microprocesador de 200Mhz, que se pueda instalar en una arquitectura ARM y en base a los

chipset que soportan los proyectos Open Source: Serial Monkey [30], WLAN-NG [31] y USB WLAN[32].

La tarjeta Wireless IOGear GWU523, posee un chipset ZD1211 el cual es soportado por el proyecto WLAN USB. Soporta modalidad Ad-Hoc, es compatible con GNU/Linux, transmite a 54Mbps, consume hasta 350 mA durante la transmisión y no exige una velocidad de procesador mínima para su funcionamiento.

Dicha tarjeta USB Wireless fue instalada y probada con éxito en la SBC, luego de haber cross compilado el driver y configurado la interfaz de red.

Para determinar si el uso de radio frecuencia era viable en cuanto a consumo y tiempo de transmisión se realizaron varias pruebas:

- Utilizando una red Ad-Hoc punto a punto se fue variando el tamaño de los paquetes que se transmitían y finalmente la mejor velocidad de transmisión que se logró fue de 2Mbps.

- Utilizando un router como modo de acceso. La mejor velocidad de transmisión se logró con un ancho 1512 byte y fue de 1 Mbps.

- Modalidad managed Ad-Hoc. Su resultado fue bastante similar al de la red Ad-Hoc.

Teniendo en cuenta que se necesitan transmitir 8 GB junto con los encabezados de cada paquete se calculó que para transmitir la totalidad de los datos eran necesarias 8 horas; y considerando que el consumo de la placa en estas condiciones se encontraba en el entorno de 3W se decidió no utilizar esta tecnología aún cuando ya estaba implementada.

Finalmente, para recolectar los datos almacenados en el pendrive el usuario deberá retirar el mismo y conectarlo en un PC.

A.5. Apagado del sistema

Debido a que la SBC seguia consumiendo a pesar que el sistema operativo ya se había apagado, es que se decidió realizar un sistema de apagado de la SBC.

Para el diseño se tuvieron en cuenta varios componentes y configuraciones.

Las opciones que se consideraron fueron:

- Interruptor.
- Opto acoplador con un contador de tiempo y regulador de tensión.
- Opto acoplador, transistores y capacitor.

A.5.1. Interruptor

Esta opción consiste en agregar un interruptor dejando en manos del usuario el encendido y correcto apagado de la SBC. Esta opción a pesar de ser la más sencilla y de menor costo se descartó debido a que no solo el usuario debe ser consciente del método para apagar correctamente la SBC sino que también posee algunos inconvenientes[6].

A.5.2. Opto acoplador con un contador de tiempo y regulador de tensión

Otra alternativa es diseñar un sistema con un contador de tiempo (NE555) y un opto acoplador (CYN17)[7]. Este diseño se basó en una comunicación a través del puerto de entrada/salida DIO1, el mismo daba un pulso al contador de tiempo mientras el sistema operativo se encontrara activo, alimentando el opto acoplador dejando pasar la corriente desde la batería a la SBC. Se encontró que se debía colocar otro transistor a la salida del opto acoplador para aumentar la corriente que se debía entregar a la SBC, para lo que se utilizó una configuración Darlington entre la salida del opto acoplador y el nuevo transistor.

[6]Por ejemplo, si la memoria ya está llena, el software del sistema embebido manda apagar el sistema operativo pero la SBC seguiría consumiendo corriente hasta que el usuario vaya a apagarla correctamente, esto generaría un desgaste innecesario de las baterias acortando la vida útil de las mismas.

[7]Estos componentes son de bajo costo y se consiguen en plaza.

Esta configuración presenta varios problemas, por ejemplo si no se usa alimentación de 5 V, se debe agregar un regulador de tensión (KA7805) para manejar la entrada del opto acoplador y no dañar dicho integrado. Además del regulador de tensión se agregó un capacitor para mantener estable la tensión y los intergrados activos durante algún tiempo luego de que se mando apagar el sistema.

A pesar de haber funcionado correctamente, esta solución fue descartada porque el consumo del regulador de tensión aumentaba significativamente el consumo del sistema (un 40 % más del consumo del sistema orginal).

A.5.3. Opto acoplador, transistores y capacitor

Debido a que el puerto de DIO1 no entrega la suficiente corriente como para manejar el opto acoplador directamente se agregó un transistor con su colector conectado directamente al borne positivo de la batería, aumentando así la corriente que se le entrega al opto acoplador. La salida del opto acoplador se conecta en configuración Darlington (para aumentar la corriente que se le entrega a la SBC, como en la opción anterior) y se agrega un capacitor entre los bornes de alimentación de la SBC para tener una tensión estable y poder mantenerla alimentada unos segundos con la energía almacenada en el mismo, para poder terminar de apagar correctamente el sistema operativo[8].

Se decidió utilizar este diseño ya que consume sólo un 5 % más que el sistema original y brinda beneficios al usuario.

[8] Evitando que en el próximo arranque realice un file system check.

Apéndice B

Análisis de la señal

B.1. HTK Linux

Fuente: http://htk.eng.cam.ac.uk
IMPORTANTE! Instalar `libx11-dev` (synaptic)

1. Descargar: `HTK-3.4.1.tar.gz` (http://htk.eng.cam.ac.uk/ftp/software/HTK-3.4.1.tar.gz)
2. Descomprimirlo queda en el directorio donde se encuentra el archivo .tar.gz y crea el directorio `htk`(tar xzf HTK-3.4.1.tar.gz)
3. Ir a el directorio `htk`
4. Ejecutar `./configure --prefix=/tmp/` Este comando verifica que las dependencias necesarias para su instalación se encuentren instaladas. Es conveniente descomprimirlo en`/usr/bin` pues de esta manera luego se podra ejecutar desde cualquier directorio del sistema.
5. Si no hubo ningún error de dependencias ejecutar el comando `make all`.
6. Ejecutar el comando `make install`.
7. Ejecutar `cp /tmp/bin/* /usr/bin/`

Para verificar que quedo instalado correctamente:

1. Descargar `HTK-samples-3.4.1.tar.gz` (http://htk.eng.cam.ac.uk/ftp/software/HTK-samples-3.4.1.tar.gz)
2. Descomprimirlo (No es necesario descomprimirlo en un directorio particular).
3. Ir a la carpeta `HTKDemo`
4. Ejecutar `mkdir -p hmms{tmp,hmm.{0,1,2,3}} proto acc test` Si existe `tmp,hmm.{0,1,2,3}` crea los siguientes directorios: `proto acc test`
5. Ejecutar `perl runDemo configs/monPlainM1S1.dcf`

Apéndice C

Funciones del HTK

A continuación se detallan las funciones utilizadas del HTK.

C.1. HSLab

Utiliza la librería gráfica HGraf para desplegar una una interfaz usuario que permite seleccionar una zona del archivo de audio y asignarle una etiqueta. Una vez etiquetado las zonas de interes, se genera un archivo de texto .lab indicando el inicio, fin y etiqueta.

C.2. HCopy

Esta función copia uno o más archivos, a otro archivo de salida designado; con la opción de convertir los datos originales a datos paramétricos (dependiendo de los parámetros establecidos en el archivo de configuración).

C.3. HInit

Se utiliza para proporcionar una estamición inicial de los parámetros de un HMM a partir de una secuencia de observaciones. Trabaja usando el algoritmos de *Viterbi*[33] repetidas veces.

Normalmente utiliza como archivo de entrada un archivo que define el prototipo

del HMM y a partir de él toma la topología del mismo pero se ignoran los valores de media y varianza.

C.4. HRest

Implementa la reestimación de los parámetros de un HMM (inicialmente estimados con la función HInit) utilizando secuencias de observación y el algoritmo de *Baum-Welch*[33].

C.5. HParse

A partir de un archivo de texto donde se define la gramática (interacción entre los modelos) genera un archivo con extensión slf que representa la interacción entre los estados (modelos) a más bajo nivel que el archivo de gramática.

Al archivo generado lo denominamos red y es el que se utliza para invocar el reconocimiento de eventos (utilizando *HVite*)

C.6. HVite

A partir de los parámetros de un archivo de audio (características), el archivo de la red, los archivos que contienen los parámetros entrenados y el archivo de diccionario, retorna un archivo de texto en el que se detalla el inicio y fin de la sucesión de eventos que se encuentran en el archivo de audio.

Apéndice D

Aplicación cliente

D.1. Especificación de requerimientos

D.1.1. Casos de uso

En la figura D.1 se pueden observar los casos de uso de la aplicación cliente los cuales se detallan en las siguientes secciones.

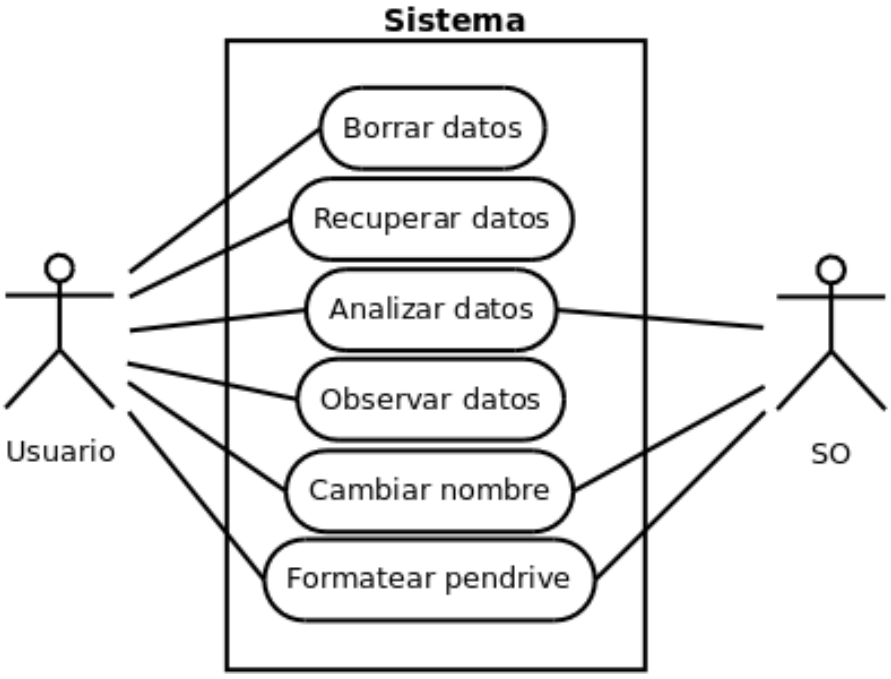


Figura D.1: Casos de uso

Borrar datos

Nombre:		Borrar datos	
Id:		br_datos	
Actores:		Usuario del sistema	
Descripción:		El usuario del sistema borra los datos.	
Precondiciones:		El sistema se encuentra en funcionamiento y correctamente conectado.	
Poscondiciones:		Curso básico: El usuario borra los datos. Curso alternativo: El usuario no borrará.	
Curso básico			
Usuario		**Sistema**	
1.	El usuario del sistema elige la opción de borrado.		
		2.	El sistema solicita confirmación.
3.	El usuario confirma que desea borrar.		
		4.	El sistema borra los datos
		5.	El sistema confirma que los datos han sido borrados
		6.	Fin de caso de uso.
Curso alternativo			
3.1	El usuario desea cancelar.		
		3.2	El sistema cancela la operación.
		3.3	Fin de caso de uso.

Recuperar datos

Nombre:	Recuperar datos
Id:	tr_datos
Actores:	Usuario del sistema
Descripción:	El usuario del sistema recupera los datos.
Precondiciones:	El sistema se encuentra correctamente conectado.
Poscondiciones:	Curso básico: El usuario solicita recuperar los datos y luego borra los mismos. Curso alternativo: El usuario solicita recupera los datos pero no los borra.

Curso básico

Usuario		**Sistema**	
1.	El usuario elige la opción de recuperar datos.		
		2.	El sistema le solicita al usuario que datos desea recuperar.
3.	El usuario selecciona los datos a recuperar.		
		4.	El sistema recupera los datos.
		5.	El sistema le informa al usuario que los datos han sido recuperados exitosamente.
		6.	El sistema pregunta al usuario si desea borrar los datos recuperados.
7.	El usuario confirma que desea borrar los datos		
		8.	El sistema llama a br_datos.
		9.	Fin de caso de uso.

Curso alternativo			
7.1	El usuario no desea borrar los datos		
		7.2	El sistema cancela la opercación
		7.3	Fin de caso de uso.

Análizar datos

Nombre:	Analizar datos
Id:	analisis_datos
Actores:	Usuario del sistema
Descripción:	El usuario elige analizar datos.
Precondiciones:	Los datos que se desean analizar fueron previamente recuperados.
Poscondiciones:	Curso básico: El usuario solicita analizar los datos. Curso alternativo 1: El usuario no selecciona los datos a visualizar. Curso alternativo 2: El usuario no desea visualizar los datos.

Curso básico			
Usuario		**Sistema**	
1.	Un usuario elige la opción analizar los datos.		
		2.	El sistema solicita al usuario los datos a analizar.
3.	El usuario ingresa los datos solicitados.		
		4.	El sistema decima, filtra, corta y analiza los datos.
		5.	El sistema confirma que los datos fueron analizados correctamente.

		6.	El sistema pregunta si desea visualizar los datos analizados.
7.	El usuario confirma que desea visualizar los datos analizados.		
		8.	El sistema llama a vis_datos.
		9.	Fin de caso de uso.
Curso alternativo 1			
3.1	El usuario cancela.		
		3.2	El sistema canela la operación.
		3.3	Fin de caso de uso.
Curso alternativo 2			
7.1	El usuario cancela.		
		7.2	El sistema canela la operación.
		7.3	Fin de caso de uso.

Visualizar análisis de sonido

Nombre:	Visualización del análisis del sonido
Id:	vis_datos
Actores:	Usuario del sistema
Descripción:	El usuario elige observar los datos recuperados.
Precondiciones:	Los datos que se desean visualizar fueron previamente analizados.
Poscondiciones:	Curso básico: El usuario solicita visualizar los datos. Curso alternativo: El usuario no selecciona los datos a visualizar.
Curso básico	
Usuario	**Sistema**

1.	Un usuario elige la opción visualizar los datos.		
		2.	El sistema solicita al usuario los datos a visualizar.
3.	El usuario ingresa los datos solicitados.		
		4.	El sistema despliega los datos.
		5.	Fin de caso de uso.
Curso alternativo			
3.1	El usuario cancela.		
		3.2	El sistema canela la operación.
		3.3	Fin de caso de uso.

Cambiar nombre pendrive

Nombre:	Cambiar nombre
Id:	cambiar_nombre
Actores:	Usuario del sistema
Descripción:	El usuario elige cambiar nombre al pendrive.
Precondiciones:	El sistema se encuentra correctamente conectado.
Poscondiciones:	Curso básico: El usuario solicita cambiar nombre al pendrive. Curso alternativo: El usuario cancela.

Curso básico			
Usuario		**Sistema**	
1.	Un usuario elige la opción cambiar nombre al pendrive.		

		2.	El sistema solicita al usuario el pendrive que desea cambiar el nombre.
3.	El usuario selecciona el pendrive.		
		4.	El sistema solicita al usuario el nombre nuevo que le desea asignar al pendrive.
5.	El usuario ingresa el nombre nuevo.		
		6.	El sistema confirma que el nombre del pendrive se cambió con exito.
		7.	Fin de caso de uso.
Curso alternativo			
3.1	El usuario cancela.		
		3.2	El sistema canela la operación.
		3.3	Fin de caso de uso.
Curso alternativo			
5.1	El usuario cancela.		
		5.2	El sistema canela la operación.
		5.3	Fin de caso de uso.

Formatear pendrive

Nombre:	Formatear nombre
Id:	formatear
Actores:	Usuario del sistema
Descripción:	El usuario elige formatear pendrive.
Precondiciones:	El sistema se encuentra correctamente conectado.
Poscondiciones:	Curso básico: El usuario solicita formatear el pendrive.

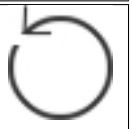
		Curso alternativo: El usuario cancela.	
Curso básico			
Usuario		**Sistema**	
1.	Un usuario elige la opción formatear pendrive.		
		2.	El sistema solicita al usuario el pendrive q desea formatear.
3.	El usuario selecciona el pendrive.		
		4.	El sistema solicita al usuario el nombre que le desea asignar al pendrive.
5.	El usuario ingresa el nombre.		
		6.	El sistema formatea el pendrive y le confirma al usuario.
		7.	Fin de caso de uso.
Curso alternativo			
3.1	El usuario cancela.		
		3.2	El sistema canela la operación.
		3.3	Fin de caso de uso.
Curso alternativo			
5.1	El usuario cancela.		
		5.2	El sistema canela la operación.
		5.3	Fin de caso de uso.

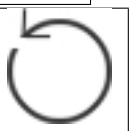
D.1.2. Tarjetas CRC

Clase:	ConectorShell	
Descripción:	Es el encargado de realizar solicitar la ejecucion de comandos en el sistema operativo.	
Responsabilidades:		**Colaboraciones:**
Ejecutar comando		

Clase:	CambiarNombre	
Descripción:	Es el encargado de solicitar nuevo nombre.	
Responsabilidades:	**Colaboraciones:**	
Obtener nombre nuevo		

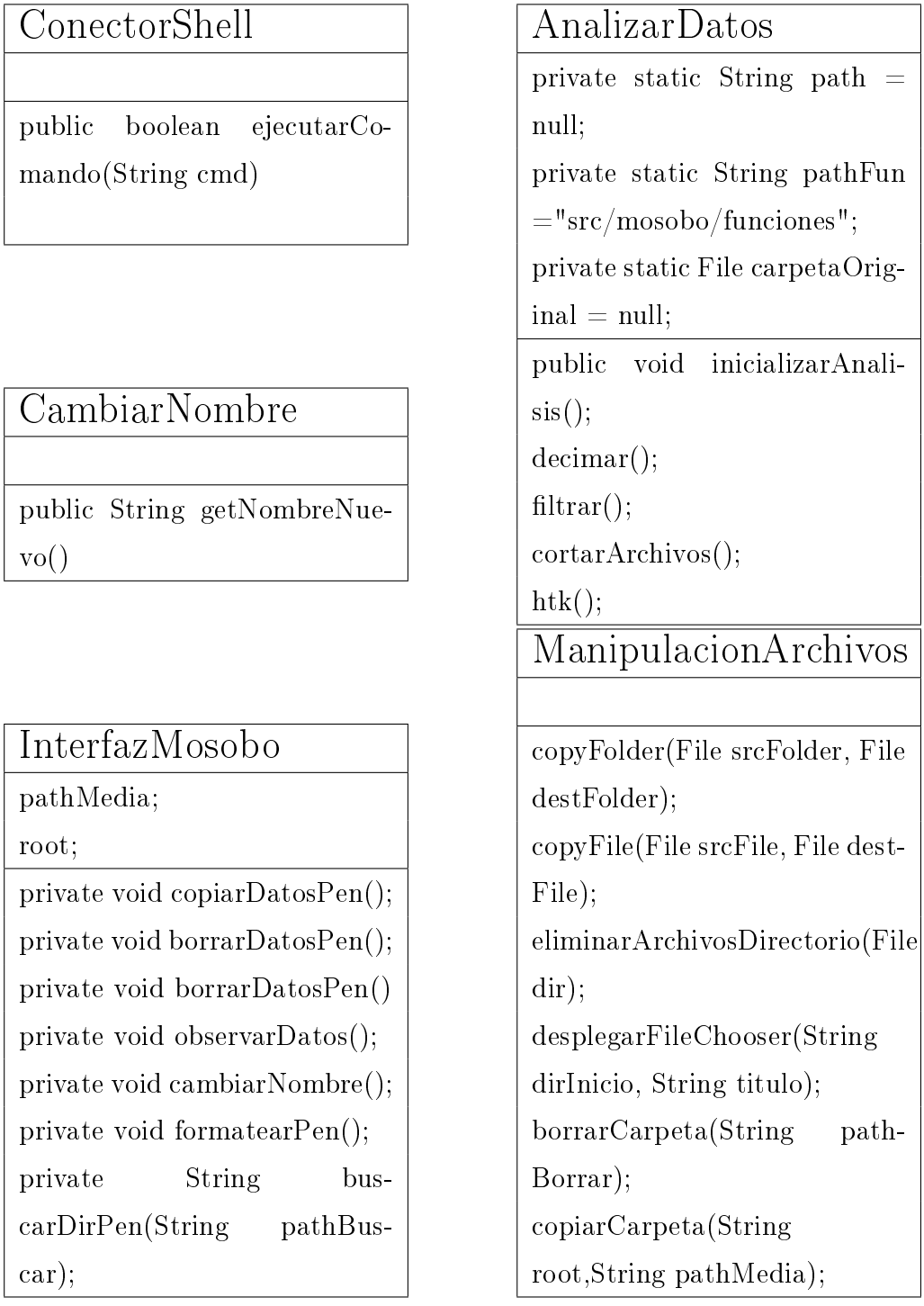
Clase:	InterfazMosobo	
Descripción:	Interfaz de comunicación con el usuario.	
Responsabilidades:	**Colaboraciones:**	
Borrar datos	ManipularArchivos	
Copiar datos	ManipularArchivos	
Analizar datos	AnalizarDatos	
Observar datos	Datos	
Formatear	ConectorShell	
Cambiar nombre	ConectorShell	

Clase:	ManipularArchivos	
Descripción:	Encargada de eliminar/copiar archivos y directorios.	
Responsabilidades:	**Colaboraciones:**	
Copiar directorio Copiar directorio		

Clase:	AnalizarDatos	
Descripción:	Se encarga de decimar, filtrar, cortar y analizar los datos.	
Responsabilidades:	**Colaboraciones:**	
Decimar datos	ConectorShell	
Filtrar datos	ConectorShell	
Cortar datos	ConectorShell	
Analizar datos	ConectorShell	

D.2. Diseño

D.2.1. Diagrama de clases



Apéndice E

Lista de compras del Hardware

Los principales componentes para realizar el diseño del sistema embebido son: la placa TS-7260, baterías, caja contra polvo, agua y golpes; micrófono, tarjeta de audio y memorias de almacenamiento. La placa TS-7260 y la caja se compran en Estados Unidos mientras que los demás componentes se consiguen en el mercado local.

En las siguientes secciones se detallan todos los componentes que fueron necesarios para desarrollar el sistema operativo y gastos.

E.1. Componentes para ensayos

En la Tabla E.1 se muestra el costo de los componentes que se compraron para realizar los ensayos preliminares. Algunos de los componentes necesarios para realizar los ensayoes fueron micrófonos, cassettes, etc.

Concepto	Cantidad	Precio unitario (USD)	Sub Total (USD)
Componentes para ensayos	NC	NC	28
Total: USD 28			

Cuadro E.1: Gasto de los componentes para realizar los ensayos preliminares

E.2. SBC TS-7260 y accesorios on-board

La placa TS-7260 se compró en *Technologic Systems*[34]. En la Tabla E.2 se detalla los precios de la placa con 32 MB de RAM y Flash, los componentes onboard que se agregarán, como son: el real time clock, el socket para la tarjeta SD, sensor de temperatura y el kit de desarrollo, así como también la cantidad comprada.

Concepto	Cantidad	Precio unitario (USD)	Sub Total (USD)	Comentarios
TS-7260-64-128F	2	179,00	358,00	TS−7260 SBC with 32 MB RAM and 32 MB Flash.
OP-SDSOCKET	2	8,00	16,00	Sockets de SD Card.
OP-BBRTC	2	10,00	20,00	Real time clock.
OP-ROHS-NC	2	0,00	0,00	Para no dañar el medio ambiente.
OP-TMPSENSE	2	3,00	6,00	Opcional.
KIT-7260	1	100,00	100,00	Kit de desarrollo.
			Total: USD 500	

Cuadro E.2: Detalle de la placa, de los accesorios on-board y del kit de desarrollo comprado

E.3. Tarjetas inalámbricas, de audio y almacenamiento

Para implementar la transmición de datos mediante radio frecuencia se compraron 2 tarjetas inalámbricas, una de ellas se compró en *Amazon*, Estados Unidos. La otra tarjeta inalámbrica, las tarjetas de audio y unidades de almacenamiento

se compraron en el mercado local (Ver Tablas E.3 y E.4).

Concepto	Cantidad	Precio unitario (USD)	Sub Total (USD)
iogear gwu-523	1	40	40
ENCORE ENUWI-G	1	35	35
Tarjeta audio 3D Sound	2	3	6
Total: USD 81			

Cuadro E.3: Gastos de las tarjetas inalámbricas y tarjetas de audio.

Concepto	Cantidad	Precio unitario ($)	Sub Total ($)
Pendrive Kingston USB 2.0 8GB	2	587	1174
Tarjeta SD Kingston 4GB	2	370	740
Total: $ 1914			

Cuadro E.4: Gastos de las unidades de almacenamiento.

E.4. Batería

Se compraran 4 baterías de gel de 6V 3.2A. De esta manera, una vez al día se obtendrían los datos almacenados transmitiéndolos por radio frecuencia y se cambia la batería, por lo que podríamos realizar la adquisición de audio prácticamente las 24 horas al día durante varios días consecutivos. Los detalles se pueden observar en la Tabla E.5.

Concepto	Cantidad	Precio unitario ($)	Sub Total ($)
Batería de gel 6V, 3.2A	4	355	1420
Cargador para baterías de gel	1	210	210
Total: $ 1630			

Cuadro E.5: Batería y cargador comprados.

E.5. Caja, bozal e interfaz usuario

La caja, el conector de micrófono tienen que ser estancos porque el dispositivo se va a encontrar a la intemperie junto con el animal. Los leds, switches van a estar dentro de la caja por lo que no seleccionamos que fueran estancos.

La caja se compró en *Pelican*[16], mientras que el conector estanco para el micrófono, leds, switches y demás en *Eneka*, *Mundo Electrónico* y ferreterias. Detalles de la se pueden observar en las Tablas E.6 y E.7.

Concepto	Cantidad	Precio unitario (USD)	Sub Total (USD)
Pelican 1060 Micro Case	2	16,24	32,48
Total: USD 32,48			

Cuadro E.6: Caja comprada.

Conceptpo	Cantidad	Precio unitario ($)	Sub Total ($)
Componentes varios para interfaz usuario y acondicionamiento de la caja	NC	NC	1000
PCB placa interfaz y control de energía	2	300	600
Acondicionamiento de la caja	NC	NC	600
		Total:	$ 2200

Cuadro E.7: Componentes para la interfaz usuario, bozal y pasacables para el micrófono estanco.

E.6. Micrófono

Concepto	Cantidad	Precio unitario ($)	Sub Total ($)
Micrófono xTreme	2	100	200
		Total:	$ 200

E.7. Envíos y aduana

Concepto	Cantidad	Precio unitario ($)	Sub Total ($)
Aduana - cajas	1	764	764
		Total:	$ 764

Concepto	Cantidad	Precio unitario (USD)	Sub Total (USD)
Envío - cajas	1	32	32
Envío - placas	1	22	22
Total: USD 54			

E.8. Gastos totales

En la siguiente tabla se resumen todos los gastos realizados durante el desarrollo del proyecto. No se consideran las tarjetas inalámbricas ya que finalmente no fueron utilizadas. En total se gastó USD 969.

Concepto	Cantidad	Importe (USD)
Componentes para ensayos	NC	28
SBC TS-7260	2	400
Kit desarrollo -7260	1	100
Tarjeta audio 3D Sound	2	6
Baterías	4	71
Cargador	1	11
1060 Micro case (caja)	2	33
PCB placa interfaz	2	30
Componentes placa interfaz	NC	50
Acondicionamiento de la caja	NC	30
Pendrive 8 GB	2	60
Tarjetas SD 2 GB	2	40
Micrófono	2	10
Envíos y aduanas	NC	100
Total		**969**

E.9. Costos para fabricar un dispositivo

En la siguiente tabla se detallan todos los componentes necesarios para la fabricación de un dispositivo. El costo de los materiales de un dispositivo es 375 dólares,

sin considerar los gastos de importación, impuestos y envíos. Cabe destacar que solamente la SBC y la caja deben ser importadas.

Concepto	Cantidad	Importe (USD)
SBC TS-7260	1	200*
Tarjeta audio 3D Sound	1	3
Baterías	2	35
Cargador	1	11
1060 Micro case (caja)	1	16*
PCB placa interfaz	1	15
Componentes placa interfaz	NC	25
Acondicionamiento de la caja	NC	15
Pendrive 8 GB	1	30
Tarjetas SD 2 GB	1	20
Micrófono	1	5
Total		**375**

Parte IX

Bibliografía

Bibliografía

[1] P. Chilibroste, P. Soca, D. Mattiauda, O. Bentancur, and P. Robinson., "Short term fasting as a tool to design effective grazing strategies for lactating dairy cattle: a review.," *Australian Journal of Experimental Agriculture*, 2007.

[2] J. Hodgson, "The control of herbage intake in the grazing ruminant.," *Hill Farming Reasearch Organization*, 1985.

[3] E. Sazonov, S. Schuckers, P. Lopez-Meyer, O. Makeyev, N. Sazonova, E. L. Melanson, and M. Neuman., "Non-invasive monitoring of chewing and swallowing for objective quantification of ingestive behavior.," *Electronic Journals from Institute of Physics Publishing.*, 2008.

[4] E. D. Ungara, N. Ravida, T. Zadaa, E. Ben-Moshea, R. Yonatana, H. Barama, and A. Genizib., "The implications of compound chew - bite jaw movements for bite rate in grazing cattle.," *Science Direct*, 2005.

[5] M. J. Gibb, C. A. Huckle, R. Nuthall, and A. J. Rook., "Effect of sward surface height on intake and grazing behaviour by lactating holstein friesian cows.," *Inter Science*, 2008.

[6] Orr, M. Gibb, and Robert., "Grazing behaviour of ruminants.," *Prifysgol Aberystwyth University*, 1997.

[7] D. Milone, H. Rufiner, J. Galli, E. L. f, and C. Cangiano, "Computational method for segmentation and classification of ingestive sounds in sheep," *Science Direct*, 2009.

[8] Galli, C. A. Cangiano, and J. R., "Super campo." Marzo 2009.

[9] J. Galli, C. Cangiano, M. Demment, and E. Laca., "Acoustic monitoring of chewing and intake of fresh and dry forages in steers.," *Animal Feed Science and Technology.*, 2005.

[10] Gumstix, "Gumstix developer site - gumstix overo - feature overview.." http://www.gumstix.net/Hardware/view/Hardware-Specifications/Overo-Specifications/112.html, Agosto 2009.

[11] Eurotech. http://www.eurotech.com/DLA/datasheets/Products_Eurotech /TITAN_sf.pdf, Agosto 2009.

[12] WinSystem. http://www.pc104plus.com/products/PPM−LX800−G.cfm, Agosto 2009.

[13] WinSystem. http://sbc.winsystems.com/products/EPX−GX500.cfm, Agosto 2009.

[14] T. Systems. http://www.embeddedarm.com/products/board−detail.php?product=TS−7260, Agosto 2009.

[15] BusyBox, Junio 2010.

[16] "Pelican products 1060 micro case." http://www.pelican.com/cases_detail.php?Case=1060, Setiembre 2009.

[17] T. System, "Ts−boot." http://www.embeddedarm.com/software/arm−linux−fastboot−ts7300.php, Noviembre 2009.

[18] "Hidden markov models." http://jedlik.phy.bme.hu/∼gerjanos/HMM/node2.html, Setiembre 2009.

[19] W.-H. Steeb, *The Nonlinear Workbook*. World Scientific Publishing Co. Pte. Ltd., 2005.

[20] "Linear predictive coding (lpc)." http://www.otolith.com/otolith/olt/lpc.html, Julio 2010.

[21] K. Tokuda, T. Kobayashi, and S. Imai, "Recursive calculation of mel cepstrum from lp coefficients.," 1994.

[22] U. de Cambridge, "htk3," Noviembre 2009.

[23] L. M. Garshol, "Bnf and ebnf: What are they and how do they work?." http://www.garshol.priv.no/download/text/bnf.html, Julio 2010.

[24] Ezprogear. http://www.ezprogear.com/index.php/av-jefe-avl-634-35-tan-headset-microphone.html, Setiembre 2009.

[25] "The agriculture research organization of israel.." Marzo 2009.

[26] W. M. Griffiths, V. Alchanatisb, R. Nitzana, V. Ostrovskyb, E. B. Moshea, R. Yonatana, S. Brenera, H. Barama, A. Genizic, and E. D. Ungar., "A video and acoustic methodology to map bite placement at the patch scale.," *Science Direct*, 2005.

[27] E. electronics. http://www.encore−usa.com/product_item.php?region=us&bid=2&pgid=20&pid=22#DOWNLOAD, Setiembre 2009.

[28] "Proyecto ar5523 with libusb." http://fatah.afraid.org/files/ar5523libusb−project/, Noviembre 2009.

[29] "Proyecto libusb." http://www.libusb.org/, Noviembre 2009.

[30] "Proyecto serial monkey." http://rt2x00.serialmonkey.com/wiki/index.php/Main_Page, Febrero 2010.

[31] "Wlan−ng." http://www.linux−wlan.com/linux−wlan/, Febrero 2010.

[32] "Usb wlan." http://sourceforge.net/projects/zd1211/develop, Febrero 2010.

[33] F. Jelinek, *Statistical Methods for Speech Recognition*. The MIT Press, 1997.

[34] T. System, "Ts-7260 ultra−low power arm9 single board computer for embedded systems."

http://www.embeddedarm.com/products/board-detail.php?
product =TS−7260#, Setiembre 2009.