UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EMBEBIDO PARA OBTENCIÓN DE IMÁGENES A TRAVÉS DE UNA RED IP

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA

SEBASTIÁN BAS KANA

PROFESOR GUÍA:
EYAL SHATS YUDILEVICH

MIEMBROS DE LA COMISIÓN:
FRANCISCO RIVERA SERRANO
MARCOS DÍAZ QUEZADA

SANTIAGO DE CHILE
2014

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELECTRICISTA
POR: SEBASTIÁN BAS KANA
FECHA: 21/10/2014
PROF. GUÍA: EYAL SHATS YUDILEVICH

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA EMBEBIDO PARA OBTENCIÓN DE IMÁGENES A TRAVÉS DE UNA RED IP

Los sistemas embebidos son probablemente las tecnologías más masivas que existen hoy en día. Prácticamente todo equipo tecnológico cuenta con un sistema embebido. En esto radica la importancia de esta memoria. Se pretende mostrar cómo funciona el proceso de crear un prototipo de sistema embebido que permita probar un concepto o una idea.

En este caso particular, el sistema que se diseñó e implementó tiene el objetivo de solicitar y obtener imágenes de forma remota. Además, el proyecto se enmarca en un proceso empresarial, situación que permite crear una solución a un problema real. De esta manera, conviven el desarrollo y la investigación, con el objetivo de encontrar el estado del arte de una serie de tecnologías y realizar adaptaciones que permitieron concretar el proyecto.

En este documento se expone el desarrollo fundamental que se llevó a cabo para diseñar y luego implementar un prototipo funcional. Se muestran y describen los distintos equipos utilizados en el prototipo. Además, se exponen los módulos de software que componen el sistema, junto con explicar su desarrollo y objetivo. En algunos casos se da cuenta de un proceso real de desarrollo, en el que ciertas soluciones no cumplen con los objetivos originales o simplemente los requerimientos cambiaron por razones propias de la industria y los negocios. Esto provocó que se tuvieran que adaptar ideas.

El proyecto en el que se enmarca este trabajo intenta evaluar la factibilidad de instalar estos sistemas embebidos en las grandes tiendas. La idea es obtener información acerca del inventario y usarla para optimizar procesos, lo que finalmente se traduce en aumento de ventas y disminución de costos. En el fondo, este trabajo pretende proveer la tecnlogía a un proyecto mayor con requisitos claros.

Agradecimientos

Quiero agradecer a todos los que me acompañaron en este trayecto, en especial a mi familia, amigos, y a la empresa que me dio esta gran oportunidad. Muchas gracias!

Tabla de contenido

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto

Este trabajo se enmarca en un proyecto realizado en la empresa SCOPIX S.A., la cual se dedica a proporcionar reportes, análisis, y una serie de medidas útiles para los administradores de las grandes tiendas del rubro del retail. A partir de videos obtenidos en las tiendas, SCOPIX los procesa y obtiene información de todo tipo acerca de la experiencia del cliente en la tienda, para luego procesar esta información y entregarla en un formato que permita a un jefe de tienda tomar decisiones óptimas para mejorar la calidad del servicio. La idea es generar soluciones que otorguen valor agregado, optimicen procesos, y permitan dedicar más horas al cliente final en vez de a procedimientos internos.

En el rubro del retail, las grandes tiendas enfrentan un gran problema. Es tremendamente difícil para una tienda saber en todo momento cuál es su situación de inventario, es decir, no le es posible revisar constantemente si sus productos están disponibles y si están correctamente presentados. Este problema le genera muchas pérdidas a las grandes tiendas por distintas razones, por lo que han surgido muchos proyectos que han intentado resolverlo. Este trabajo representa un sub-proyecto que pretende abordar el lado tecnológico de una solución y probarla en terreno.

La solución tecnológica involucra utilizar un tubo de aluminio bastante delgado que contiene una pequeña cámara de alta resolución en su interior. Este tubo se ubica sobre las góndolas de las tiendas. La foto que se adjunta a continuación clarifica la idea.

Figura 1.1: Fotografía del sistema instalado

Se requiere que este sistema obtenga fotografías de los productos en las góndolas a ambos lados del lugar de instalación. Luego las fotografías se deben enviar a los servidores de SCOPIX para ser evaluadas. El proceso que sigue no está considerado en este trabajo o sub-proyecto. Básicamente, lo que sucede luego de obtener las muestras de imágenes es un análisis semi-automático que obtiene el estado del inventario. A partir de esa información se genera información útil para la tienda, la cual le permite tomar decisiones operacionales para mejorar su servicio. Esto tiene un gran valor para una empresa de retail; es un hecho avalado por varios estudios del rubro.

Este sub-proyecto se centra en la electrónica del sistema y en el software utilizado y desarrollado para entregar el servicio. El detalle de las distintas etapas y las herramientas utilizadas se exponen en las siguientes secciones y capítulos.

1.2. Descripción

Inicialmente, este documento enuncia los conceptos básicos y describe el hardware que se va a utilizar para el diseño e implementación del sistema. Luego se muestra cómo se logró implementar cada bloque y módulo. Finalmente se expone y discute el rendimiento de cada parte del sistema, explicando cómo podría mejorar y las justificaciones decisionales del diseño final.

Capítulo 2

Antecedentes

Este capítulo pretende explicar y describir los conceptos base para comprender la totalidad del trabajo presentado. Se expone lo esencial de las tecnologías y herramientas utilizadas, de manera que el lector pueda referirse a este capítulo en caso de no conocer algún concepto tratado en este documento.

2.1. Hardware

2.1.1. Overo FireSTORM COM

Figura 2.1: Overo FireSTORM

El *Overo FireSTORM* es la plataforma base que se utilizó para este proyecto. A este tipo de equipos se les denomina *computer-on-module* (COM), e integran todas las funcionalidades de un computador en un paquete compacto, generalmente en una sola tarjeta. Este es un tipo de sistema embebido que se encuentra en términos de funcionalidad y poder, entre un computador clásico (laptop, escritorio) y un microcontrolador.

Este COM se basa en el procesador *DM3730* de Texas Instruments y se fabrica en tecnología de $45[nm]$. A su vez, este procesador utiliza la arquitectura *ARM Cortex-A8*, la cual es adoptada por una serie de procesadores que se insertan en equipos de uso masivo, como por ejemplo: *Nokia N900* y *Samsung Galaxy S*.

El *Overo FireSTORM*, al igual que la mayoría de los COM del mercado, es tremendamente compacto y por esta razón no cuenta con conectores de fácil acceso para periféricos. Por lo tanto, se debe usar una plataforma que exponga los buses de comunicación del COM para así poder interactuar con el y prepararlo para cumplir su función. Para este propósito se utilizaron las denominadas *expansion boards*.

2.1.2. Summit Expansion Board

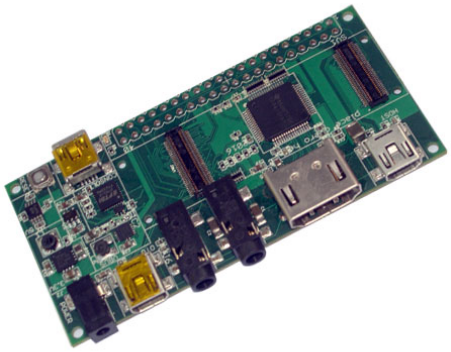

Figura 2.2: Frontal

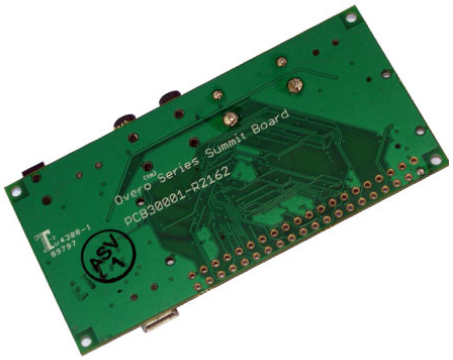

Figura 2.3: Trasero

Esta tarjeta es una *expansion board* que permite exponer los terminales para tener acceso a ellos mediante conectores estándar. En particular, algunos de los tipos de comunicación/conexión que expone esta plataforma son los siguientes:

- HDMI
- USB OTG
- USB Consola

Además, esta tarjeta permite acceso a una serie de pines de GPIO y energía entre otros.

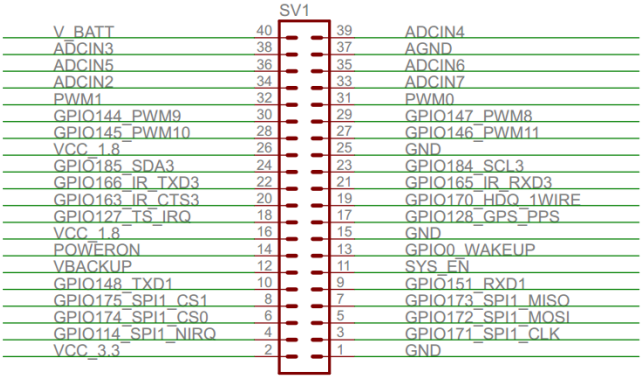

Figura 2.4: Pines de la Summit

El uso principal que se le dió a esta plataforma fue para contruir el sistema. El acceso a consola permite controlar todo el sistema de manera muy directa y fácil.

2.1.3. Pinto-TH Expansion Board

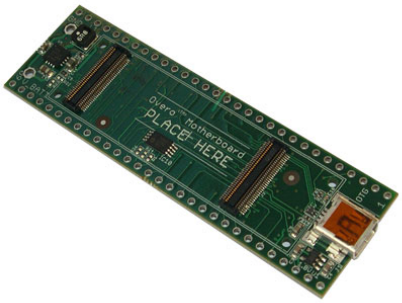

Figura 2.5: Frontal

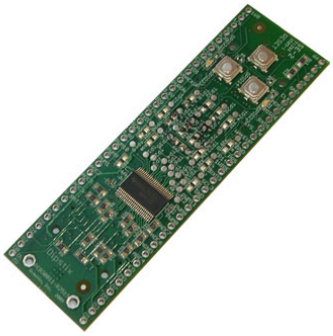

Figura 2.6: Trasero

Al igual que la tarjeta Summit, esta *expansion board* expone las funcionalidades del COM. Sin embargo, esta plataforma es mucho más compacta que la Summit y está pensada para prototipos terminados. La única conexión estándar que admite esta tarjeta es una USB OTG.

También en este caso se da acceso a ciertos pines.

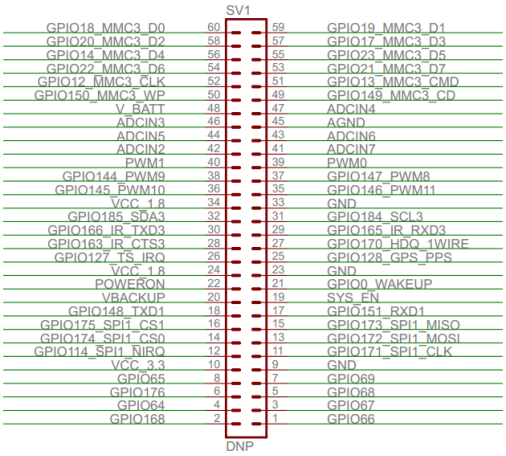

Figura 2.7: Pines de la Pinto-TH

La tarjeta Pinto-TH se usó para el producto terminado, es decir, una vez que el sistema estaba operativo en la Summit, se trasladó a esta *expansion board* para dejar el diseño compacto como exige el proyecto.

2.1.4. Cámara e-con Systems

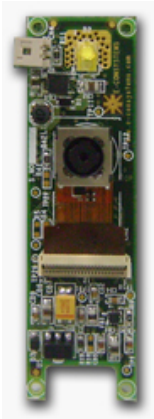

Figura 2.8: e-CAM50

Esta cámara es la que se decidió utilizar para la obtención de imágenes. Dentro de las cámaras compatibles con el COM, esta es la de mayor resolución (2592x1944), cualidad altamente requerida en el proyecto.

Para comunicarse con la e-CAM50 se cuenta con un driver V4L2 de linux. Además, la cámara incluye una aplicación que permite acceder a la gran mayoría de las funcionalidades de la cámara. Esta aplicación viene liberada con su código fuente para poder modificar y adaptar al diseño del prototipo.

Lo siguiente es una foto de cómo se ve el montaje completo de la cámara, el COM, y la tarjeta de expansión Summit.

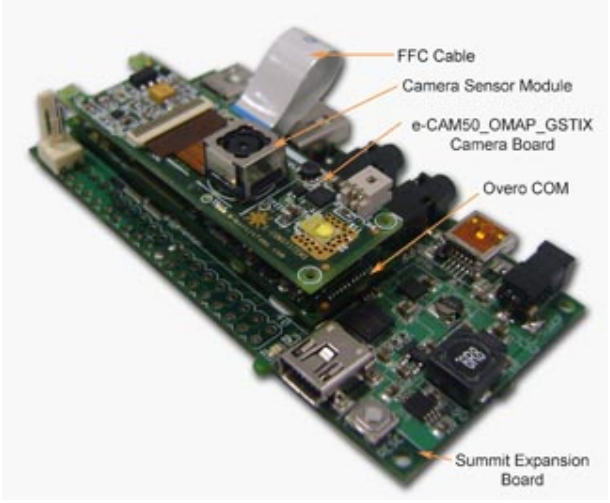

Figura 2.9: Montaje completo

2.2. Software

2.2.1. Sistema Operativo

Ångström - GNU/Linux Embebido

Ångström es una distribución de linux diseñada para operar en sistemas embebidos. El proyecto es liderado por un pequeño equipo de personas que surgieron de la unión de tres proyectos, Openembedded, OpenZaurus, y OpenSimpad.

Esta distribución es la que viene instalada en la memoria interna del COM. Además, la compañía que desarrolla la Overo FireSTORM entrega actualizaciones de esta distribución especialmente diseñadas. Esto quiere decir que incluye todo tipo de paquetes y aplicaciones para experimentar con las distintas funcionalidades ofrecidas por la plataforma.

En términos generales, Ångström se comporta como un sistema linux común y corriente. Al ser de código abierto, existen otros proyectos basados en esta distribución, ya que está especialmente pensado para adaptarse al sistema embebido que se está diseñando. Es muy probable que esta sea una razón fuerte por la que recientemente se aceptó a la distribución Ångström como un participante activo del proyecto Yocto, del cual se habla en la sección siguiente.

Yocto Project

El proyecto Yocto tiene una gran relevancia en el mundo de los sistemas embebidos. Este proyecto es parte de la *Linux Foundation*, por lo que es financiado y apoyado por múltiples compañías del ámbito *open source*.

El objetivo principal de este proyecto es generar herramientas que permitan facilitar la creación de distribuciones de linux para software embebido. Uno de los grandes valores de esta iniciativa es que es independiente de la arquitectura que se esté trabajando, la herramienta se adapta a la arquitectura y genera una distribución compatible con la máquina sobre la que se está trabajando. Esto es válido para algunas arquitecturas solamente. Sin embargo, la lista de arquitecturas es bastante amplia e incluye ARM, MIPS, PowerPC y x86/x86_64.

Hace un par de años, el proyecto OpenEmbedded, junto con su gran comunidad, se alineó con el proyecto Yocto y ahora trabajan en función de metas comunes. Esto es una gran señal de que linux está en alza y se convertirá en el sistema operativo base del hardware y software embebido.

La manera de operar de estas herramientas se basa en principalmente en *bitbake*. *Bitbake* es una herramienta enfocada en ayudar a realizar la compilación cruzada de distribuciones y paquetes de linux. *Bitbake* lee unos archivos que se denominan *recipes* o recetas, las cuales instruyen al software sobre dónde se ubica el código fuente de lo que se quiere compilar y cuáles son sus dependencias y relaciones con otros paquetes del sistema.

2.2.2. Servidor

El sistema embedido que se diseñó en este proyecto, necesita un servidor para responder a los requerimientos de imágenes que llegan del exterior. Por esta razón se estudiaron las distintas posibilidades y se optó por una aplicación de buena mantención en la distribución y un uso muy bajo de recursos. En realidad, la decisión es muy simple de tomar para el caso de sistemas embebidos, ya que existe mucha información y es básicamente aceptado por gran parte de la comunidad el uso del software *lighttpd* para este tipo de implementaciones.

A *lighttpd* se le llama lighty y es una aplicación de servidor tremendamente liviana y eficiente. Esta es utlizada por sitios de muy alto tráfico web como lo son Youtube y Wikimedia. Este software permite cumplir los estándares de seguridad y fiabilidad de cualquier servidor competitivo.

Para este proyecto, el uso de lighty cumple la función de alojar el sitio de configuración del sistema junto con atender los pedidos de imágenes. El sitio de configuración es una versión reducida de una aplicación de un router común y corriente, es decir, permite observar el estado del sistema y cambiar los parámetros de IP, nombre de la red, y contraseña entre otras variables menores. Por otro lado, los pedidos de imágenes se sirven mediante un programa interno que devuelve un archivo imagen.

2.2.3. Programación

Para implementar las distintas aplicaciones del sistema se recurre a distintos lenguajes de programación. La elección de estos lenguajes está pensada en el desarrollo rápido de un prototipo y no considera criterios de mantención de código ni eficiencia. La idea es generar una prueba de concepto.

bash

Bash fue elegido como lenguaje de scripting por su simplicidad. En realidad, bash es un shell de Unix que acepta una gran cantidad de comandos de alto nivel. Por este motivo, es muy fácil generar pequeños programas o scripts que hacen tareas realmente complejas. La desventaja es que el código es poco portable y la mantención es nefasta, es decir, los scripts no son ampliamente compatibles con otros sistemas y son sensibles a los cambios en el funcionamiento del software. Sin embargo, el proyecto exige probar un concepto y el objetivo es hacerlo rápido, razón por la cual bash se ajusta perfectamente.

La idea es que los scripts escritos con bash cumplan las funciones de monitoreo, edición y movimiento de archivos, cambios en la configuración del sistema, etc.

PHP

PHP es un lenguaje de programación del lado del servidor que fue diseñado para el desarrollo web. Es un lenguaje con una gran documentación y permite insertarse directamente en el HTML, por lo que es bastante simple de aprender y utilizar. Es por esta razón que se elegió para construir algunas de las aplicaciones que se alojan en el servidor del sistema.

Cron

Cron es un programa de Unix que permite organizar tareas periódicas en segundo plano. La idea de cron es configurarlo para que ejecute ciertos programas o scripts en fechas y horarios específicos cuando el sistema se encuentra activo.

Este programa es justamente lo que se necesita en este sistema embebido para supervisar procesos y asegurar que todo está en orden con el servicio que se requiere entregar.

Capítulo 3

Metodología

3.1. Diseño del Sistema Embebido

En esta sección se describe en términos generales lo que compone la implementación del sistema embebido. Se enuncian las partes del sistema para comprender la arquitectura general y luego se detalla la implementación de cada bloque.

3.1.1. Objetivos del Sistema

Todo sistema embebido tiene objetivos y exigencias muy claras con respecto a la función que debe cumplir y la confiabilidad que debe ofrecer. Para este proyecto, los objetivos operacionales son bastante simples. En términos generales se requiere una cámara muy compacta que pueda ser controlada de manera remota. Esto involucra una serie de requerimientos:

- El hardware que obtiene las imágenes debe ser lo más pequeño posible.
- El sistema debe contar con una conexión a internet.
- Sitio remoto de configuración (similar al de un router).
- Botón de restauración de fábrica del sistema.
- Se debe reportar y monitorear el estado del sistema.
- Inicialización automática de los servicios.

Existen situaciones en que es tremendamente grave que un sistema embebido falle, por ejemplo cuando se opera alguna función sensible de un automóvil o alguna maquinaria pesada. En el caso de este proyecto, las fallas influyen solamente en términos económicos a través de las mantenciones que se deban hacer. Sin embargo, esto parece ser un tema muy sensible en este proyecto, ya que en experiencias similares el item de costos de mantención ha sido un factor clave al evaluar la factibilidad de

la implementación. Es por esta razón que se agregaron monitoreos con capacidad de solucionar problemas simples. Lo que se decidió monitorear es la conexión a internet, la cámara de fotos, y el servidor interno que aloja las aplicaciones de configuración remota.

3.1.2. Arquitectura del Sistema

Para cumplir con los requerimientos que exije el proyecto, se diseñó una estructura del sistema que adopta las tecnologías disponibles en la plataforma base. Esta estructura se separa en dos partes, hardware y software.

Hardware

La estructura o arquitectura de hardware es bastante simple. El COM es el centro del sistema. Este se comunica con el exterior mediante Wifi y algunos botones de propósito general. Además, cuenta con una entrada paralela para conectar la cámara y operarla.

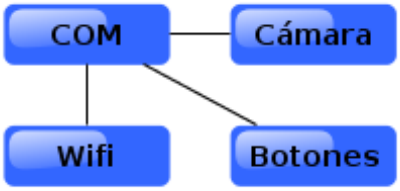

Figura 3.1: Estructura Hardware

En términos de conexiones físicas, el sistema embebido que se quiere diseñar es tremendamente simple como se ve en la figura. Sin embargo, para lograr que las partes interactúen correctamente se debe trabajar bastante en el software.

Software

El software se organiza de la siguiente manera. Primero se tiene el sistema operativo basado en linux, sobre el cual se monta todo el resto de las aplicaciones que requiere el sistema. Las aplicaciones se dividen en dos, las que se operan bajo un servidor con conexión a la red y las que cumplen una función interna en el sistema.

Las aplicaciones que se instalan en el servidor corresponden a los servicios que se deben externalizar. Esto incluye la obtención de imágenes, el sitio de configuración del sistema, y los reportes de estado que se envían al administrador remoto.

Por otro lado están las aplicaciones que cumplen funciones locales. Estas corresponden al monitoreo interno, la restauración de configuración, y la inicialización automática de las funciones o servicios. La restauración va ligada a los botones que se mencionaron en la sección de hardware.

3.2. Implementación del Sistema Embebido

3.2.1. Sistema Operativo

Para este tipo de plataformas, los sistemas operativos disponibles son todos basados en linux. Se experimentó con varias opciones, pasando por distribuciones prefabricadas hasta compilando versiones personalizadas mediante las herramientas del proyecto Yocto. Finalmente se optó por una imagen de distribución que entregó el fabricante de la cámara. Esto se debe a que los drivers de la cámara no se entregaron junto a su código fuente, por lo que se debían instalar en un kernel compatible. Para simplificar este paso, se instaló una imagen preparada como se explica a continuación.

Antes, información acerca de la plataforma. El COM posee una memoria interna en la que se aloja un OS de pruebas que permite entrar al sistema y probar algunas funcionalidades. Además, cuenta con un slot de tarjetas microSD para flexibilizar la prueba de distintos sistemas. Este mecanismo fue el que se usó para montar el OS necesario, por lo tanto, se conectó una tarjeta microSD al computador de desarrollo mediante un adaptador, y se grabó la imagen prefabricada. Esto se realiza de la siguiente manera:

Inicialmente es preferible eliminar todo el contenido de la tarjeta:

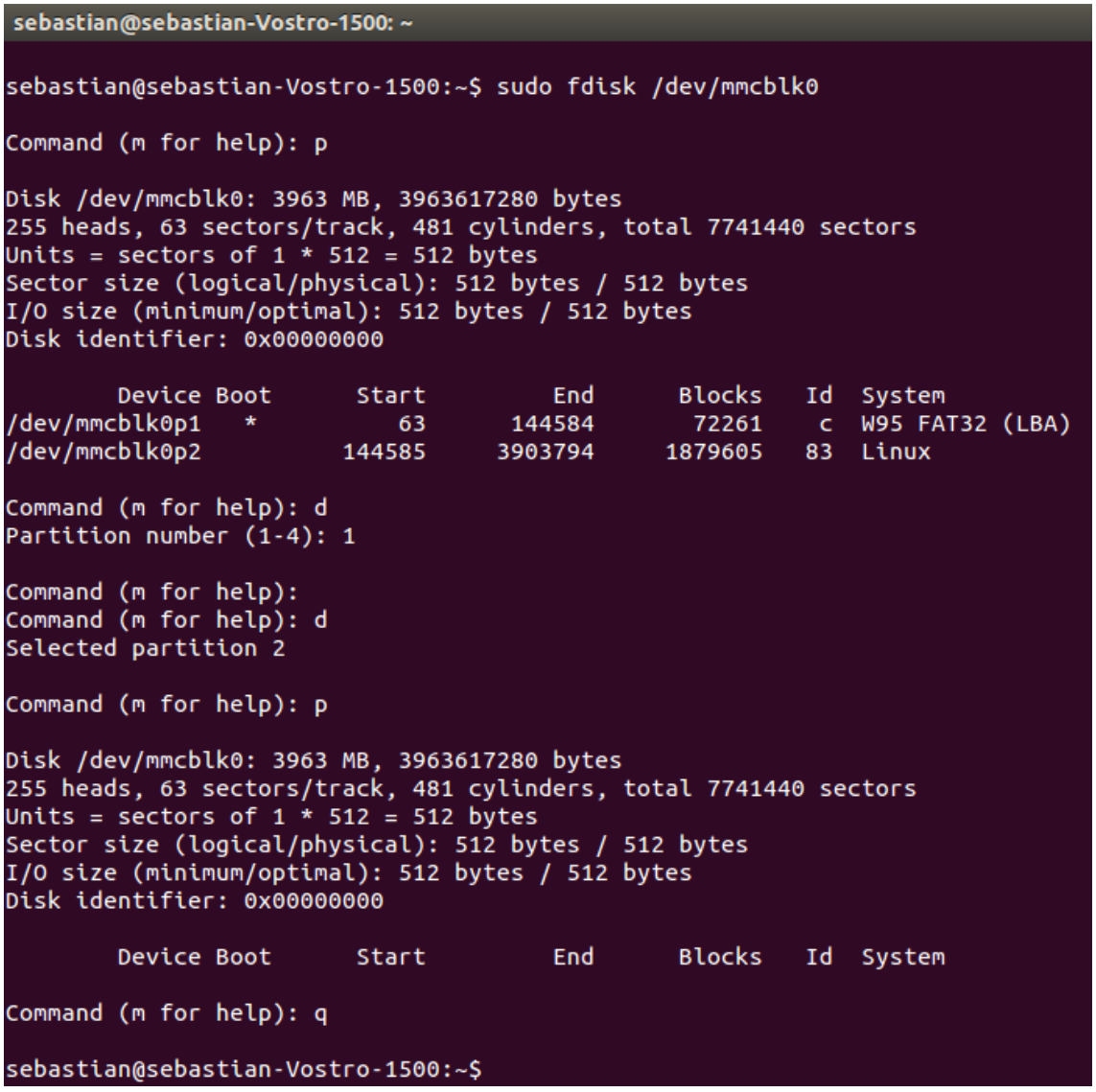

Figura 3.2: Vaciar tarjeta.

Como se aprecia en la figura, se ejecuta el programa *fdisk* sobre la tarjeta microSD que está montada en el directorio:

```
/dev/mmcblk0
```

El programa permite observar las particiones existentes (*p*) y eliminarlas (*d*). De esta manera se limpia completamente la tarjeta microSD y queda lista para el siguiente paso.

Para grabar la imagen se utiliza un comando que se ilustra en la siguiente figura, junto con el resultado del proceso.

```
sebastian@sebastian-Vostro-1500:~$ sudo dd if=e-CAM32_OMAP_GSTIX_1.356.img of=/d
ev/mmcblk0 bs=1M
1910+0 records in
1910+0 records out
2002780160 bytes (2.0 GB) copied, 332.239 s, 6.0 MB/s
sebastian@sebastian-Vostro-1500:~$
```

Figura 3.3: Grabar imagen.

El parámetro *if* recibe el nombre del archivo imagen y el parámetro *of* el destino, es decir, el directorio donde se encuentra montada la tarjeta. Luego de que termina este proceso, se deben modificar algunos archivos. Primero, una explicación sobre esta imagen. Esta contiene dos particiones, una para *bootear*, es decir, para ejecutar el sistema operativo, y otra que contiene el sistema de archivos completo. Lo que se debe modificar tiene relación con el *booteo* y se debe al hecho de que se está utilizando una plataforma de la serie STORM.

Para hacer modificaciones, primero se deben montar las particiones como se muestra en la figura que sigue.

```
sebastian@sebastian-Vostro-1500: ~
sebastian@sebastian-Vostro-1500:~$ sudo mkdir /media/{boot,rootfs}
sebastian@sebastian-Vostro-1500:~$ sudo mount -t vfat /dev/mmcblk0p1 /media/boot
sebastian@sebastian-Vostro-1500:~$ sudo mount -t ext3 /dev/mmcblk0p2 /media/rootfs
sebastian@sebastian-Vostro-1500:~$
```

Figura 3.4: Montar particiones.

Luego lo que se hace es borrar los archivos conflictivos y reemplazarlos por las versiones actualizadas. La figura que sigue muestra una manera de hacer esto:

```
sebastian@sebastian-Vostro-1500: ~
sebastian@sebastian-Vostro-1500:~$ sudo rm /media/boot/MLO
sebastian@sebastian-Vostro-1500:~$ sudo rm /media/boot/u-boot.bin
sebastian@sebastian-Vostro-1500:~$ sudo cp mlo-updated /media/boot/MLO
sebastian@sebastian-Vostro-1500:~$ sudo cp u-boot.bin /media/boot/u-boot.bin
sebastian@sebastian-Vostro-1500:~$
```

Figura 3.5: Reemplazar archivos conflictivos.

De esta manera queda operativa la tarjeta microSD. Se debe desmontar del computador de desarrollo y luego insertar en la plataforma para iniciar el sistema.

3.2.2. Cámara

La utilización de la cámara requiere algunos ajustes del sistema. Esto se realiza mediante una sesión de terminal en la plataforma.

Una vez que se inicia el sistema lo que se ve en pantalla debiera ser algo similar a lo que muestra la siguiente figura:

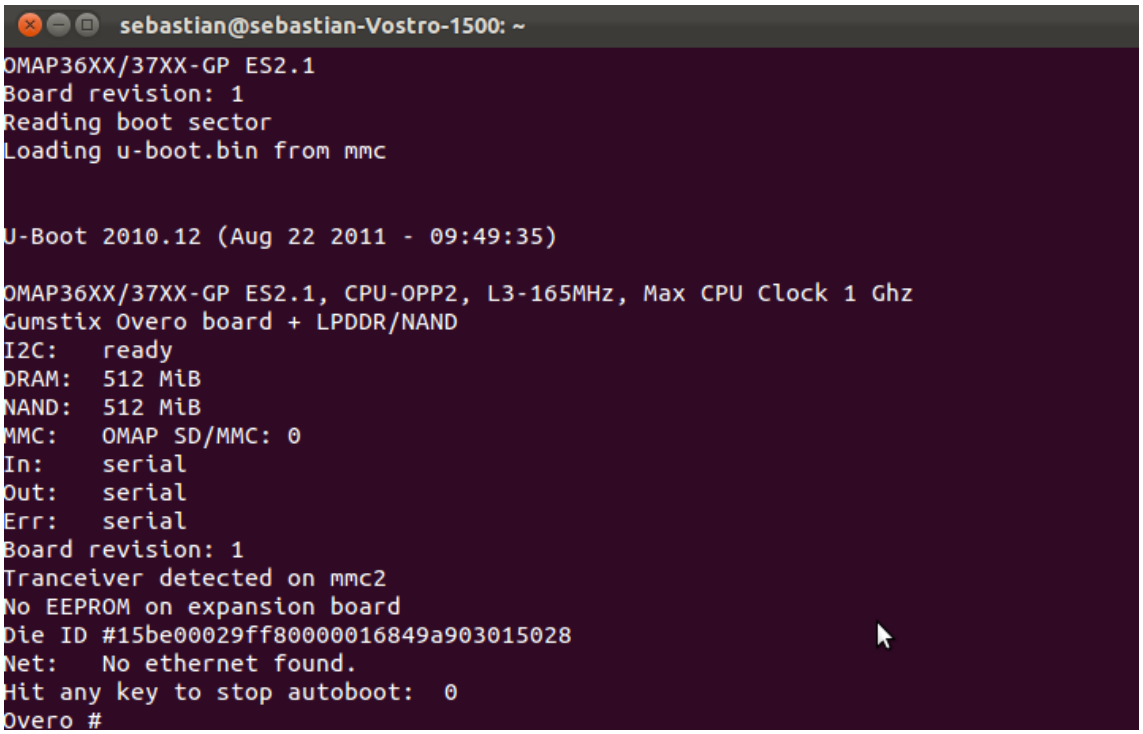

Figura 3.6: Inicio sistema.

Se observa en el pantallazo que en un momento apareció un diálogo que dice *Hit any key to stop autoboot*. En ese momento se debe apretar alguna tecla para entrar a lo que se denomina el *bootloader*. Esto es el primer programa que se ejecuta y está encargado de manejar algunas variables de entorno y de inicializar el sistema operativo. El ajuste requerido tiene relación con las variables del sistema. Es necesario cambiar algunas y agregar otras. La figura que se adjunta a continuación lista lo que se debe ejecutar en esta situación:

```
setenv i2cspeed "3,100"

setenv mmcargs setenv bootargs mem=87M@0x80000000 mem=128M@0x88000000
console=${console} vram=${vram} omapfb.mode=dvi:${dvimode}
omapfb.debug=y omapdss.def_disp=${defaultdisplay} i2c_bus=${i2cspeed}
root=/dev/mmcblk0p2 rw rootfstype=ext3 rootwait
```

Figura 3.7: Variables de entorno.

Primero se crea una variable que guarda la velocidad de transferencia del protocolo i2c, el cual es utilizado con la cámara. Esto es necesario debido a un problema del driver originado por el fabricante. Luego se asignan las variables a sus valores correspondientes y queda configurado el sistema. Vale la pena mencionar que estas variables quedan guardadas en la memoria NAND interna de la plataforma, por lo que no es necesario realizar este paso cada vez que se inicia sesión. Para asegurar que todo queda bien guardado e iniciar el sistema operativo se ejecuta lo siguiente:

```
saveenv
reset
```

Luego se espera a que se termine de inicializar el sistema operativo. En el directorio raiz debe haber una carpeta asociada a la cámara. Dentro de esta se encuentra el driver que se necesita instalar y una aplicación de prueba que permite configurar la cámara y tomar imágenes. Al localizar el driver se ejecuta un comando que se visualiza en la figura:

```
sebastian@sebastian-Vostro-1500: ~

overo login: root
root@overo:~# insmod /e-CAM32_OMAP_GSTIX_1.356/Driver/Binary/v4l2_driver.ko
-------------------------------------------------
PRODUCT_NAME       : e-CAM32_OMAP_GSTIX
SVN_REVISION       : 356
SVN_DATE           : 2011-12-27
SVN_TIME           : 13:59:38
Driver Module info : V4l2 driver module
Build Time stamp   : Thu Dec  8 19:08:03 2011
-------------------------------------------------

Memory overlap detected between camera reserved memory and kernel memory
phy_addr_start  0x85d00000
phy_addr_end    0x862fffff
phys_end_kernel 0x8d700000

Using Reserved memory for V4l2 driver module
From Start address - 0x85d00000
To End address - 0x862fffff

ov3640 found: product id is 0x36
lm3553 flash ic not found
root@overo:~#
```

Figura 3.8: Insertar driver.

Se ve que existen algunos errores al insertar el módulo, pero estos no son de

gravedad. En el caso en que la inserción del módulo falle, aparecen mensajes claros diciendo que el driver o módulo no es compatible con el kernel. Muchas veces pasa que la inserción falla debido a que el cable de la cámara está mal conectado. Lamentablemente estas conexiones son bastante sensibles y delicadas, por lo que se debe tener mucho cuidado. En este momento en que el mensaje de inserción es exitoso, la cámara está lista para operar.

3.2.3. Servidor

Al contar con una plataforma limitada, la instalación de un servidor propone un desafío de encontrar una aplicación liviana y confiable. En esta búsqueda asoma rápidamente *lighttpd* (*lighty*), debido a su uso reducido de recursos y a su estabilidad. Además, existe un requerimiento adicional, el servidor debe aceptar conexiones seguras del tipo HTTPS. Por esta razón, la instalación del servidor se torna un poco más compleja.

Para cumplir el requerimiento especial, la aplicación debe ser compilada desde el código fuente. Gracias a que el OS utilizado cuenta con las herramientas indicadas para compilar, se optó por la opción simple de construir la aplicación de manera local.

Primero se debe contar con una conexión a internet. La manera de hacer esto se explica en la sección dedicada al Wifi. Luego se utiliza el gestor de paquetes (*opkg*) para instalar algunas aplicaciones necesarias:

```
# opkg install openssl openssl-dev libpcre-dev
```

Luego se requiere descargar la aplicación del servidor. La manera más fácil de hacerlo es a través de la línea de comando. Se descarga, descomprime, y sitúa la sesión de terminal en la carpeta indicada:

```
# wget
    "http://download.lighttpd.net/lighttpd/releases-1.4.x/lighttpd-1.4.31.tar.gz"
# tar zxvf lighttpd-1.4.31.tar.gz
# cd lighttpd-1.4.31
```

Dentro de la carpeta se encuentran archivos de tipo README e INSTALL. Se deben seguir las instrucciones de estos archivos para poder instalar correctamente el software.

```
# ./configure --with-openssl --with-openssl-libs=/usr/lib --without-bzip2
```

La carpeta */usr/lib* es donde se ubican las librerias instaladas de openssl en este caso. Es importante corroborar que las librerias estén en esa carpeta con el comando:

```
# ls /usr/lib | grep ssl
```

El output debería mostrar algunos archivos asociados a estas librerias. El flag –*without* se tuvo que agregar para evitar errores debido a paquetes no instalados. El programa *configure* avisa si no tiene los paquetes y es posible instalarlos con *opkg*, o ignorarlos con este flag, dependiendo de su utilidad. El paquete obligatorio es el de openssl. Luego:

```
# make
# make install
```

Estos comandos deben correr sin problemas. Luego se debe configurar lighttpd para aceptar las conexiones HTTPS. Se genera el certificado ssl y luego se modifica el archivo de configuración, *lighttpd.conf*.

```
# mkdir /etc/lighttpd/ssl/domain.com -p
# cd /etc/lighttpd/ssl/domain.com
# openssl req -new -x509 -keyout server.pem -out server.pem -days 365 -nodes
# chown lighttpd:lighttpd /etc/lighttpd/ssl -R
# chmod 0600 /etc/lighttpd/ssl/domain.com
```

Figura 3.9: Generar certificado.

Al ejecutar el programa openssl para generar el certificado, la aplicación requiere una serie de información muy básica. Esto incluye el país, nombre de dominio, etc. Esto no requiere ser muy formal, ya que el certificado se utiliza sólo internamente. Para editar el archivo de configuración:

```
# vi /etc/lighttpd/lighttpd.conf
```

Figura 3.10: Editar archivo de configuración.

Simplemente mediante un editor de texto se abre el archivo indicado y se le agrega el siguiente bloque de información:

```
$SERVER["socket"] == "192.168.1.100:443" {
server.document-root = "/home/lighttpd/domain.com"
ssl.engine = "enable"
ssl.pemfile = "/etc/lighttpd/ssl/domain.com/server.pem"
}
```

Figura 3.11: IP y directorio de aplicaciones.

Es importante elegir el *document-root* y la IP que corresponden. El *document-root* es la ubicación de las aplicaciones del servidor. En el caso de esta implementación, se utilizó el directorio */www/pages*. Con esto queda habilitado el servidor para conexiones HTTPS a través del puerto 443.

3.2.4. Imágenes

La obtención de imágenes es un servicio que tiene requerimientos específicos de implementación. Se exije que el sistema tenga un comportamiento similar al de una cámara IP, por lo que se debe incorporar una API HTTP que permita pedirle imágenes al sistema. Por esta razón se analizó la situación y se buscó un lenguaje de programación que permita hacer esto. De inmediato PHP suena como una excelente opción debido a la gran cantidad de documentación y además la facilidad de instalación en el servidor.

La API es bastante simple. Al conectarse al servidor del sistema a través de su IP fija, este carga por defecto la aplicación PHP que genera las imágenes. Esta aplicación recibe tres parámetros mediante el método GET de HTTP, es decir, se incorporan en la URL. Estos parámetros son ancho (w), alto (h), y nivel de exposición (e). Este último tiene que ver con la cantidad de tiempo que permanece abierto el obturador, por lo que controla en gran medida la luminosidad de la foto.

Un ejemplo de una llamada a la aplicación es la siguiente:

```
https://192.168.2.1/?w=2592&h=1944&e=700
```

Esta llamada finalmente provoca que la aplicación llame a un script de linux (bash) que se encarga de correr la aplicación de la cámara, sacar la foto, y convertirla a un formato indicado. Una vez que la foto está lista para descargar, la aplicación de PHP se encarga de redirigir al usuario a un sitio de descarga automática, el cual envía la foto directamente en formato JPG.

En caso de que hayan problemas o errores, se generan mensajes detallados acerca del problema y se le muestran al usuario con una redirección al sitio de errores.

3.2.5. Configuración

El sitio de configuración es la aplicación más compleja que se desarrolló. La idea es proporcionar al usuario un sitio fácil de utilizar que permita cambiar los datos de red del sistema. Al igual que un router o una cámara IP, este sistema debe adaptarse a las condiciones en las que se va a instalar, por este motivo es muy importante que esta aplicación funcione perfectamente. El hecho de que la plataforma no cuente con una conexión ethernet directa hace mucho más relevante a esta aplicación.

3.2.6. Monitoreo

La condición remota de la instalación de este sistema embebido produce la necesidad de monitorear o supervisar los procesos y servicios. Esto permite realizar pruebas y analizar el comportamiento del sistema bajo condiciones variadas.

El monitoreo considera cuatro pilares clave del sistema, la conexión a internet, el driver de la cámara, el servidor, y el driver del wifi. Mediante un *cronjob* se ejecuta un script periódicamente que revisa el estado de cada variable a supervisar. La conexión a internet se prueba con un *ping* al router/access point; se envían un número fijo de paquetes y se evalúa si la conexión está activa. En el caso de los drivers se revisa que estén cargados en el kernel y para el servidor simplemente se necesita buscar en la lista de programas en ejecución y confirmar que está presente. Una vez que se observa el estado de cada item, se guarda el resultado en un archivo de texto. La idea final es que las evaluaciones se acumulen y se envíen al administrador del sistema. Se determinó que la mejor manera de hacer esto era mediante correo electrónico, por lo que se buscó un cliente de correos compatible con el OS.

El monitoreo se realiza bastante seguido y luego todos los resultados son enviados por correo electrónico etiquetados por fecha y hora. De esta manera se obtiene una idea del comportamiento dinámico de los procesos. La periodicidad de las evaluaciones y envíos se elige arbitrariamente y se configura muy fácilmente en el *cronjob*.

3.2.7. Restauración

En la situación en que haya un problema de configuración de los datos de red y se pierda la conexión al sistema, se necesita volver a la configuración por defecto. Este objetivo se logra gracias a que el kernel de linux permite de una manera muy simple exponer el estado de los botones de la plataforma. Mediante el software *sysfs* se tiene acceso a saber si un botón está presionado o si no lo está. De esta manera se programó un script que se mantiene atento al estado del botón y gatilla la función de restauración en caso que el botón permanezca presionado por un tiempo mayor a diez segundos.

La restauración consiste en que la IP y los datos de red se reemplazan por los valores que se definen por defecto y se reinicia el sistema. Esto permite tener una configuración conocida para volver atrás en caso de fallas o problemas.

3.2.8. Inicialización Automática

Al iniciarse el sistema operativo, se deben iniciar también los procesos y servicios. En general, cuando se instala un programa este viene con lo que se denomina un *boot script*, es decir, un pequeño software que se ocupa de inicializar el programa. Para el software que se programó específicamente para esta implementación, se necesitó generar *boot scripts* especiales. La función que realizan estos scripts es muy similar a lo que hace el software de monitoreo cuando detecta que un servicio no está operativo. Se cargan los drivers y se ejecutan los procesos y servicios necesarios.

Este tema es profundo y tiene muchas aristas. Sin embargo, existe una manera simple de lograr que un *boot script* se ejecute correctamente al iniciar el sistema. El software que facilita la tarea se llama *update-rc.d* y se ocupa de tomar el script y generar accesos directos en los directorios correspondientes. Un ejemplo:

```
# update-rc.d lighttpd defaults
```

Esto crea accesos directos al script que inicializa el servidor.

Capítulo 4

Resultados y Discusión

En este capítulo se pretende presentar y discutir el rendimiento de cada parte del sistema embebido implementado. El objetivo es exponer las fortalezas y debilidades, siendo crítico acerca de lo que se puede mejorar.

4.1. Implementación

Esta sección presenta los resultados que entregó la implementación del sistema. Esto tiene relación con analizar si cada bloque del sistema cumple su objetivo o si lo hace solo parcialmente.

4.1.1. Sistema Operativo

El uso de una imagen prefabricada simplificó enormemente los pasos posteriores. Además, el hecho de que el OS esté basado en linux ayudó a flexibilizar la configuración del sistema. Linux permite meterse en cada detalle del sistema, por lo que cada bloque pudo ser adaptado de manera exitosa. Esta distribución prefabricada cuenta con los ambientes de desarrollo que permitieron compilar programas e instalar dependencias mediante el gestor de paquetes. Esto no es trivial cuando se piensa en crear una distribución personalizada.

Los experimentos que se llevaron a cabo con el proyecto Yocto para producir una distribución personalizada, demostraron que la curva de aprendizaje es cada vez más empinada cuando se quieren ajustar detalles. Tomó bastante tiempo generar un OS efectivo, lo cual fue tremendamente valioso pensando en la escalabilidad del proyecto, pero como solución de prototipo es demasiado trabajo para una prueba de concepto. Sin embargo, el conocimiento adquirido se aplicó y ayudo a entender temas esenciales de

una distribución de linux, lo cual significó una mayor rapidez en la solución de problemas.

4.1.2. Cámara

El modelo de cámara que se utilizó no incluía el código fuente de su driver. Esto limitó las opciones pero aún así la cámara respondió como se esperaba. La única situación que produjo problemas fue el parámetro de exposición. Este parámetro siempre se utilizó en modo automático y en la aplicación de pruebas original no se podía cambiar. Por alguna razón desconocida, el ajuste de exposición automático no funcionaba bien, por lo que muchas veces se generaban imágenes excesivamente brillantes. Para solucionar esto se aprovechó que la aplicación de prueba si incluía su código fuente, por lo que se editó para que permitiera modificar este parámetro y darle un valor fijo aceptable.

Por el lado del hardware, la conexión física entre la cámara y la plataforma COM fue débil. Los cables paralelos eran delicados y varias veces hubo que cambiarlos. De hecho, se encargaron más de estos cables a EEUU por temor a que fallaran todos. En todo caso, es probable que la constante manipulación haya provocado el daño mecánico a los cables. Sin embargo, fue un problema que hubo que abordar y afectó el rendimiento del sistema.

En ocasiones se observó que las imágenes tomadas bajo luces de tubos fluorescentes contenían marcas o bandas oscuras. No fue posible encontrar la verdadera razón por la que eso ocurría, pero se pensó que tenía que ver con la calidad del sensor. Sin embargo, este fenómeno no afectó la calidad de las imágenes en la prueba de instalación. A continuación se adjunta una foto tomada por el sistema instalado:

Figura 4.1: Foto tomada utilizando el sistema en el lugar de instalación.

4.1.3. Servidor

El servidor lighttpd fue sin lugar a dudas una de las piezas de mejor rendimiento del sistema. Muy estable y rápido en todo momento, considerando que fue instalado en un sistema de bajo poder de procesamiento. También es cierto que el servidor recibió muy pocas conexiones externas simultáneas, por lo que se esperaba que no colapsara tan fácilmente. En todo caso, el objetivo del servidor era servir a un único usuario a la vez, por lo que este requerimiento se cumplió a la perfección.

4.1.4. Imágenes

El servicio de obtención de imágenes también se comportó de excelente forma durante la prueba de instalación. Muy rara vez ocurrió algún error y las veces que si ocurrió, siempre bastó con reintentar el proceso para obtener una imagen válida.

El funcionamiento conjunto entre PHP y el servidor fue en gran parte el responsable del buen rendimiento de esta pequeña aplicación. Siendo PHP un lenguaje maduro y estable, el programa no tenía muchas opciones de fallar. También se debe mencionar que el software se comunica con la aplicación de pruebas de la cámara, por lo que esa aplicación también resultó ser de buena calidad según las pruebas que se hicieron. Prácticamente nunca se originó un error en este bloque de servicio.

4.1.5. Configuración

La falta de una conexión ethernet directa como la que existe en las cámaras IP hace que la instalación de este equipo se complique. El sitio de configuración está para cambiar los datos de manera simple una vez que el sistema ya está conectado, por lo que es necesario configurar un router/access point especialmente para el equipo, lo cual es una gran desventaja.

Considerando las limitaciones, el sitio de configuración respondió en todo momento. Se detectaron una serie de *bugs* en la aplicación, los cuales no tenían efectos graves sobre el funcionamiento del software sino que eran más bien casos de borde que no se abordaron debido a lo poca prioridad que presentaban. Además, gran parte de la aplicación se desarrolló con bash, un lenguaje muy poderoso para scripts de tamaño medio o pequeño, por lo que el sitio de configuración se hizo cada vez menos mantenible a medida que crecía. Esta desventaja se compensa con la rapidez con que se construyó la aplicación, hecho que es muy valioso cuando un proyecto se trata de una prueba de concepto.

4.1.6. Monitoreo

El monitoreo remoto que se realizó en las pruebas estuvo activo tres días, de sábado a lunes. Después de ese tiempo no se sabe qué sucedió, ya que nadie tuvo la oportunidad de ir al sitio de instalación nuevamente. Una opción es que simplemente le hayan quitado la energía al sistema; esto se deduce de que en pruebas previas no oficiales, el sistema se mantuvo activo durante semanas. También es una opción probable que haya fallado la conexión a internet. Los reportes indicaron que la conexión no era estable todo el tiempo. Sin embargo, los reportes llegaron cada una hora sin falta, por lo que llegaron alrededor de 76 reportes. Esto indica que la conexión no debe haber sufrido demasiado. Es por esta razón que se cree que la inactividad del sistema fue provocada por agentes externos.

En realidad el monitoreo más importante tiene que ver con la conexión a internet. Los drivers y el servidor tenían una probabilidad muy baja de fallar. El monitoreo de estos procesos responde a la necesidad de estar completamente seguros que están activos y descartar algún posible mantenimiento. En este sentido el objetivo se cumplió. La única desventaja de este programa de monitoreo se comparte con el sitio de configuración. En gran parte se programó con bash, lenguaje que resultó muy útil para desarrollo rápido y efectivo, pero que a largo plazo demostró ser completamente inútil por su dificultad de debugueo y mantención.

4.1.7. Restauración

El botón de restauración que se implementó cumplió el objetivo de una manera simplificada. Originalmente, un botón de restauración debe generar una excepción de alta prioridad que se ejecuta aisladamente. En este caso, la excepción se detecta a través del OS, por lo que el botón no hace efecto si el OS no se ha cargado correctamente. Esta es una limitación que se aceptó debido a la complejidad que supone generar la excepción aislada. Eso requiere entender en profundidad cómo funciona un *bootloader*, lo cual se alejaba de los objetivos del proyecto.

Nuevamente se enfrenta un proceso que tiene la característica de prueba de concepto. Aunque nunca se necesitó realmente utilizar, las pruebas fueron efectivas y el sistema respondió como se esperaba.

4.1.8. Inicialización Automática

Uno de los objetivos que se quería alcanzar al generar una distribución personalizada era justamente que la inicialización o booteo fuera más ágil. Esto efectivamente se logró hacer, alcanzando una gran rapidez de carga del OS y todos sus servicios y procesos.

Fue el driver de la cámara, como se explicó anteriormente, el que provocó volver a la imagen prefabricada. Aunque la carga del OS era más lenta, los *boot scripts* siempre hicieron su trabajo correctamente y los procesos se encendían y mantenían activos al momento de iniciar el OS.

Capítulo 5

Conclusiones

Considerando que el proyecto fue una prueba de concepto, los objetivos generales se cumplieron. Como en todo proyecto de una empresa, las prioridades son distintas a las que se enmarcan en un proyecto académico. En este caso, la investigación llegaba hasta el punto en que se obtenía un módulo funcional o hasta que se estimaba que la investigación más profunda tomaría más allá del tiempo disponible. En esta última situación se decide tomar un camino distinto que cumpla los requerimientos esenciales. Lo importante es alinear el objetivo de la empresa con el proyecto y eso es lo que se hizo. Se necesitaba probar que un sistema embebido de características muy compactas lograra obtener imágenes de manera remota a través de una API HTTP. El camino recorrido para llegar a ese objetivo involucra una gran cantidad de investigación que no se refleja en la implementación final. Un ejemplo de esto es el proyecto Yocto, el cual no tiene presencia en el prototipo pero generó una visión de lo que se puede crear a largo plazo, lo cual es tremendamente valioso en un proyecto así.

Como continuación a este trabajo, se propone tomar las ideas que se discutieron con respecto al largo plazo. Esto se refiere al uso de lenguajes de programación que permiten darle escalabilidad a este proyecto. Además, se propone pensar en diseñar una plataforma propia que resuelva algunas de las limitaciones que se encontraron. Los siguientes pasos tienen relación con transformar una prueba de concepto en un producto real, por lo tanto, es un trabajo enorme que debe tener el respaldo de la evaluación del proyecto en términos económicos.

Bibliografía

[1] *Compiling Kernel Modules*, 2013. http://tldp.org/LDP/lkmpg/2.6/html/x181.html.

[2] *Gumstix*, 2013. http://gumstix.org.

[3] *Gumstix + Camera Demo*, 2013. http://www.oz9aec.net/index.php/gumstix-overo/457-gumstix-overo-water-ecam32-camera-demo.

[4] *Gumstix Buttons and LEDs*, 2013. http://www.jumpnowtek.com/index.php?option=com_content&view=article&id=80&Itemid=67.

[5] *JumpNow Tips*, 2013. http://www.jumpnowtek.com/index.php?option=com_content&view=article&id=70&Itemid=89.

[6] *JumpNow Yocto*, 2013. http://www.jumpnowtek.com/index.php?option=com_content&view=article&id=85:yocto-gumstix&catid=35:gumstix&Itemid=67.

[7] *lighttpd + ssl*, 2013. http://www.cyberciti.biz/tips/how-to-install-ssl-lighttpd-https-configuration.html.

[8] *Monitoreo*, 2013. http://www.anyexample.com/linux_bsd/bash/check_if_program_is_running_with_bash_shell_script.xml.

[9] *OpenEmbedded*, 2013. http://docs.openembedded.org/usermanual/usermanual.html.

[10] *PHP*, 2013. http://www.php.net/manual/es/index.php.

[11] *update-rc.d*, 2013. http://www.debuntu.org/how-to-managing-services-with-update-rc-d/.

[12] *Wifi on Gumstix*, 2013. http://fastr.github.io/articles/marvell-wifi-on-gumstix-overo-fire.html.

[13] William E. Shotts, Jr.: *The Linux Command Line*. LinuxCommand.org, 2009.