UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

"DISEÑO E IMPLEMENTACIÓN DE UN FRAMEWORK DE DESARROLLO DE APLICACIONES DE VIDEO DIGITAL"

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

MARCELO VALENZUELA DIEGUEZ

PROFESOR GUÍA:
JOSÉ MIGUEL PIQUER GARDNER

MIEMBROS DE LA COMISIÓN:
LUIS MATEU BRULÉ
ERIC TANTER

SANTIAGO DE CHILE
JULIO 2008

RESUMEN DE LA MEMORIA
PARA OPTAR AL TITULO DE
INGENIERO CIVIL EN COMPUTACION
POR MARCELO VALENZUELA DIEGUEZ
PROF. GUIA: SR. JOSE MIGUEL PIQUER GARDNER

"DISEÑO E IMPLEMENTACION DE UN FRAMEWORK DE DESARROLLO DE APLICACIONES DE VIDEO DIGITAL"

En los últimos años varios estudiantes del DCC han realizado sus memorias de título basadas en video digital. En general esos estudios se han centrado en diferentes formas de envío de video tolerantes a fallas. Todos ellos tuvieron en común el no contar con una plataforma, documentada y funcional, que les permitiera extraer y trabajar video de forma adecuada para alcanzar sus metas.

Por eso al inicio de este trabajo, nos percatamos de la necesidad imperiosa de contar con un conjunto de funciones que permita, en forma concisa y precisa, la manipulación de video digital y sus explotaciones posibles por medio de despliegue y transmisión.

Con la fuerte penetración en el país de la telefonía móvil, entendiendo que los celulares con capacidades de captura y despliegue de video que ya no son un lujo exclusivo de pocos, se decidió extender el trabajo inicial de este informe de modo de permitir comunicar video no solo entre PCs, sino que además extenderlo a los usuarios de dispositivos móviles como los celulares Nokia S60. Los protocolos diseñados para envío de video en esta memoria fueron escritos pensando siempre que el receptor es el más débil en términos de procesador, permitiendo así una mayor disponibilidad de receptores móviles.

Al final de esta memoria se obtuvo un conjunto de funciones y estructuras, agrupadas en bibliotecas de fácil incorporación para aplicaciones de terceras personas. Dichas bibliotecas permiten manipular la captura, transmisión, recepción y despliegue de video en tiempo real, permitiendo a los futuros desarrolladores poder focalizar sus esfuerzos en los algoritmos propios de su tesis, en vez de la construcción de una plataforma de trabajo que ahora, con esta memoria, es una realidad.

Para demostrar empíricamente que las funciones y estructuras escritas funcionan en hardware actual, se escribieron varios programas que usan las bibliotecas ofrecidas, incluyendo los protocolos diseñados y las herramientas que permiten hacer pruebas de velocidad, interconexión entre computadores personales y dispositivos móviles, además de pruebas de rendimiento de extracción y despliegue local.

Esto recién comienza: en el presente trabajo no hemos estudiado nuevas formas de codificar video, sino que hemos cimentado de forma ordenada y documentada la base para poder analizar futuros algoritmos de trabajo en video digital. Esperamos que los futuros desarrolladores agradezcan este esfuerzo pues fue realizado pensado en ellos.

A mi abuelo Renato... quien me enseñó a ser libre.

Agradecimientos

A mi familia quien me ha apoyado todos estos años, pese a que no necesariamente entienden el camino que he seguido, por su cariño incondicional y por permitirme construir mi vida en mis propios términos.

A mi profesora Ivonne Schain quien me enseñó a los seis años a leer y a amar y defender a mi patria. A Mónica Aravena, mi profesora de educación media que me inculcó el gusto por las matemáticas y de quien copié la manera directa de decir las cosas.

A mis amigas Maria Paz Delanouy, Jesús Larrain, Angélica Munchmeyer y Nicole Wild, cada una de ellas en su época, me dieron soporte y consejos que siempre valoraré y llevaré conmigo.

A mis mejores amigos de primer año, a quienes considero hermanos Julio "Rus" Aguirre, Javier "Polle" Guzmán y Claudio "Clapi" Pizarro, gracias a ellos y sus familias por recibirme en sus casas como uno más y por la trascendental ayuda que me dieron en tiempos difíciles.

A Loreto Arenas por darme mi apodo y a su marido William Young que ha sido un fiel amigo desde el comienzo de mi estadía en Beauchef.

A todos quienes participamos en el Boletín Sei, en el Centro de Estudiantes de Ingeniería (CEI), en la Centro Derecha Universitaria (CDU), en el Centro de Desarrollo Estudiantil (CDE) y en tantos otros proyectos.

A la directiva que me acompañó en mi Presidencia del CADCC 2005, Alfredo Cádiz (gracias por tu entrega total al proyecto), Nicolás Dujovne (gracias por alegrar el ambiente), Cristián Cámpora (gracias por tu dedicación y por invitarme a comer tantas veces), Luis Cárdenas (gracias por tu seriedad, por tu calidad moral, y por tus tips en C).

A la directiva que me acompañó en mi Presidencia del CEI 2006, Sergio Courtin (gracias mecánica por ese 80 % de los votos), Andrés Hurtado (gracias por la gran cantidad de cosas que hiciste en nuestra gestión), Daniela Jerez (gracias por tu lealtad, sé que fue difícil defenderme en tu partido), Rodrigo López (gracias por tu apoyo).

A los funcionarios con quienes trabaje en el CEI, Guillermo Mella (gracias por tu alegría), Luis Rivera (por tu apoyo en tareas tediosas), a Jeannette Henríquez (gracias por tu excelente disposición). A Carmen Labraña por su apoyo preciso y útil que varias veces nos permitió salir de problemas.

Al decano de mi facultad, don Francisco Brieva, quien me apoyó y dio consejos de cómo sobresalir y liderar situaciones complejas cuando se está a cargo de mucha gente.

Al ex vicedecano de Beauchef y actual vicerrector económico de la Universidad, don Luis Ayala, con cuyo ejemplo entendí que con disciplina y orden se puede hacer mucho con poco.

A los partidos de la Alianza por Chile, Renovación Nacional (RN) y la Unión Demócrata Independiente(UDI), por haberme apoyado pecuniaria y políticamente durante mis años de dirigente universitario, en especial en mis campañas FECH.

Al senador por Santiago y colega dirigente don Pablo Longueira, por inspirarme con su ejemplo a seguir el camino del servicio público, quien además me apoyó incondicionalmente durante mi Presidencia y permitió recuperar la personalidad jurídica del CEI perdida hace casi 100 años.

A todos los vocales, concejeros, centros de alumnos, dirigentes, funcionarios, académicos y estudiantes que me apoyaron en mis proyectos durante

mi larga vida universitaria, quienes con su entrega me permitieron concretar tantas cosas de las que me siento orgulloso.

A mi profesor guía José Miguel Piquer, por ser ejemplo a seguir, por su calidad humana, su apego a la familia, su forma amigable de relacionarse, sus consejos personales y profesionales, por su apoyo total y por comprender que mi trabajo de dirigente demoró esta titulación mas de lo esperado.

Índice general

Capítulo 1

Introducción

La World Wide Web ha tenido un crecimiento exponencial en los últimos años. Millones de nuevos usuarios llegan año a año, seducidos por el material que encuentran en Internet. Las páginas web y los diversos programas de comunicación existentes están utilizando cada vez con más frecuencia el video digital, para otorgar páginas fáciles de acceder con contenidos más completos, simultáneamente la video conferencia se hace más conocida y utilizada por los usuarios finales.

La transmisión de video está lejos de ser algo trivial: por naturaleza implica un gran ancho de banda, ya que hay que enviar varias imágenes por segundo (en la TV chilena son 30 cuadros por segundo). Si bien las conexiones caseras a Internet han mejorado notablemente en los últimos años, aun no son suficientes para transmisiones crudas de video (que necesitan unos 100Mbps). Por eso se utilizan algoritmos de compresión para poder disminuir la cantidad de información a transmitir.

El otro problema es que por la naturaleza de Internet (red de paquetes,

no de circuitos), la conexión entre hosts no está garantizada con un 100 % up-time, por tanto nuestros algoritmos de compresión y de transmisión tienen que ser tolerantes a fallas.

Lo que se necesita es poder transmitir video con elegancia, es decir que se utilice el menor ancho de banda posible, al mismo tiempo que la pérdida de información no genere mucho impacto.

Para poder desarrollar y trabajar transmitiendo video, es necesario tener un conjunto de funciones a priori, para poder focalizar los esfuerzos en los algoritmos de transmisión y no en los de captura.

1.1. Justificación

Se han desarrollado bastantes tesis e informes de memoria sobre video digital y sus derivados. El tema es lo suficientemente complejo como para proporcionar tema de análisis y estudio por mucho tiempo más.

Pese a que muchos ingenieros han hecho estudios sobre transmisión de video, el tema de la captura siempre ha sido una piedra de tope. No es trivial como obtener 25 o 30 imágenes por segundo de una manera segura y ordenada. Ante eso la mayoría de los trabajos parten de cero, esto es, una parte importante de su tiempo de estudio lo utilizan en la fase de captura de video. Tienen que definir por experiencia propia, qué herramientas son propicias para el trabajo, y cuales son las estructuras y funciones más eficientes. Como la parte de captura no es la parte central de sus estudios, por lo general no se prioriza el desarrollo ni la documentación de esa función.

Eso ha provocado que cuando se quiere trabajar en mejoras en la trans-

misión de video, los ingenieros por lo general terminan adaptando un programa ya escrito. Lo malo de eso es que el código fuente de los programas queda con una estructura que no es precisamente natural al software terminado. No existe certeza de buena parte del código fuente, ya que solo se comprueba su exactitud con pruebas pragmáticas de uso simple. Tampoco hay referencia disponible que explique los conceptos y procedimientos claves para capturar y trabajar video. En fin, lo único que está al alcance del estudiante novato en el área, son un cúmulo de programas que capturan y envían video, pero cuyos códigos fuente son sucios y con poca lógica u orden.

Es fundamental alguna fuente que de manera clara, concisa y precisa, explique las distintas aristas de este tópico y que, al mismo tiempo, indique qué primitivas se tienen que usar. De esa forma los futuros investigadores podrán centrar sus esfuerzos en el desarrollo y perfeccionamiento de los algoritmos en juego, sin frustrar sus intentos en la "selva negra" en que han caído sus antecesores.

1.2. Motivación

En la medida en que la conectividad es cada vez más masiva y las velocidades ofrecidas más baratas y rápidas, se abren oportunidades para usar canales de video que hasta poco tiempo atrás solo podían ser soñadas.

Con la filosofía del software libre a la vuelta de la esquina en todo el mundo, cada vez son más los científicos que comparten su trabajo. Las herramientas GNU proporcionan de manera gratuita, alternativas de desarrollo competentes para todos los estudiantes. Lo que antes solo estaba al alcance

de desarrolladores que pudieran costear frameworks de desarrollo de video, hoy está al alcance de todos con la Api Video4Linux.

El envío de video digital por una red de paquetes como Internet en forma masiva, es una disciplina joven que tiene mucho por desarrollar. Para poder hacer avances en esa área se hace imperioso tener un reglamento que nos muestre cuales son los elementos en juego, y como trabajarlos.

Documentar los pasos claros que permitan lo anterior es necesario para lograr los grandes objetivos. El solo hecho de dar a los que vendrán una explicación de los componentes básicos del video digital, es motivo suficiente para desarrollar este trabajo.

1.3. Objetivos Generales

Estudiar, documentar y programar los procedimientos claves en el trabajo de video digital con la realidad tecnológica actual, generando una estructura de trabajo (framework) que permita enviar, recibir y experimentar con video desde ambiente Linux y en telefonía móvil con dispositivos Nokia S60.

1.4. Objetivos Específicos

- Documentar todos los conceptos necesarios para trabajar en video digital.
- Estudiar acuciosamente las API de captura v4l1 y v4l2.
- Desarrollar desde cero un framework de captura y envío de video digital.

- Programar aplicaciones que permitan visualizar, transmitir y recibir video por red.
- Investigar y programar los dispositivos móviles Nokia S60, para poder interactuar el desarrollo Linux con la transmisión de telefonía móvil.

Capítulo 2

Antecedentes

La naturaleza del video es una secuencia de varias imágenes por segundo de forma continua. Trabajar con video implica una carga de información muy grande, si comparamos con texto o fotografías, donde la información es estática. Para comprender bien como se trabaja el video digital, es necesario entender algunos conceptos y manejar ciertos estándares y procedimientos, de eso trata este capítulo.

2.1. Trabajos previos

En el DCC varios alumnos han realizado sus trabajos de memoria de titulación en captura y transmisión de video. El primero fue Fermín Uribe, quien adaptó una copia de videodog para enviar y recibir video a través de una red local. Los que vinieron después de él reutilizaron ese código para hacer mejoras e incluir más funciones.

2.2. Video4Linux

En linux hay una librería llamada Video4Linux(V4L) que entrega las funciones necesarias para capturar video. Video4Linux tiene dos versiones, la primera (v4l1) que esta deprecada, y la segunda (v4l2) que esta incluida en todos los kernels actuales.

2.2.1. Historia v4l1

El chipset bt878 es uno de los más populares en las tarjetas capturadoras para PC. El driver que se desarrolló para extraer video era el módulo bttv, que permitía extraer video desde aplicaciones en linux(/dev/bttv0). La necesidad de poder utilizar webcams y otras capturadoras desde linux, provocó que se definiera una API de captura bastante simple, llamada Video4Linux, que se introdujo en el kernel de desarrollo al final del ciclo 2.1. No es ningún secreto que Video4Linux (que viene a ser lo que hoy día conocemos como Video4Linux1, o v4l1), heredó gran parte del código del bttv, ya que era la experiencia exitosa del momento, que ayudó a reutilizar líneas de código en la creación de varios drivers.

Sin embargo, lo que en primer momento fue su fortaleza, a mediano plazo fue una limitante seria para los desarrolladores, ya que no había un framework, o una API bien definida, que respetara la diversidad de hardware que existe en el mercado de las capturadoras.

Por tanto lo que sucedió fue que se necesitaba un rayado de cancha más completo para poder ofrecer un puente sólido entre los fierros y el kernel de linux.

2.2.2. Historia v4l2

Debido a los problemas de diseño y de inflexibilidad de v4l1, en agosto de 1998, Bill Dirks propuso una lista de mejoras y se puso a trabajar en ejemplos de drivers y aplicaciones, todo con una buena documentación. Recibió ayuda de más voluntarios, y con el tiempo todo ese esfuerzo se transformó en el reemplazo de v4l1 en vez de un upgrade. Cuatro años después, el trabajo fue aceptado como parte del kernel oficial.

2.3. Color

El color es la percepción asociada a las ondas visibles[1] del espectro electromagnético. Es la impresión que producen en la retina los rayos de luz reflejados y absorbidos por un cuerpo.

El ojo humano contiene tres receptores, cada uno es sensible a casi un tercio del espectro visible. El color que el ojo ve depende de cuanta luz roja, verde y azul sea reflejada, el negro es percibido como ausencia de luz.

El ojo humano entonces no funciona como una máquina para análisis espectral, y gracias a eso, se pueden producir sensaciones de cualquier color (aproximaciones) mezclando rojo, verde y azul.

Las imágenes a color tienen por lo general varios cientos de colores diferentes. Sin embargo, para reproducirlas ya sea en un monitor o de forma impresa, se mezclan solo tres colores, logrando aproximaciones que dada la naturaleza de nuestros ojos son absolutamente tolerables.

Se definen tres propiedades del color:

[1] ondas con longitud entre 400nm a 700nm

Tinte o tonalidad (Hue): es lo que nos permite diferenciar un color de otro, es lo que le da identidad al color: tinte AMARILLO, tinte CAFE, etc. Cuando hablamos del color ROJO, en realidad solo estamos definiendo una de sus cualidades: el tinte. Los programas de edición le asignan generalmente un rango entre 0 y 360.

Valor o luminosidad(Value or Brightness): es el grado de claridad u oscuridad de un color. Un azul mezclado con blanco da como resultado un azul más claro, es decir de un valor más alto. El valor de un color se puede modificar sumándole blanco o negro, el valor nunca modifica el tinte. El azul claro sigue siendo azul, solo que de un valor más alto. Por otra parte, cada color tiene un valor intrínseco, el azul es de un valor más bajo que el amarillo, el rojo más alto que el violeta, etc.

Saturación o Cromaticidad (Saturation): es el grado de pureza o intensidad de un color. Cuando un tinte es puro, es decir, no tiene ninguna mezcla, presenta la máxima saturación. Si a un color, por ejemplo, azul, lo mezclamos con blanco, no solo aumentará su valor sino que disminuirá su pureza (se desaturará). El rango es entre 0 % a 100 %. 0 es sin color, es decir, una mezcla gris entre blanco y negro, y 100 es el color intenso y puro.

2.4. Modelos de color

Un modelo de color es un modelo matemático abstracto, mediante el que se identifica el color mediante tuplas de números, por lo general 3 o 4 (por ejemplo RGB o CMYK).

Si a un modelo de color se le asocia una descripción precisa de sus componentes (condiciones de visualización, dispositivos empleados, etc.) el grupo de colores resultantes se denomina espacio de color (ejemplos: sRGB, AdobeRGB).

Si una pantalla tradicional (CRT) recibe una señal del 50 % del valor máximo, uno esperaría que mostrara una luz con la mitad de la intensidad posible. Sin embargo la relación no es lineal y solo se observaría una luz de casi el 18 % del máximo de intensidad posible. Eso sucede porque los monitores de cátodos tradicionales tienen una respuesta no lineal que es aproximada a la potencia de 2.5 ($0{,}5^{2{,}5} \approx 0{,}18$). El coeficiente de esa función puente entre el pixel obtenido y el desplegado se le llama "Gamma".

El valor de Gamma depende de cada monitor, pero en general es cercano a 2.5. Los monitores de plasma o LCD no necesitan esta corrección y por tanto tienen que re-corregir en caso de recibir una señal para monitor CRT.

2.4.1. Modelo RGB

RGB es un modelo que describe una paleta de colores generados a partir de tres colores primarios: rojo, verde y azul (Red, Blue, Green). Es un modelo de combinación aditiva de colores usados en dispositivos luminosos como los monitores, en donde la suma de los primarios da el color blanco, y la ausencia de ellos el color negro.

El modelo RGB no define exactamente cuales son sus componentes primarios, por lo tanto el rojo base en un dispositivo puede ser muy distinto al de otro. Es por eso que una misma fotografía digital se puede ver distinta en distintos monitores. No es raro entonces que los distintos espacios de color

RGB muestren colores diferentes.

Existen varias implementaciones digitales de este modelo:

RGB24: asigna 8 bits por color, es decir 3 bytes por pixel.

RGB565: asigna 5 bits para el canal rojo y azul, 6 para el canal verde (es decir 2 bytes por pixel).

RGB32: 8 bits por color y 8 bits para el canal alfa que es el cuarto canal usado para transparencia o simplemente descartado.

2.4.2. Modelo CMYK

El modelo CMYK es un modelo de colores sustractivo que se utiliza en la impresión de colores. La mezcla sustractiva es la que se realiza con pigmentos, es decir con pinturas. Al mezclar los pigmentos, la química propia de cada componente mineral absorbe la luz que el ojo humano percibe, o sea al mezclarlos restan luz.

Los colores base son el Cian, Magenta y el Amarillo. El Cian es el opuesto al rojo, lo que significa que actúa como un filtro que absorbe dicho color. El magenta es el opuesto al verde y el amarillo el opuesto al azul.

La ausencia de los 3 componentes bases resulta blanco, y al mezclar los 3 se obtiene el negro. Sin embargo la implementación de este modelo no es ideal, ya que la combinacion de pigmentos no es perfectamente sustractiva, por lo que al mezclar las 3 colores base no se obtiene un negro aceptable. Es por eso que al modelo se le agrega el color negro como base, llamado Key.

2.4.3. Modelo YUV

El ojo humano es mucho más sensible a cambios de brillo que a cambios de color. El modelo YUV separa los colores en un canal de luminosidad (Y), y en dos componentes con información de color (U y V). En caso de usar solo el canal Y obtenemos una imagen en blanco y negro. La televisión análoga usa YUV como base porque así permite que una misma señal la usen los televisores a color y los con pantalla blanco y negro. El canal U es la diferencia de color azul, el canal V la del color rojo, mientras que la información de verde está en su mayor parte en el canal Y junto con el brillo general. A los componentes U y V se les llama componentes de crominancia y por lo general se les aplica una especie de compresión llamada Submuestreo Cromático. La notación del modelo YUV depende de su submuestreo.

El submuestreo se identifica con tres dígitos separados por doble punto. El primer dígito es relativo a la cantidad de muestras, que por lo general es 4. El segundo dígito representa el factor de submuestreo horizontal de U y V con respecto a Y. Si el tercer dígito es igual al segundo, no hay submuestreo vertical y si es cero indica que hay submuestreo vertical a razón de 2:1 de U y V con respecto a Y.

YUV 4:2:2 Usa 16 bits por pixel, ya que el canal Y usa un byte y 2 pixeles continuos, usan el mismo valor U y V. Y0U0Y1V0

YUV 4:4:4 Usa 24 bits por pixel, un byte por componente.

Hay dos maneras de almacenar una imagen en un arreglo de bytes:

PACKED: los componentes YUV de cada pixel van juntos para cada pixel.

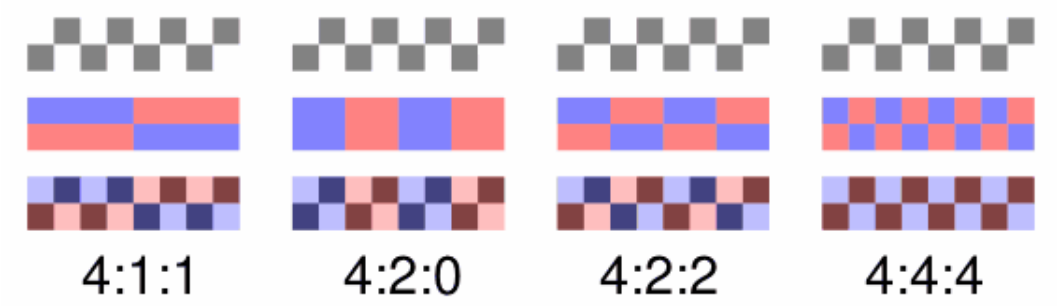

Figura 2.1: Submuestreos YUV

PLANAR: los componentes van separados: primero todos los Y, luego el U y luego los componentes V.

2.5. Formatos de Television Analoga

Existen tres sistemas de codificación para las señales de TV análogas en el mundo. La más popular es la norma Phase Alternating Line (PAL), que permite 25 cuadros por segundo. En Chile se adopto la norma norte-americana National Television System Committee (NTSC) que funciona con 30 cuadros por segundo. Tambien existe la norma Séquentiel Couleur a Mémoire (SECAM) que es usada en Francia, Asia y algunos paises africanos.

2.6. Compresión de video

El video por naturaleza consume mucho espacio, ya que son miles de imágenes por minuto, por tanto en la actualidad se hace menester usar algoritmos de compresión.

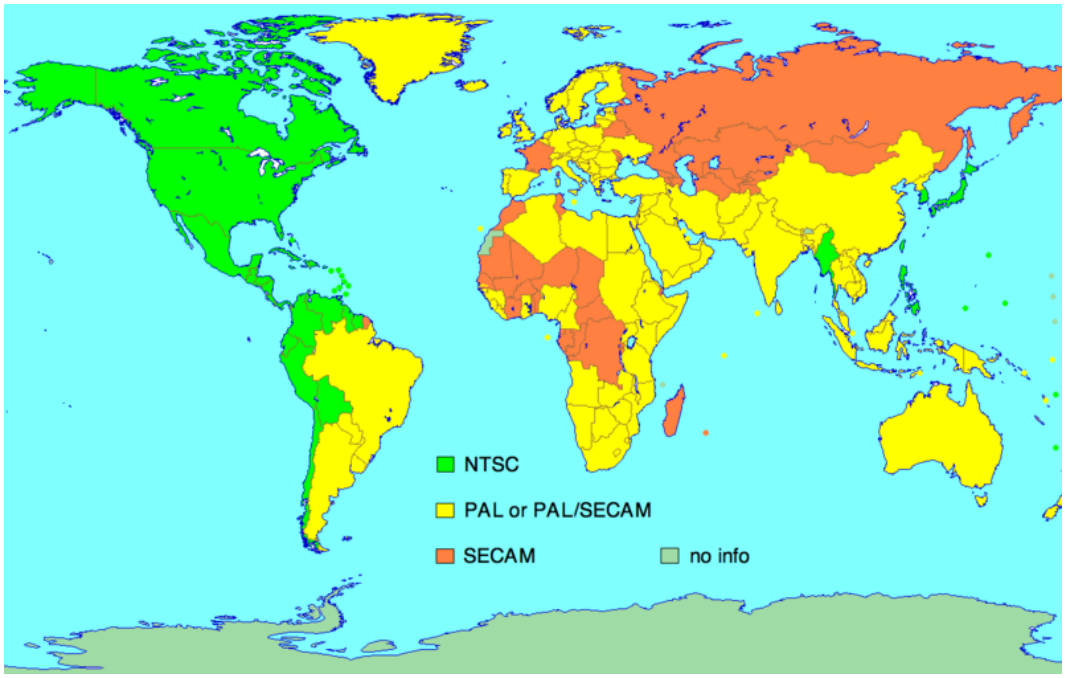

Figura 2.2: Formatos de TV analoga en el mundo

Un mecanismo de compresión es "lossless" cuando el proceso de descompresión retorna el mismo valor original que se comprimió (o sea no hay pérdidas). Por otro lado están los mecanismos "lossy", que sí tienen pérdidas (por muy mínimas que sean, a veces indetectables para el ojo humano).

La gran ventaja de los algoritmos "lossy" es su gran capacidad de compresión (se puede llegar a 1/50 de efectividad) en comparación con los algoritmos "lossless" (en promedio 1/3 de tasa de compresión).

Los algoritmos de compresión buscan reducir al mínimo la redundancia en la información. En las imágenes hay redundancia espacial que es la correlación entre los valores de los pixeles vecinos, hay redundancia espectral que es la correlación que se visualiza en los histogramas de color de una imagen. Y en

caso de video tenemos redundancia temporal que es la correlación entre las imágenes de una secuencia.

2.7. Jpeg

JPEG es un mecanismo de compresión de imágenes. JPEG es la sigla de Joint Photographic Experts Group, que es el nombre original del comité que lo escribió.

JPEG fue diseñado para comprimir imágenes a color o en escala de grises de la vida real, funciona muy bien en paisajes y fotografías, pero funciona tan bien en escritos y dibujos.

JPEG es un mecanismo de compresión lossy, que explota las limitaciones del ojo humano, descartando la información que el ser humano no distingue con facilidad. Por lo anterior JPEG se hizo pensando en comprimir imágenes que serán vistas por humanos.

Una propiedad muy útil del mecanismo JPEG es que se puede variar el nivel de pérdida, por lo tanto al momento de comprimir se puede privilegiar espacio o calidad.

Las etapas para crear una imagen JPEG son:

1. Transformar la imagen de RGB a YUV (si es necesario)
2. Aplicar sub muestreo cromático a la imagen YUV
3. Dividir la imagen en varias imágenes de 8x8 píxeles
4. Aplicar la Transformada Discreta del Coseno a esas imágenes (DCT), una función "lossless" y reversible (solo hay pérdida por el redondeo

pero es pequeño)

5. Cuantizar los valores dividiéndolos por los valores enteros de una tabla de cuantización, los resultados se redondean al entero más cercano. Este paso es lossy, aquí es donde se define la compresión del archivo JPEG, la tabla de cuantización se divide o multiplica antes, para definir el nivel de compresión real.

6. El resultado se recorre en zigzag, y se le aplica una compresión lossless y reversible llamada "run-length encoding" (ejemplo: MMMMMVD-DDDDD, queda : 5MV6D).

7. Luego se aplica "Huffman encoding" para representar de forma más eficiente lo obtenido (también hay una alternativa con un algoritmo aritmético, que tiene mejor compresión pero es más complejo y menos usado). Esta etapa también es "lossless" y reversible.

Para descomprimir una imagen JPEG las etapas se recorren en orden y función inversa. Como se ve la única parte donde hay pérdida de información es la etapa de Cuantificación (5).

2.8. Memoria RAM

Linux segrega su memoria en memoria del kernel y memoria del usuario.

La memoria del kernel está reservada para el kernel y sus módulos (drivers), ésta nunca se copia a disco y usa direcciones reales. En contraste, la memoria del usuario usa direcciones virtuales, puede ser transferida a disco y es donde las aplicaciones trabajan.

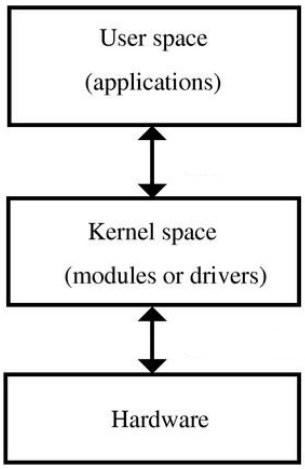

Figura 2.3: Diagrama memoria RAM

2.9. Captura de video

Hay varias maneras en que una aplicación puede obtener las imágenes que un dispositivo está capturando. Es importante notar que los módulos generalmente no implementan todos estos métodos de captura, por lo que es necesario elegir un método soportado.

Método Read: Es la forma más sencilla de capturar video, implica leer secuencialmente desde el dispositivo hacia la memoria de la aplicación. El tradeoff de esta sencillez es el uso del CPU, ya que es el procesador el que tiene que hacer todas las operaciones.

Método Mmap: Este método consiste en mapear la memoria del dispositivo en un espacio de la memoria de la aplicación. Con esto se libera al cpu de tener que estar actualizando las imágenes capturadas. Los dispositivos mas modernos traen varios buffers de captura, lo que permite que

la aplicación trabaje una imagen, mientras el módulo está capturando la siguiente.

2.10. Overlay

Overlay es una técnica usada para mostrar video en una pantalla, saltándose el procesamiento del CPU. Es muy común que las tarjetas capturadoras de video utilicen overlay para desplegar en pantalla completa, evitando saturar el cpu.

Overlay por hardware es una técnica implementada por la mayoría de las tarjetas modernas y permite que las aplicaciones escriban en una parte de la memoria de video, logrando desplegar, escalar y mover imágenes en la pantalla, usando el procesador de la tarjeta de video en vez del CPU central.

2.11. Redes

Internet la red más grande y usada del mundo esta estructurada como un modelo TCP/IP. Dicho modelo describe la red en distintas capas, cada una con sus funciones y características propias. La capa de Transporte tiene dos protocolos que son usados masivamente por los usuarios comunes de Internet.

2.11.1. TCP

TCP (Transmission Control Protocol) es uno de los protocolos de transporte fundamentales en Internet, ya que garantiza que los datos serán entregados a su destino, sin errores y en el mismo orden en que se enviaron.

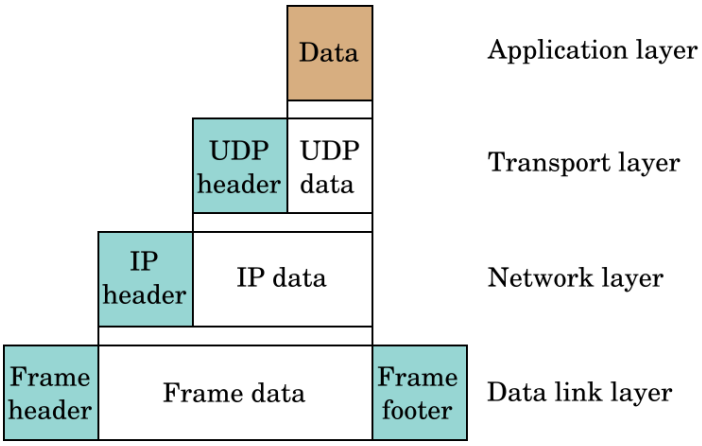

Figura 2.4: Modelo UDP/IP

Algunas aplicaciones basadas en TCP son: HTTP (web), SMTP(mail) y SSH.

2.11.2. UDP

User Datagram Protocol (UDP) es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite enviar información sin que se haya establecido una conexión previa. No tiene confirmación, ni control de flujo, por lo tanto los datagramas enviados pueden llegar en desorden, o simplemente no llegar. Las peticiones DHCP y los queries de zona DNS se hacen con UDP.

2.12. Telefonía móvil

Desde un tiempo a esta parte los celulares han venido evolucionando desde ser grandes y pesados aparatos de telefonía análoga, a la actualidad en que tenemos pequeños y sofisticados aparatos que además de codificar digitalmente la transmisión de voz, también permiten transmisión de datos y captura y despliegue de fotografías y video. Con el tiempo los costos de celulares multimedia con capacidad de conexión han caído fuertemente, permitiendo su uso masivo.

En esta memoria se trabajó con los dispositivos móviles de la línea S60 de Nokia, empresa líder en el mercado actual. Y se usó la red móvil de datos de EntelPCS, específicamente su red de datos Edge[35].

Capítulo 3

Ambiente de Desarrollo Plataforma Linux

Para poder trabajar capturando video, desplegándolo y transmitiéndolo en Linux, es necesario tener bien configurado todo el hardware disponible, a través de un sistema operativo estable, que tenga bibliotecas de desarrollo funcionales al objetivo de este trabajo de memoria. En este capítulo se explica todo el software ocupado en el PC de desarrollo, omitiendo el ambiente de desarrollo Symbian que tiene un capítulo especial.

3.1. Sistema Operativo

El sistema operativo que se usó para desarrollar y para hacer las pruebas es la distribución linux Ubuntu en su versión 7.10. Ubuntu es una distribución basada en Debian, dispone de una amplia gama de programas ya compilados, que permitían instalar de manera rápida y fácil las bibliotecas y herramientas

necesarias. Se usó el kernel versión 2.6.22-14-generic que trae dicha distribución de Ubuntu.

3.2. Módulos Requeridos

El kernel de linux es monolítico pero modular, por lo que hay que cargar en él los módulos de los dispositivos de hardware que queremos usar (drivers).

Para la parte captura cada capturadora tiene su módulo que está anotado en el anexo Hardware. Todos esos módulos van de la mano del módulo "videodev" que es el módulo genérico de Video4Linux.

Para desplegar en pantalla se necesita el módulo para la tarjeta de video. En la actualidad hay dos empresas que tienen la gran mayoría del mercado de tarjetas de video (ATI y NVIDIA).

Las tarjetas ATI que se usó en el PC de desarrollo, tienen un driver de código libre llamado "ati". La empresa sacó un módulo propietario llamado "fglrx". Las tarjetas NVIDIA tienen un driver de código libre llamado "nv". La empresa tiene un módulo propietario llamado "nvidia".

Se comenzó desarrollando con una tarjeta ATI, pero luego se cambio a NVIDIA por mejor calidad del módulo.

Finalmente se optó por trabajar con una tarjeta de video integrada Intel, cuyo driver opensource es soportado oficialmente por Intel. Si bien el desempeño en aceleracion 3d es menor, se eligió por la estabilidad del módulo disponible.

Cuando se elige instalar un modulo propietario para ATI o NVIDIA, el kernel queda "tainted", decisión que en general coarta el soporte de la

comunidad al desarrollador.

Cuando se usa un módulo propietario se gana en velocidad, pero se pierde el conocimiento de la implementación y en la práctica el sistema se pone mas inestable.

3.3. Bibliotecas de desarrollo

El lenguaje de programación que se usó fue C, por su eficiencia y por la amplia gama de bibliotecas que permiten trabajar con recursos de red y de video. Para la parte de captura se usó la biblioteca Video4Linux, en sus dos versiones. Una vez que la captura estaba desarrollada fue necesario programar el despliegue en pantalla de los cuadros de video. Para ello se empezó usando Xlib[17] , que es la librería nativa del sistema XWindows que se usa en linux (hay 2 implementaciones populares: XFree86 y Xorg). Para desplegar las imágenes con las primitivas de Xlib, es necesario tener la imagen en RGB. Esto nos complicó cuando se capturaba en YUV, ya que la transformación de YUV a RGB ocupa mucho tiempo del CPU. Por eso se empezó a trabajar con XVideo[4], que es una extensión de X11 que permite intervenir directamente la memoria de video de la tarjeta. Con eso se logra que la tarjeta de video haga la transformación de YUV a RGB, liberando al CPU de esa carga. Con XVideo se puede desplegar directamente en formato YUV y RGB (en la mayoría de los módulos ati, intel y nvidia).

3.4. Herramientas de desarrollo

Para escribir el código, se uso el IDE Eclipse, con el plugin CDT que permite usar el IDE para código C.

Para editar el código fuente de nuestras aplicaciones, también se usó VI, un editor de texto bastante potente y conocido, además del editor de Eclipse.

Para compilar se uso gcc4.1.3 desplegando todo tipo de warnings, para tener un código ordenado y ajustado al estándar gnu c.

Para enviar las instrucciones a la cámara de video Sony, se usó la librería de código libre libVisca[18] (la cual hubo que adaptar).

Para depurar el código se uso gdb [19], que es una aplicación gnu probada en la materia.

Capítulo 4

Ambiente de Desarrollo Plataforma Symbian S60

Para implementar envió de video a dispositivos móviles, se trabajo con dispositivos Nokia.

La línea S60 de Nokia ofrece un ambiente de desarrollo a bajo nivel con su lenguaje propietario Symbian C++. Existen herramientas gratuitas y comerciales para trabajar en esta plataforma. Manejar los conceptos y particularidades de Symbian C++ no fue un trabajo menor, ya que el desarrollo en ese lenguaje orientado a objetos, obliga al desarrollador a manejar cientos sino miles de conceptos.

4.1. Ambiente desarrollo Symbian

Lo primero que fue necesario, fue registrarse (gratuitamente) en el Foro Nokia, para poder bajar las herramientas[20].

Nokia solo soporta desarrollo de software en ambiente Windows.

Hay un IDE a disposicion, para ello Nokia ha desarrollado a partir de Eclipse uno especial para Symbian C++, llamado Carbide[21] . Carbide viene en 4 versiones, 1 de las cuales es gratuita y basta con ésa para desarrollar. Las versiones pagadas traen extras como hacer debug en el dispositivo móvil, o análisis de rendimiento (profiler) de una aplicación.

Como se desarrolla en Windows, es necesario instalar Active Perl[22] que es usado en los makefile o cadenas de compilación de las aplicaciones.

Hay dos compiladores disponibles en Symbian.

RVCT: es comercial y trae opciones para compilar priorizando tamaño o rapidez [23].

GCCE: es gratuito y es el usado en esta memoria [24].

Por ultimo, tenemos que descargar el SDK pertinente a nuestro dispositivo[25].

El SDK es un paquete que trae consigo todas las librerías necesarias para cierto celular, esto cambia según el modelo del celular que están clasificados en versiones y en FP (Feature Pack). Por Ejemplo el Nokia 6881 es Second Edition Feature Pack 2, el Nokia E50 es Third Edition, y el N95 es Third Edition Feature Pack 1. Elegir un SDK erróneo para nuestros dispositivos hará que nuestras aplicaciones no funcionen en el dispositivo.

Cada SDK trae consigo un emulador del dispositivo, para poder probar nuestras aplicaciones sin exponer al celular, los emuladores son lentos de cargar, pero en general funcionan bien, salvo la parte Captura de Video que no esta implementada en la actualidad.

4.2. Patron Model-View-Controller

Las aplicaciones en Symbian siguen un patrón de desarrollo que permite cambiar fácilmente la interfaz con el usuario. Este patrón divide la aplicación en un controlador que maneja el input del usuario, el modelo que provee la funcionalidad eje del programa y la(s) vista(s) que despliega(n) la información al usuario. Esto se implementa con varios objetos básicos en cualquier aplicación grafica Symbian.

Para el programador común esto puede ser un poco difícil de trabajar en un principio, pues hasta el ejemplo más simple requiere cumplir con este patrón.

4.3. Objetos Activos en Symbian

La forma en que se recomienda programar aplicaciones multitasking en Symbian no es directamente con librerías de multithreading, sino que con objetos activos. Dichos objetos son esenciales para usar funciones asíncronas en nuestro desarrollo symbian. Las funciones básicas de red o las de transformación de imágenes son todas asíncronas y la forma de manejarlas es con objetos activos. Cuando un objeto activo ejecuta una función asíncrona, este queda suspendido hasta que la función termina, en ese momento automáticamente se ejecuta la función RunL() de dicho objeto, permitiendo así retomar el control.

Se pueden tener varios objetos activos en una aplicación, un objeto siempre tiene un Scheduler que lo administra (y permite darle prioridad entre varios objetos activos de una aplicación).

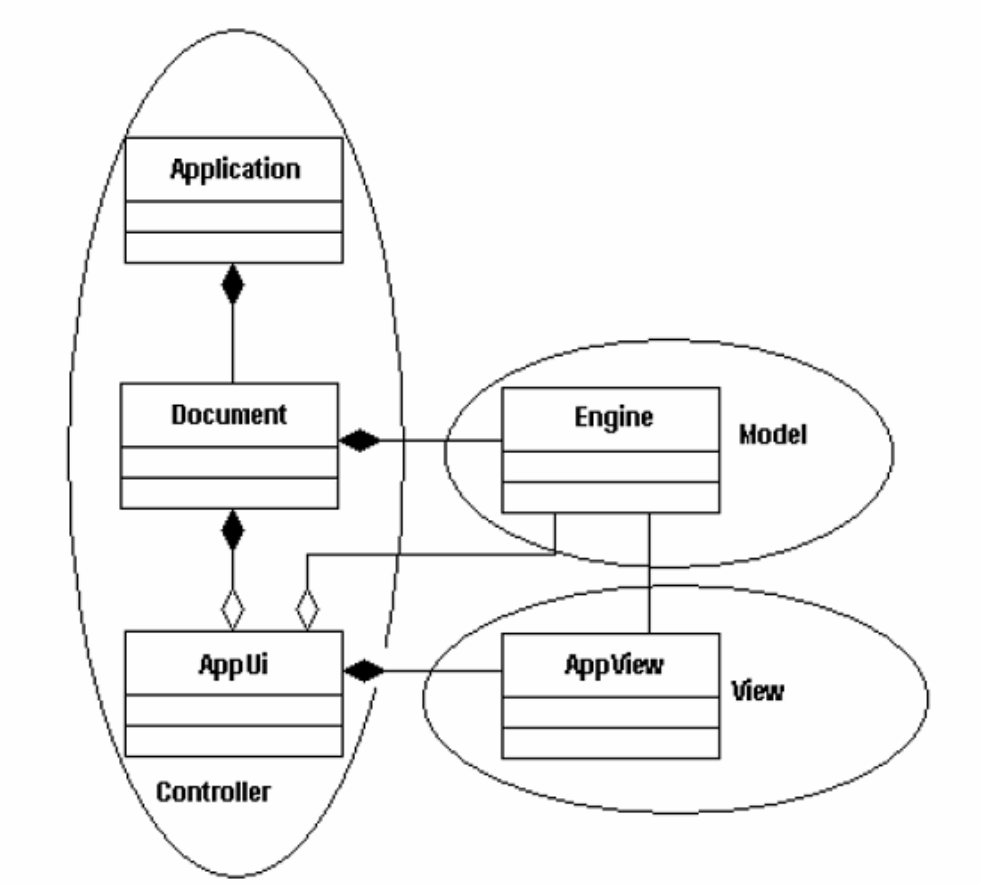

Figura 4.1: Patron Model-View-Controller en Symbian

4.4. Descriptors

Symbian provee de objetos que reemplazan a los tradicionales strings. Estos objetos que hacen de Strings, se llaman descriptors, y difieren por el lugar de la memoria que usan (Heap o Stack), si se pueden modificar o no y si son punteros a la información o si traen la información consigo mismo.

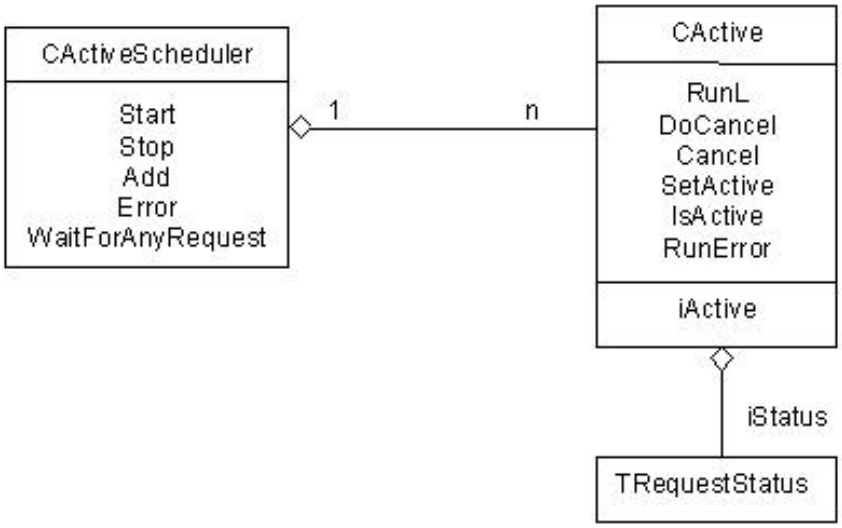

Figura 4.2: Objeto Activo en Symbian

4.5. Errores y nomenclatura

Cuando se escribió Symbian, los desarrolladores vieron que portar el sistema de excepciones de C++ a Symbian iba a salir muy caro en términos de carga, por lo que idearon los "Leaves". Básicamente cuando una aplicación tiene un error, esta genera un "Leave" que puede ser interceptado por un "TRAP".

Los celulares a veces pasan meses sin ser apagados o reseteados, por lo que el uso de memoria en Symbian es crítico. Es por ello que en Symbian existe un "Cleanup Stack", pila donde se meten los objetos mientras se hacen operaciones críticas que puedan generar un "Leave", de esa forma en caso de error el sistema operativo podrá liberar la memoria usada por esos objetos. Es muy común crear los objetos en dos pasos, primero realizando lo que no

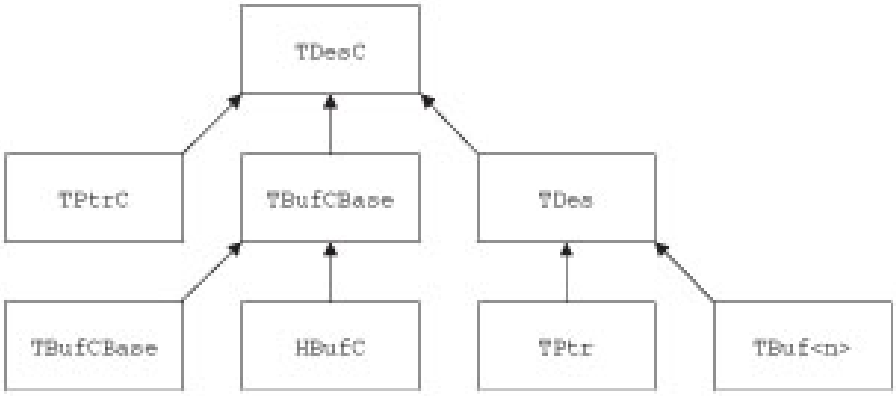

Figura 4.3: Descriptors

puede producir error, y luego lo que puede generar "Leaves" (previo inserto en la pila de limpieza).

La nomenclatura[34] de variables, funciones y clases esta totalmente normada en Symbian, esto nos permite saber muchas cosas solo al mirar los nombres de los ítems en el código, pero inclina aún más la curva de aprendizaje necesario para trabajar en Symbian.

4.6. Captura de imagenes en Symbian

La primera etapa en una aplicación de video es la captura. Video son secuencias de imágenes, por lo tanto para enviar video necesitamos tener acceso a cada una de esas imágenes para enviarlas secuencialmente por red. Para ello Nokia nos ofrece la librería CCamera[26] en sus dispositivos S60 desde la segunda versión FP2. Con CCamera uno puede elegir que formato quiere usar (la cámara tiene que soportarlo eso si), en nuestra aplicación las

imágenes son capturaras en formato EXIF JPEG.

4.7. Red en Symbian

La conexión entre dispositivos se logró a través de redes IP. Cada celular tenía su número ip que era accesible para el otro. Se eligió TCP/IP para estar seguro que las imágenes llegaban y en orden (aunque fuera mas lento). Los sockets en symbian vienen implementados por al Clase RSocket[27].

4.8. Display en Symbian

Para poder desplegar una imagen en pantalla era necesario trabajar la imagen como Symbian Bitmap, que es un formato nativo de Symbian. Sin embargo ya que la captura y envió es sobre buffers jpeg, se necesita transformar en tiempo real desde jpeg a este formato de symbian bitmap, para esa tarea Nokia soporta una librería llamada CImageDecoder[29].

Pero aún nos faltaba resolver el problema del tamaño de las imágenes, porque las pantallas de los celulares son de distinta resolución, por lo tanto una imagen que cubre la pantalla del enviador, puede ser 1/4 de la pantalla del receptor (eso nos sucedió con un Nokia N91 como enviador y un Nokia E50 como receptor). Por lo tanto para redimensionar bitmaps nativos de symbian usamos la clase CBitmapScaler[30].

Capítulo 5

Desarrollo

Para trabajar con video se hizo necesario manejar y comprender bien las API de captura v4l1 y v4l2. Por ello fue necesario estudiar durante varias semanas cada función de dichas bibliotecas. Lo mismo sucedió con las bibliotecas estándares del lenguaje C. También fue vital aprender a usar avanzadamente las herramientas de compilación, de versionamiento, y de depuración que provee GNU.

Una vez que ya se habían recabado y estudiado los conceptos mínimos para empezar a programar, se planificó el orden en que serían implementados los procedimientos.

Una vez que teníamos el diseño y la implementación de las funciones de captura, se precisó primero almacenar las imágenes capturadas en archivos, aprovechando el simple formato PPM, que permite usar un feed RGB24 para ser usado en cualquier editor de imágenes (solo agregando un header simple). Seguros que estábamos capturando bien, se empezó la etapa de despliegue en pantalla.

La idea es que el software desarrollado se pueda usar en la mayor cantidad de sabores de linux, con la menor cantidad de bibliotecas necesarias. Por ello se estudió y eligió la librería nativa del sistema XWindows, XLib, en desmedro de otras librerías de desarrollo X más completas como SDL, Tcl/Tk o Glade.

Con XLib se tuvieron resultados satisfactorios, comprendiendo que las imágenes desplegadas tenían que estar en formato RGB, lo que ponía un impedimento a trabajar en formato YUV, ya que la transformación de RGB a YUV tenía un alto costo para el procesador.

Luego, a petición del profesor guía, se agregó soporte para la API deprecada (v4l1), esfuerzo que fue mucho menor al requerido por la versión actual (v4l2) debido a que su complejidad es órdenes de magnitud menor.

A esta altura se tenían las funciones para poder inicializar los dispositivos con los valores necesarios para capturar video según demanda, pudiendo visualizar la captura en pantalla.

Lo que siguió fue incluir Libjpeg en nuestro framework, ahora la limitante es que la librería original de JPEG solo define funciones para generar archivos jpeg desde buffers RGB en memoria, o para leer desde archivos jpeg a buffer rgb en memoria. Es decir no provee de funciones para generar o extraer desde y hacia buffers en memoria.

Por lo tanto, se adaptó un manager de memoria para libjpeg desarrollado por un estudiante de doctorado[36], que justamente permite trabajar desde y hacia buffers en memoria. Pero además se estudió y entendió la librería de jpeg, para poder generar jpegs tanto desde buffers en memoria yuv o rgb,o para generar buffers en memoria yuv o rgb desde buffers en memoria jpeg.

Libjpeg trabaja con YUV444, por lo que fue necesario escribir una función

YUV444 a YUV422 para poder conectar el output de la librería a un display. A este nivel del desarrollo, estábamos limitados a trabajar en RGB, porque trabajar en YUV imponía hacer la conversión por software al espacio RGB que pedía XLib. Por lo tanto se estudió la librería XVideo, que permite escribir directamente en la memoria de video, logrando que sea el procesador de la tarjeta de video el que efectúe las transformaciones.

Una vez que XVideo fue dominado por el autor, se pudo trabajar con video en cualquiera de los 2 espacios de color. Es importante recalcar que desde que se adaptó libjpeg a nuestro framework, se pudo capturar y desplegar video desde nuestros dispositivos jpeg nativos (orbit, zoran y creative live).

Luego se desarrollaron las funciones para cambiar parámetros a los dispositivos de captura, ya sea movimiento en el caso de la Orbit, o controles de hue, color, brillo y tinte en todos los dispositivos. Se programó de forma genérica, para que el software detectara los rangos que cada módulo proporciona, a fin de evitar seteos corruptos.

Esas funciones de control, se enlazaron con un panel de control implementado con funciones básicas de XLib, que permitió cambiar valores de captura en tiempo de ejecución.

Una vez que teníamos dominados Xlib y Xvideo para despliegar video, y v4l1 y v4l2 para capturarlo, además de tener funciones para transformar imágenes a todos los espacios necesarios y también libjpeg, lo que faltaba era la parte de transmisión.

Para la parte de red, se decidió estructurar todo lo más simple posible, para dar espacio a futuras mejoras o formas de envío.

Se definieron 2 protocolos para envió de video, uno para envió de frames

completas, en que primero se envían 4 bytes que representan el porte de la imagen enviada (esto es importante cuando se trabajan con jpegs, porque cada frame tiene un tamaño distinto), para luego enviar la imagen misma. Con este protocolo el cliente sabe exactamente cuantos bytes recibir para la siguiente frame.

También se definió e implementó un protocolo para envio parcial de los frames, que consiste en dividir la imagen en macro bloques de 8x8 pixeles (para esto es importante que el ancho y la altura sean múltiplos de 8).

Para enviar video en ese protocolo de macrobloques, en cada datagrama (de tamaño modificable) se empieza por cuatro bytes que permite saber en qué frame se está, después viene un byte es el número de macro bloques que vienen en el datagrama(máximo 256), seguidos de las coordenadas de los macro bloques en el frame, para concluir con los pixeles de los macro bloques transmitidos en el datagrama. Un datagrama UDP puede tener hasta un máximo de 65507 bytes de información interna (sin contar headers)[37].

Hasta aquí teníamos un mecanismo que dividía cada frame en varias imágenes de 64 pixeles y los iba empacando y enviando. Pero la ganancia viene dada cuando empezamos a condicionar que macro bloques se enviaban.

Para eso se implementó una matriz de movimiento de macro bloques, esto es una matriz que contiene un valor de cambio de cada macrobloque entre la imagen actual y la anterior. Con esto la aplicación puede saber en tiempo real que partes de la imagen cambió lo suficiente como para ser enviada.

Para calcular ese valor de cambio se optó por la norma valor absoluto entre los distintos componentes de un píxel actual y el anterior. En el caso de trabajar con RGB se promediaban los 3 valores, en el caso de YUV so-

lamente se usaba el valor de luminosidad (los de crominancia se descartan). Como un byte toma valores entre 0 y 255, el máximo cambio posible es la diferencia máxima, por tanto la matriz de movimiento tiene valores para cada macrobloque entre 0 y 255.

De esta forma se pudo calcular en tiempo real la cantidad de movimiento de cada macrobloque entre un frame y otro. Para clarificarlo se escribió una función que imprime en texto una representación de la matriz de movimiento, en donde imprime desde 1 a 6 en la casilla de cada macrobloque según su valor (escala logarítmica).

Se notó que por fallas propias de los sensores de hardware, a veces una secuencia que aunque no representa cambios al ojo humano, si tiene pequeños márgenes de diferencia reales, por lo que una imagen con algo de cambio, según la norma de valor absoluto, puede representar una secuencia sin cambios. Otra característica que se programo fue la forma de recorrer los macro bloques cada vez que se evaluaba que enviar. Para eso se definió una grilla configurable, que permite recorrer todos los macro bloques en varios circuitos equidistantes. Con esto se logra que el empaquetado de los macro bloques sea disperso, y en caso de pérdida del datagrama, no se concentre la pérdida en una sección lineal (y notoria).

Se dispuso de un tiempo de vida de cada macrobloque, es decir en caso de que un macrobloque no sufra cambios por un tiempo (y por ende no se transmite), se autotransmite para estar seguros de que el cliente tenga actualizada la imagen. Se privilegió siempre que el servidor, o sea el que envía video, fuera el que realizara la carga más pesada, para que el desarrollo del cliente no requiriera de gran inteligencia, ya que se limitaba solo a recibir

datagramas o frames completos y desplegarlos en pantalla. Sin embargo luego de cientos de pruebas se comprobó que era vital saber el momento en que se tenían que hacer los refrescos al contenido. El cliente únicamente despliega la nueva frame cuando, recibe un datagrama con macro bloques de un frame posterior. De esta forma se asegura de "pintar" en pantalla cuando ya no falta información para el frame a desplegar.

Todo quedó implementado con código limpio, ordenado y modularizado y se terminó con la inclusión del protocolo Visca, que es un protocolo serial usado para controlar las cámaras de seguridad Sony. No fueron pocas las horas dedicadas para escribir el módulo que otorgara el puente entre la captura y movimiento.

Una vez que teníamos todo lo anterior codificado era el momento de hacer funcionar los distintos dispositivos de captura para ver el desempeño de éstos. Se pudo comprobar que los módulos linux que existen para controlar los dispositivos están implementados de forma bien heterogénea, ya que hay algunos que cumplen muy bien las especificaciones v4l2 o v4l1, mientras que otros solo cumplen algunas partes de la api de captura. Ante esto se aprendió por prueba y error, que hay algunos dispositivos que solo permiten algunas instrucciones, que al solo intentar otras el módulo se bloquea y se tiene que rebootear la máquina.

Como queríamos entregar un producto que permitiera trabajar con captura y envio de video desde múltiples dispositivos de video, al final de esta experiencia, se obtuvo un conjunto de funciones que detallamos a continuación, y la aplicación de estas en las tres aplicaciones que se comentan en el próximo capítulo.

Con esto listo, se implemento el protocolo de envio de frames completas en Symbian, para poder incluir el envio desde y hacia dispositivos móviles. Originalmente el protocolo implementado tenía un header de 2 bytes, lo que permitía frames de tamaño máximo de 65535 bytes, que era suficiente para las imágenes jpeg de 160x120 que capturan los celulares usados en este desarrollo. Sin embargo después se tuvo que aumentar a 4 bytes para poder trabajar con imágenes de mayor tamaño (todas las imágenes a 640x480 en formato YUV o RGB por ejemplo).

Básicamente las aplicaciones para symbian cuando envían frames lo que hacen es capturar nativamente en jpeg, y enviar el frame completo. Cuando la aplicación recibe, transforma ese buffer jpeg a symbian bitmap, lo redimensiona al tamaño de la pantalla del celular y lo despliega.

Se escribieron entonces las funciones del framework, que permiten fácilmente desarrollar aplicaciones de captura, envio y despliegue de video.

5.1. Funciones Captura

Para capturar es necesario inicializar los dispositivos y luego extraer las imágenes.

```
Función que abre el dispositivo para poder interactuar con el.
int cap_open(char * adevice);


Función que setea el formato de imagen, evita cropping,
e inicializa todo lo necesario para empezar a capturar.
int cap_init(int av4l, int amethod, int atv_standard, int awidth,
```

```
 int aheight, int aformat,int ainput, int abytes_per_pixel);

Función que lee próxima imagen.
int cap_next_frame(void ** aframe);
```

5.2. Funciones despliegue Video

Para poder desplegar en pantalla el video recibido es necesario tener funciones que comprendan el protocolo usado (Xvideo o X11).

```
Función que crea ventana.
int dis_create_window(int adisplay, int awidth, int aheight,
 int abytes_per_pixel, char * title);

Función que dibuja frame.
int dis_draw_frame(int adisplay, void * aframe);

Función que extrae la información desde un datagrama a un frame.
int dis_unpack(void * adatagram, void *aframe);
```

5.3. Funciones de Red

Para enviar y recibir el video, fue necesario implementar funciones para el envio de macrobloques y para envio de frame completa.

```
Función que permite a un cliente conectarse a un servidor.
int net_connect(int anet_transport, char * ahost, int aport,
```

```
 int amax_datagram_size);
```

Función que crea un servidor.

```
int net_listen(int anet_transport, int aport, int amax_datagram_size);
```

Función que recibe un datagrama.

```
unsigned int net_receive_datagram(void * adatagram);
```

Función que envia un datagrama.

```
unsigned int net_send_datagram(void * adatagram, int asize);
```

Función que recibe un frame completo.

```
int net_receive_frame(void * aframe);
```

Función que envia un frame completo.

```
unsigned int net_send_frame(void * aframe, int asize);
```

Capítulo 6

Programas

Usando las funciones desarrolladas en este trabajo de memoria, se implementaron tres aplicaciones en linux, y dos usando las bibliotecas de Nokia para dispositivos moviles S60. El envio y despliegue de video en tiempo real, es posible en todas las combinaciones posibles: desde S60 a PC, desde PC a S60, desde S60 a S60 y desde PC a PC.

6.1. xprimo

Esta aplicación permite visualizar video desde un dispositivo video4linux, pudiendo desplegarlo localmente en pantalla con XVideo o XLib. Permite cambiar en tiempo real valores de la captura como entrada elegida, brillo,tinte,etc. La aplicación se probó sin parar durante días sin problemas.

6.2. primo-server

Este programa no utiliza ninguna librería de despliegue, solamente captura video y lo envía a un cliente que lo escuche. Como no necesita librerías gráficas, es posible utilizarlo en un PC antiguo dispuesto para transmitir.

Para enviar a dispositivos móviles se transforma a jpeg cada frame que captura ya sea en formato yuv o rgb.

Gracias a la funcion print_diff() se va viendo en tiempo real la variacion de movimiento frame a frame (permite saber que macrobloques se envian).

6.3. xprimo-client

Esta aplicación no captura video, solo despliega la información que recibe por red. Siempre se trató al cliente como la parte menos inteligente de este framework, de forma de permitir implementaciones en dispositivos móviles de poca capacidad. Esta aplicación en particular recibe la informacion en yuv, rgb o jpeg, y hace las conversiones necesarias para poder desplegar en pantalla.

6.4. sprimo-server

Esta es una aplicación escrita en Symbian C++ para dispositivos móviles S60. Lo que hace es configurar la cámara que trae el celular para capturar frames jpeg de 160x120 pixeles y espera hasta que un cliente se le conecte al puerto 7000.

Una vez hecha la conexión empieza a enviar frame x frame al cliente.

6.5. sprimo-client

Esta es una aplicación escrita en Symbian C++ para dispositivos móviles S60. Esta aplicación se conecta a un servidor de video, y empieza a recibir frame por frame.

Cada vez que recibe el frame jpeg, lo transforma a un Symbian Bitmap, que inmediatamente despliega en la pantalla del celular.

6.6. vscan

Durante el desarrollo de la investigación, fue necesario configurar y probar diferentes dispositivos de captura, para ello se hizo preciso identificar si el módulo del hardware era v4l2 o v4l1, además de su device correspondiente. Para otorgar esa información de forma oportuna se escribió una pequeña aplicación que da un listado de los dispositivos de captura configurados en el sistema operativo, junto con indicar que api de captura soporta el módulo.

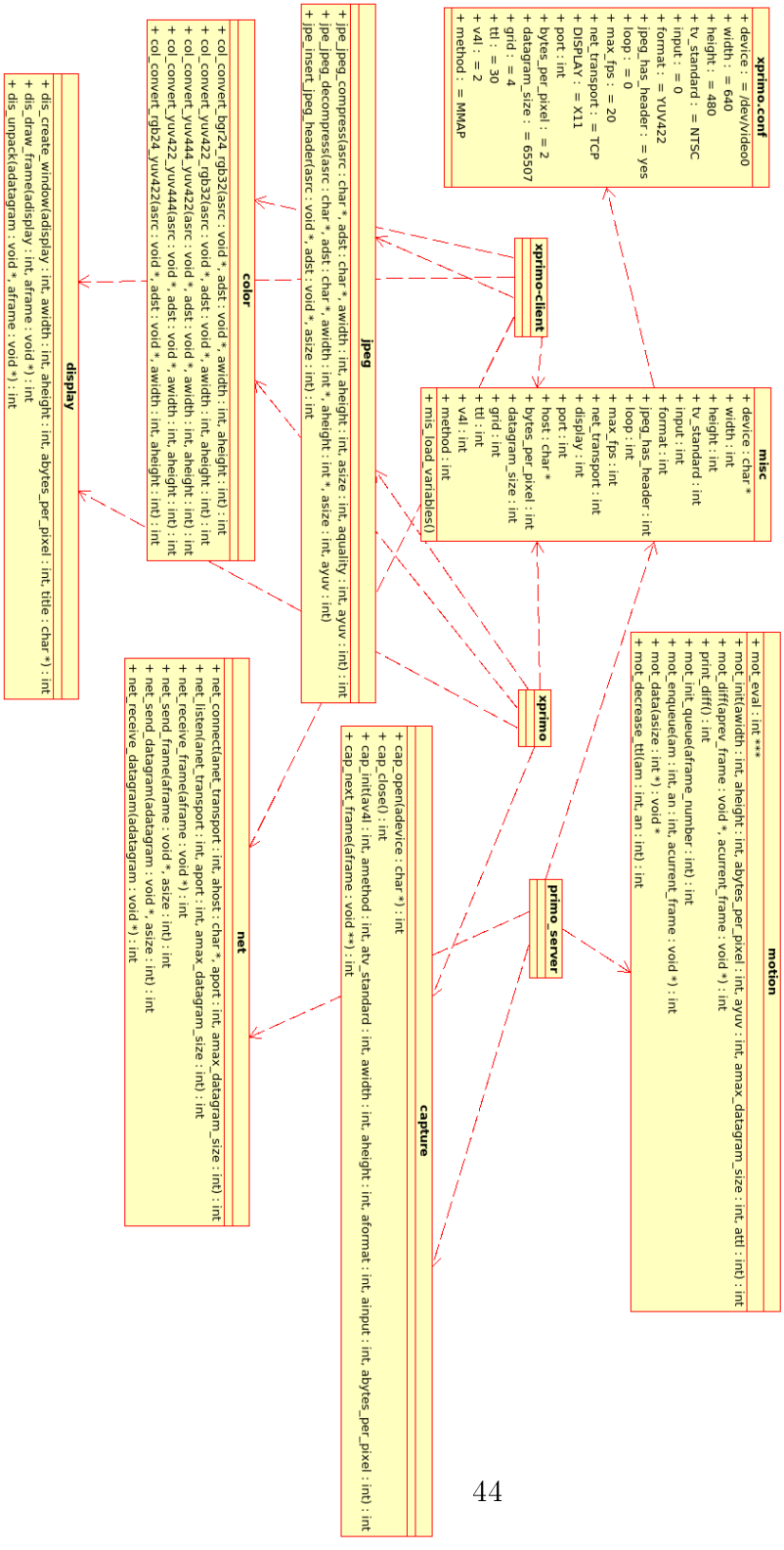

Figura 6.1: Modulos del framework

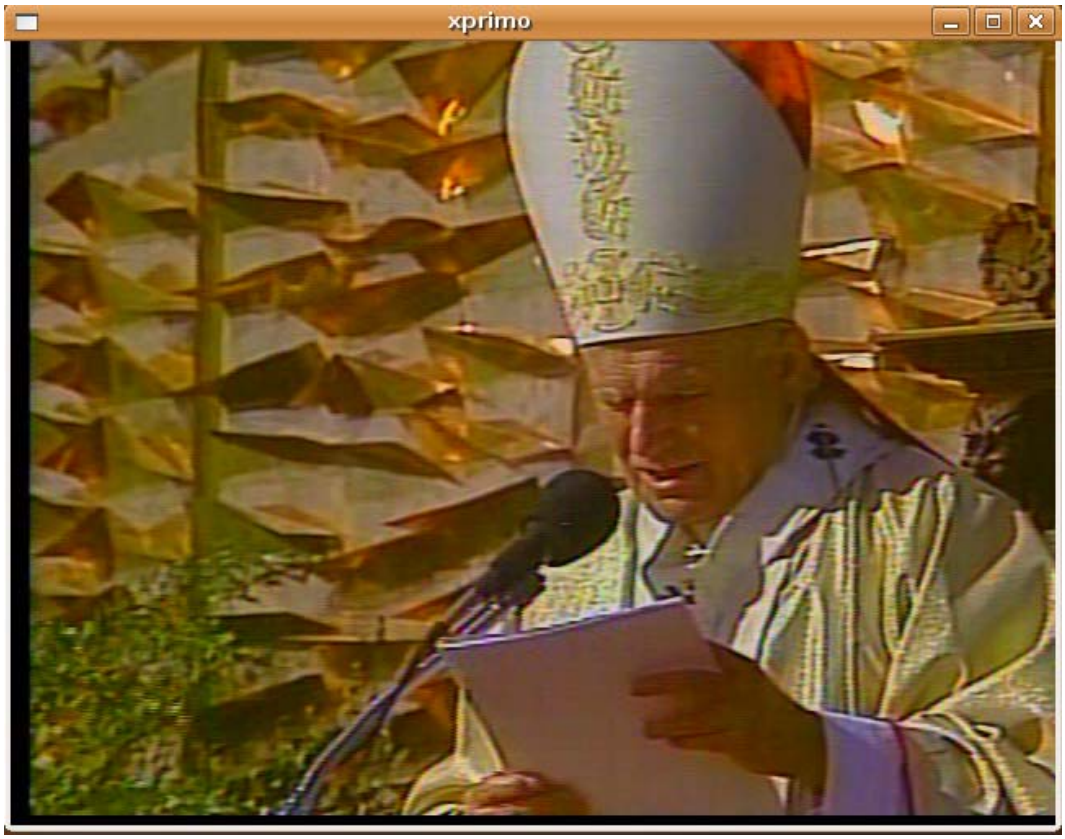

Figura 6.2: xprimo capturando y desplegando video a 640x480

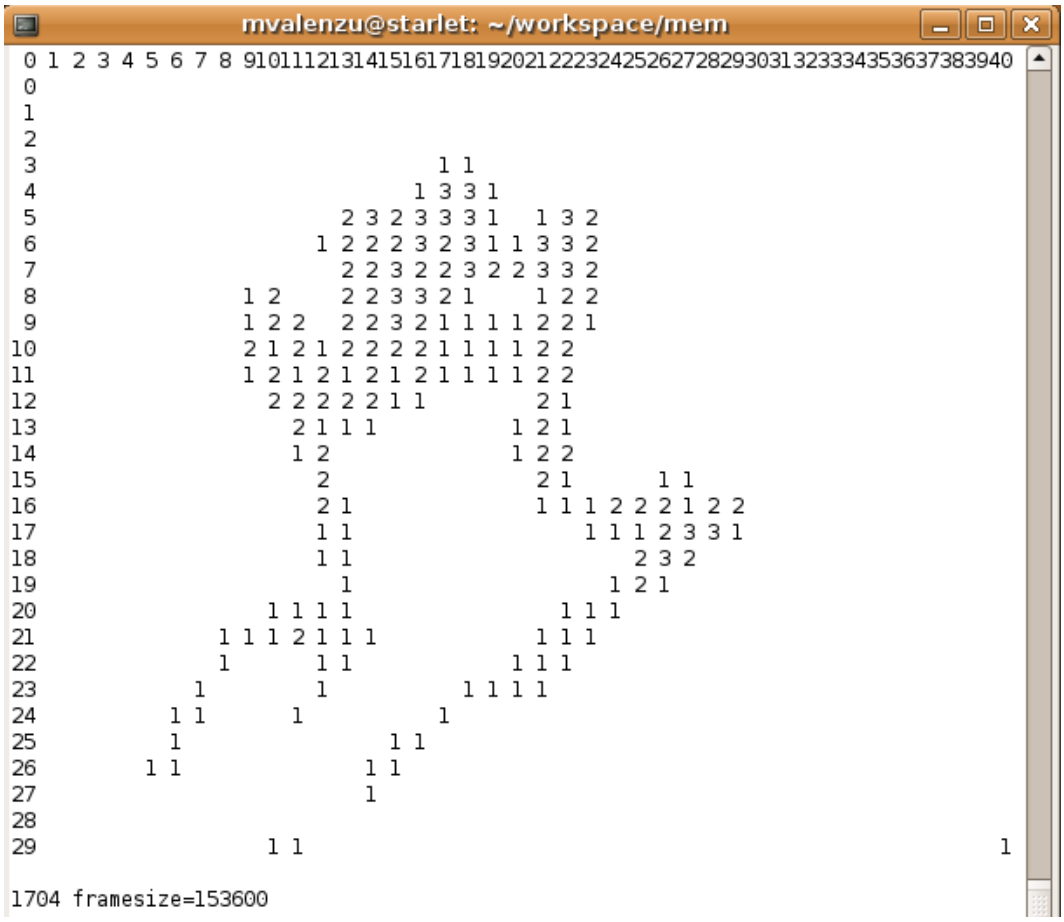

Figura 6.3: primo-server enviando video a 320x240

Figura 6.4: xprimo-client recibiendo y desplegando video a 320x240

Figura 6.5: sprimo-server enviando video a 160x120

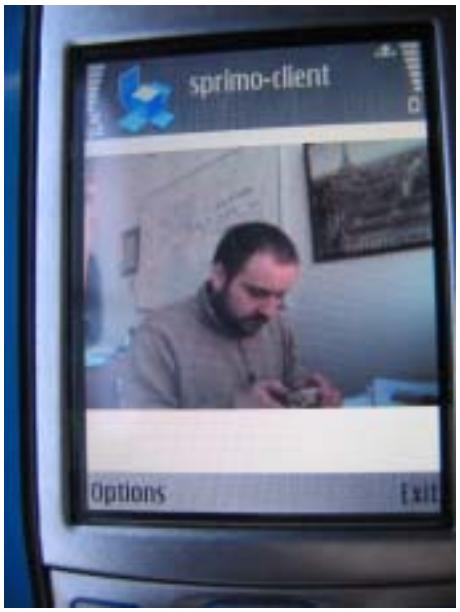

Figura 6.6: sprimo-client recibiendo video a 160x120

Capítulo 7

Pruebas de captura, despliegue y transmisión

Una vez que se tienen los programas funcionando, se pueden hacer pruebas de velocidad y entrega, para ello se envian varias frames de video con constante movimiento. Cada cierto numero de frames desplegadas se calculaba la velocidad en frames por segundo. Lo que sigue son los resultados de dichas pruebas.

Las pruebas se realizan en varias series para estar seguros de que es un rendimiento promedio, y no resultados esporadicos que suceden en caso de atochamientos en la red u otros problemas temporales.

7.1. Pruebas xprimo

Lo que se buscaba era ver si influía la forma de extraer video (READ o MMAP), la forma de desplegarlo (X11 o XVIDEO), el formato elegido

Opciones	1	2	3	4	5
MMAP/RGB24/X11	29.9841	29.9702	29.9704	29.9703	29.9704
READ/RGB24/X11	14.9503	14.9852	14.9852	14.985	14.9847
MMAP/YUV422/X11	29.8984	29.9709	29.9678	29.9709	29.9715
READ/YUV422/X11	14.7794	14.9846	14.8374	14.9099	14.549
MMAP/RGB24/XVIDEO	29.9979	29.9706	29.9703	29.9704	29.9704
READ/RGB24/XVIDEO	14.9528	14.9853	14.9852	14.9851	14.9852
MMAP/YUV422/XVIDEO	29.9714	29.9704	29.9703	29.9701	29.9706
READ/YUV422/XVIDEO	14.9546	14.9852	14.9852	14.9852	14.9852

Cuadro 7.1: Frames por Segundo a 320x240 pixeles, series de 100 frames

(RGB24 o YUV422) y si el porte de la imagen era relevante (640x480 o 320x240).

Lo primero que se observa es que el método de extracción de video si es relevante, ya que con el método READ obtenemos como la mitad de rapidez que se puede con el método MMAP. Lo anterior era esperable ya que en el primer método es el cpu el que copia bit a bit cada frame, y el modulo en el kernel tiene que esperar a que sea leído el frame completo, antes de poder renovar el buffer, mientras que en MMAP solo se copia un puntero al segmento de memoria donde esta el frame actual, y el driver o modulo tiene mínimo 2 buffers de captura para poder renovar un buffer mientras se usa el otro.

Un frame en YUV422 pesa 2/3 de lo que pesa un frame RGB24 sin embargo no se ven diferencias de velocidad solo por cambios en esa componente, sino por la necesidad de cambiar de espacio de color.

Opciones	1	2	3	4	5
MMAP/RGB24/X11	29.5448	29.9705	29.9691	29.9667	29.9655
READ/RGB24/X11	14.1465	14.0005	14.1418	14.5443	14.413
MMAP/YUV422/X11	17.1077	17.3299	17.3551	17.3176	17.3139
READ/YUV422/X11	7.45543	7.49375	7.492	7.48906	7.49636
MMAP/RGB24/XVIDEO	29.9623	29.9717	29.9689	29.971	29.9697
READ/RGB24/XVIDEO	14.349	14.6919	14.9848	14.2039	14.9112
MMAP/YUV422/XVIDEO	29.9615	29.9727	29.9702	29.9696	29.9712
READ/YUV422/XVIDEO	14.9721	14.9854	14.9852	14.9851	14.9852

Cuadro 7.2: Frames por Segundo a 640x480 pixeles, series de 100 frames

Cuando se despliega en XVIDEO, cada frame tiene que ser transformada por el cpu a YUV422, ese paso se omite si captura en YUV422, pero al capturar en RGB24 la aplicación gasta varios ciclos del cpu en transformar.

El caso más claro se ve cuando trabajamos a 640x480 pixeles, es decir cada frame pesa 4 veces lo que pesa una de 320x240 pixeles. Si elegimos desplegar en X11 es decir que cada frame hay que transformarla en RGB32 para su despliegue final, el costo de transformar desde YUV422 a RGB es notorio, si a eso le sumamos que la captura sea con READ, obtenemos framerates tan bajos como 7 fps.

7.2. Pruebas desde primo-server a xprimo-client

Siempre enviando frames completas, se probó la velocidad de envío en frames rgb y yuv.

Opciones	1	2	3	4	5
YUV422/640x480/LOCALHOST	29.9619	29.9871	29.962	29.9959	29.9719
RGB24/640x480/LOCALHOST	29.9778	29.99	29.951	29.9799	29.9806
RGB24/320x240/LOCALHOST	29.9778	29.9729	29.9702	29.9691	29.9936
YUV422/320x240/LOCALHOST	29.9629	29.9795	29.9696	29.9737	29.9654
YUV422/320x240/100MB	29.9728	29.96	29.9855	29.9449	29.9616
RGB24/320x240/100MB	29.9836	29.9712	29.9771	29.9705	29.9725
YUV422/640x480/100MB	19.0767	19.3599	18.9777	19.0079	19.21
RGB24/640x480/100MB	12.8321	12.7491	12.7425	12.6964	12.7434
YUV422/640x480/WIFI	3.78322	3.86941	3.82625	3.8688	3.83675
RGB24/640x480/WIFI	2.46042	2.45639	2.48917	2.46579	2.46583
RGB24/320x240/WIFI	10.0118	10.0819	10.0453	9.99222	9.95411
YUV422/320x240/WIFI	15.204	14.7965	14.6154	14.9768	15.1668

Cuadro 7.3: Frames por Segundo, series de 100 frames

7.3. Pruebas desde primo-server a sprimo-client

Se quería probar la integración desde captura linux a celular nokia. Para ello se capturó con una capturadora jpeg pci zoran, los frames jpeg de tamaño

variable en torno a los 37Kbytes y de 160x120pixeles, eran recepcionados por el celular y desplegados en tiempo real. La velocidad del celular es la red de datos Edge disponible en todo el país.

Opciones	1	2	3	4	5
JPEG/TCP	2.17031	2.21836	2.21744	2.11253	2.06452

Cuadro 7.4: Frames por Segundo a 160x120 pixeles

El cuello de botella es la conectividad del celular, que si bien es lenta, hay que notar que prontamente estará disponible en todo Chile una red de datos 3G que permitirá más frames por segundo.

7.4. Pruebas desde sprimo-server a xprimo-client

Para probar las distintas resoluciones de envío de sprimo-server y no estar sujetos a las restricciones que nos daba la conectividad Edge, se usó como enviador un celular Nokia N91 que permite conexión Wifi.

Las series[1] tuvieron que ser mas cortas ya que la estabilidad de la aplicación Symbian es inversamente proporcional al tamaño de la imagen transmitida. Las pruebas demuestran que con mayor ancho de banda móvil, la frecuencia se triplica fácilmente.

[1] (Tipo Imagen)/(Resolucion)/(Frames x serie)

Opciones	1	2	3	4	5
JPEG/160x120/40	7.06126	6.47334	6.67112	6.79709	6.59118
JPEG/640x480/10	1.15848	1.1421	1.06649	1.04154	1.09694
JPEG/800x600/8	0.885512	0.895245	0.969907	0.841442	0.877619
JPEG/1024x768/7	0.738521	0.735968	0.755425	0.734691	0.73968
JPEG/1600x1200/3	0.36426	0.353363	0.389717	0.368502	0.36042

Cuadro 7.5: Frames por Segundo con coneccion Wifi

Capítulo 8

Trabajo a futuro

El desarrollo de esta memoria, dejó aplicaciones capaces de enviar y recibir video en computadores con XWindows, además de dispositivos S60. Lo que no se pudo hacer por falta de tiempo fueron clientes de video para otros dispositivos que se tenían estudiados, como la consola Sony PSP y las tablet PC que usan Maemo como la N800.

La consola psp de Sony, tiene conexión wifi inalámbrica norma B (11mb/s), con ese ancho de banda es posible transmitir video en su pantalla de 300x240 pixeles. Hay bibliotecas en C para desarrollar el cliente, existe la alternativa de usar Lua, que es un lenguaje interpretado hecho en C.

Maemo es una distribución linux para tablet pcs, Nokia ya tiene dos modelos (N800 y N810) que se han vendido bastante en el mundo. Se alcanzó a instalar un ambiente de desarrollo para Maemo, pero faltó tiempo para portar nuestro framework a esa plataforma.

Para los browsers, existe una librería GNU llamada Ming, que permite generar contenido swf (flash). Con un cliente flash nuestro servidor de video

podría alimentar directamente a las páginas web con video en vivo.

Quedó pendiente la implementación del protocolo de envío de macrobloques en Symbian, si bien está avanzado dicho desarrollo, esta vez faltó estudiar la forma de poder desplegar imágenes YUV o RGB directamente en S60.

Capítulo 9

Conclusiones

Luego de estudiar los conceptos necesarios para trabajar video digital, queda claro que la disciplina del video no es menor y que en la actualidad se estresan los componentes de los PC para tener altas resoluciones. El trabajo de desarrollar desde cero una aplicación de captura, despliegue, envío y recibo de video deja una experiencia que de todas maneras servirá para los futuros desarrolladores del área, por tanto se ha puesto especial ahínco en documentar todos los pasos y conceptos involucrados en este proyecto.

Durante el desarrollo de este trabajo de memoria, varias veces se instaló la desesperanza y frustración, ya que lo que estaba escrito en papel, no se reproducía en la experimentación cotidiana. Fue así en base al trabajo diario durante meses, que se pudo comprobar una y otra vez que los módulos del kernel para captura de video están lejos de ser perfectos y que incluso las herramientas de programación GNU tienen problemas en versiones estables.

La tarea que consume más tiempo en la creación de funciones de captura y despliegue de video, es la de descubrir con qué parámetros trabajar cada

capturadora, ya que hay importantes deficiencias en el rotulado de las tarjetas pci y en la implementación de los módulos del sistema operativo que controlan dicho hardware. Es así como no es de extrañar que en el mercado encontremos tarjetas capturadoras que no especifican el chipset que usan, o cámaras web que no indiquen que resolución y frecuencias entregan nativamente, en vez de promesas que solo son posibles por interpolación hecha en software.

Con Video4Linux2 se obtiene una API de captura mucho más ordenada y poderosa que la anterior, permitiendo manejar una infinidad de dispositivos de la manera que el desarrollador necesite, y no con la lógica minimalista que ofrecía la versión original. Al ser los módulos escritos por voluntarios, no se ha podido erradicar los módulos escritos para Video4Linux1 del kernel estable, ante la falta de módulos V4L2 que administren dichos dispositivos. Esto implica que una gran cantidad de dispositivos en la actualidad solo pueden ser trabajados usando la primera version de Video4Linux, pese a que se ganaría mucho en rendimiento, orden y escalabilidad de funciones, si existieran solo módulos V4L2.

Pero la panacea no está asegurada solo con usar la nueva versión de la API de captura, se requiere que los desarrolladores de los módulos sean más rigurosos, ya que pudimos constatar que en tests de uso prolongado, varios módulos empiezan a fallar, por lo que hacen necesario el reseteo de las aplicaciones. También está pendiente que los módulos soporten todas las opciones descritas en la API, y no que solo estén implementados los formatos de video que son más usados, como así también que exista siempre la opción de extraer por más de un método el video.

La gran mayoría de las tarjetas gráficas disponibles en el mercado vienen

soportadas de facto en las distribuciones de Linux, por lo que es recomendado trabajar en YUV422, ya que es común poder desplegarlo directamente en la memoria de video de las tarjetas. Sin embargo en caso de tener que desplegar en terminales más antiguas que no cuenten con una tarjeta gráfica con XVideo soportado, es importante que las aplicaciones que se escriban, contemplen la opción de trabajar usando las bibliotecas nativas de X11, para que sin importar el administrador de ventanas que se ocupe, se pueda visualizar las imágenes.

Al término de este informe, Symbian domina el mercado de smartphones, pero se anuncia una fuerte guerra tecnológica cuando se masifique el uso de la plataforma que Google está patrocinando, Android. Con Symbian C++ podemos estrujar las capacidades de los dispositivos móviles, pero no sabremos si es mejor que Android hasta que podamos desarrollar en ambas plataformas y hacer pruebas de rendimiento, para comparar con métricas ciertas.

Entendiendo que la magia que involucra la captura y transmisión de video es un viaje que recién comienza con los conceptos y ejemplos que recorren este trabajo. Se termina este informe con la conciencia tranquila de que se logró acumular en un mismo documento el conocimiento suficiente para nivelar y dar un punto de partida a los futuros ingenieros en video.

Bibliografía

[1] *Brian W. Kernighan, Dennis M. Ritchie*, The C Programming Language, Second Edition, 1988.

[2] *Tomás Bautista, Tobias Oetiker, Hubert Partl, Irene Hyna, Elisabeth Schlegl, Cristian Bravo, Pedro Vallejos*,Una Descripciíon de LaTeX2ϵ, 1997.

[3] *Brian Hall*, Beej's Guide to Network Programming Using Internet Sockets, 2005.

[4] *David Carver*, X Video Extension Protocol Description Version 2, 1991.

[5] *Robert Mecklenburg*, Managing Projects with GNU Make 3rd Edition, 2004.

[6] *Raymond Westwater, Borko Furht*, Real-Time Video Compression Techniques and Algorithms, 1997.

[7] *Andreas Uhl, Andreas Pommer*, Image and Video Encryption, 2005.

[8] *Mohammed Ghanbari*,Standard Codecs: Image Compression to Advanced Video Coding, 2003.

[9] *Marco A. Peña Basurto, José M. Cela Espín*, Introducción a la programación en C, 2000.

[10] *Brian Gough*, An Introduction to GCC, 2004.

[11] *Mark Burgess, Ron Hale-Evans*, The GNU C Programming Tutorial, 2002.

[12] *Cliff Wootton*, A Practical Guide to Video and Audio Compression, 2005.

[13] *Paul Coulton, Reuben Edwards, Helen Clemson*, S60 Programming A Tutorial Guide, 2007.

[14] *Jo Stichbury*, Symbian OS Explained Effective C++ Programming for Smartphones, 2004.

[15] Video4Linux(1) Api Specification, `http://www.linuxtv.org/downloads/video4linux/API/V4L1_API.html`

[16] Video4Linux2 Api Specification, `http://v4l2spec.bytesex.org/spec-single/v4l2.html`

[17] X Window System protocol client library, `http://en.wikipedia.org/wiki/Xlib`

[18] libVISCA, `http://damien.douxchamps.net/libvisca/`

[19] GDB: The GNU Project Debugger, `http://www.gnu.org/software/gdb/`

[20] Nokia Forum, `(http://www.forum.nokia.com/`

[21] Carbide, http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide_cpp/

[22] Active Perl, http://www.activestate.com/store/activeperl/download/

[23] RealView® compilation tools, http://wiki.forum.nokia.com/index.php/RVCT

[24] GNU Compiler Collection for Embedded, http://wiki.forum.nokia.com/index.php/GCCE

[25] S60 Platform SDKs for Symbian OS, for C++, http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html

[26] Class CCamera, http://www.symbian.com/developer/techlib/v70sdocs/doc_source/reference/cpp/onboardcameraref/ccameraclass.html

[27] Class RSocket, http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/reference/cpp/SocketsClient/RSocketClass.html

[28] Class CFbsBitmap, http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/reference/cpp/Bitmaps/CFbsBitmapClass.html

[29] Class CImageDecoder, http://www.symbian.com/developer/techlib/v70sdocs/doc_source/reference/cpp/ImageConverterLibrary/CImageDecoderClass.html

[30] Class CBitmapScaler, http://www.symbian.com/Developer/techlib/v70sdocs/doc_source/reference/cpp/BitmapTransform/CBitmapScalerClass.html

[31] *Stephen Randy Davis*, C++ For Dummies 5th Edition, 2004.

[32] *Francis Glassborow*, You Can Program in C++, 2006.

[33] *Ben Collins-Sussman, Brian W. Fitzpatrick, C. Michael Pilato*, Version Control with Subversion, 2006.

[34] Symbian Naming Conventions, http://www.symbian.com/developer/techlib/v70sdocs/doc_source/DevGuides/EssentialIdioms/NamingConvs.guide.html

[35] Red de datos Edge EntelPCS, http://www.entelpcs.cl/GPRS-EDGE/internetmovil_gprs_index.iws

[36] Parallelizing the Condensation Algorithm for Visual Tracking, http://beowulf.lcs.mit.edu/18.337-2002/projects-2002/amay/partracker/doc/index.html

[37] UDP — Internet User Datagram Protocol, http://docs.hp.com/en/B9106-90013/UDP.7P.html

Apéndice A

Hardware

En este capítulo documentados todo el hardware que se usó durante el desarrollo de esta memoria.

A.1. Capturadoras PCI

Las tarjetas de TV más populares en los PC, usan el bus PCI, que está disponible en todas las tarjetas madre modernas.

A.1.1. bttv

Las tarjetas basadas en el chipset bt878 son por lejos las más populares en el mundo GNU. El módulo bttv es el mejor documentado e implementado de todos los módulos v4l2.

```
01:07.0 Multimedia video controller: Brooktree Corporation Bt878 Video Capt
```

```
$ modinfo bttv
filename:       /lib/modules/2.6.20-16-generic/kernel/drivers/media/video/bt8xx/bt
license:        GPL
author:         Ralph Metzler & Marcus Metzler & Gerd Knorr
description:    bttv - v4l/v4l2 driver module for bt848/878 based cards
srcversion:     DB1C6565C3A1D81C944A576
```

A.1.2. saa7134

Este chipset es una alternativa barata a bt878. El módulo actual es v4l2 y funciona sin problemas, pero trae un bug notorio, en que la entrada Televisión está cambiada por la entrada S-Video.

```
01:0a.0 Multimedia controller: Philips Semiconductors SAA7130 Video Broadcast Deco
```

```
$ modinfo saa7134
filename:       /lib/modules/2.6.20-16-generic/kernel/drivers/media/video/saa7134/s
license:        GPL
author:         Gerd Knorr <kraxel@bytesex.org> [SuSE Labs]
description:    v4l2 driver module for saa7130/34 based TV cards
srcversion:     E10BA5B5322155CB78FB041
```

A.1.3. zoran

Este chipset viene incluido en capturadoras profesionales de video. Es mucho más caro que las capturadoras anteriores, y entrega imágenes en JPEG (por hardware).

```
01:09.0 Multimedia video controller: Zoran Corporation ZR36057PQC Video cut
```

```
$ modinfo zr36060
filename:       /lib/modules/2.6.20-16-generic/kernel/drivers/media/video/z
license:        GPL
description:    Driver module for ZR36060 jpeg processors v0.7
author:         Laurent Pinchart <laurent.pinchart@skynet.be>
srcversion:     223614B5458CD50785209A5
```

A.2. Webcams usb 1.0

El mercado está lleno de cámaras web baratas, que entregan video de baja resolución. Los sensores que traen no tienen buena respuesta a cambios de luz y la velocidad de captura es menor que la obtenida por las capturadoras PCI.

A.2.1. logitech usb 1.0 webcam

Esta cámara es bien conocida por el mundo Linux, ya que tiene un modulo v4l1 disponible hace muchos años.

```
Bus 003 Device 004: ID 046d:0870 Logitech, Inc. QuickCam Express
```

```
$ modinfo c-qcam
filename:       /lib/modules/2.6.20-16-generic/kernel/drivers/media/video/c
license:        GPL
```

```
description:    Colour QuickCam for Video4Linux v0.05
author:         Philip Blundell <philb@gnu.org>
srcversion:     5FFEE104A7815A5B0EFFF6D
```

A.2.2. btc webcam

Esta cámara web es de las más baratas que existen en el mercado. La calidad de las imágenes capturadas es bastante mala y es lenta.

```
Bus 003 Device 005: ID 093a:2468 Pixart Imaging, Inc. Easy Snap Snake Eye WebCam
```

```
$ modinfo gspca
filename:       /lib/modules/2.6.20-16-generic/kernel/ubuntu/media/gspcav1/gspca.k
license:        GPL
description:    GSPCA/SPCA5XX USB Camera Driver
author:         Michel Xhaard <mxhaard@users.sourceforge.net> based on spca50x dri
srcversion:     AEAEB91F002DC1E41760970
```

A.2.3. pc380 webcam

Esta cámara web es de tamaño reducido, ya que fue hecha para ser usada por notebooks. Se pudo empezar a usar en el mundo GNU cuando se creó el módulo sn9c102.

```
$ modinfo sn9c102
filename:       /lib/modules/2.6.20-16-generic/kernel/drivers/media/video/sn9c102/
license:        GPL
```

```
version:        1:1.27
description:    V4L2 driver for SN9C10x PC Camera Controllers
author:         (C) 2004-2006 Luca Risolia <luca.risolia@studio.unibo.it>
srcversion:     FE7E2CC8486A09BDD2CBEA9
```

A.3. Webcams usb 2.0

Hace algunos años empezaron a salir al mercado unas cámaras web con sensores más potentes, que tienen mejor reacción a cambios de luz y permiten resoluciones hasta 640x480 a 15 cuadros por segundo.

A.3.1. orbit webcam

Esta cámara es lejos la más cara de todas las webcams que se probaron. Además de la alta resolución que permite, trae un micrófono incorporado y motores que le permiten moverse.

El módulo uvcvideo permite capturar imágenes YUV o JPEG directamente, además de moverla. Los frames JPEG que entrega el módulo no vienen con sus headers íntegros, por lo que hay que parcharlos antes de trabajarlos.

```
Bus 002 Device 004: ID 046d:08cc Logitech, Inc.
```

```
$ modinfo uvcvideo
filename:       /lib/modules/2.6.20-16-generic/kernel/ubuntu/media/usbvideo
license:        GPL
description:    USB Video Class driver
```

```
author:         Laurent Pinchart <laurent.pinchart@skynet.be>
srcversion:     41D4E2FBB38757CD7CEF23E
```

A.3.2. creative webcam

Esta cámara es la nueva línea de las webcams creative! El módulo se congela por unos segundos cada vez que se quiere interactuar con este dispositivo. El módulo actual es v4l1, pero las próximas versiones saldrán en v4l2.

```
Bus 001 Device 003: ID 041e:4051 Creative Technology, Ltd
```

```
$ modinfo gspca
filename:       /lib/modules/2.6.20-16-generic/kernel/ubuntu/media/gspcav1/gspca.k
license:        GPL
description:    GSPCA/SPCA5XX USB Camera Driver
author:         Michel Xhaard <mxhaard@users.sourceforge.net> based on spca50x dri
srcversion:     AEAEB91F002DC1E41760970
```

A.4. Cámaras Análogas

Las capturadoras de video PCI necesitan recibir feeds de video, para eso se usan cámaras analógicas.

A.4.1. Sony CCD-TR705

Esta cámara la proporcionó el grupo de sistemas del DCC, está en mal estado ya que no se puede grabar con ella, la única función que usamos de ella es la captura de video, que se entrega por la entrada S-Video (de muy buena calidad).

A.4.2. Sony EVI D70

Esta cámara captura colores nítidos, y trae zoom y motores para moverse. Es muy completa, ya que es configurable por un control remoto, o vía RS232 (usando el protocolo visca).

A.4.3. SMal Camera Network

Esta cámara es una network-cam, pero para este trabajo solo se utilizó su salida análoga de video.

A.5. Celulares

Para el desarrollo Symbian, se usaron dos celulares Nokia S60 Tercera Edicion.

A.5.1. Nokia E50

El Nokia E50 es un celular delgado y liviano, que solo permite tener conectividad a la red Edge (disponible en todo el país). Se uso para ser el receptor de video, es decir para probar la implementación de sprimo-client.

A.5.2. Nokia N91

El Nokia N91 es un celular pesado y lleno de extras. Para empezar tiene una cámara que permite capturar videos y fotografías en varias resoluciones. Además trae 4 gigabytes de memoria, especialmente otorgados para guardar canciones. Fue muy práctico porque además de conectividad edge, permite usar hotspots wifi para tener velocidades de acceso mas rápidas.

Apéndice B

Api v4l1

La primera versión de Video4Linux es bastante sencilla, de hecho la documentación de la API son cerca de 7 páginas tamaño carta. En este capítulo describimos esta librería brevemente.

B.1. Abrir y cerrar dispositivos

Para capturar video desde un dispositivo lo primero que tenemos que hacer es abrirlo. Para ello se utilizan las librerías estándar de C:

```
int fd;
char * devname="/dev/video0";
fd = open (dev_name, O_RDWR, 0);
```

B.2. Consulta de capacidades

Luego de abrir le preguntamos al dispositivo qué puede hacer. La estructura video_capability es llenada con la información que proporciona el módulo del dispositivo.

```
struct video_capability cap;
ioctl(fd,VIDIOCGCAP,&cap);
printf("Nombre: %s\n",cap.name);
printf("Channels: %d\n",cap.channels);
printf("Audios: %d\n",cap.audios);
printf("Maxwidth: %d\n",cap.maxwidth);
printf("Maxheight: %d\n",cap.maxheight);
printf("Minheight: %d\n",cap.minheight);
printf("Minwidth: %d\n",cap.minwidth);
```

B.3. Entradas

Para las tarjetas PCI necesitamos elegir si queremos capturar de la entrada de TV, video in, SVIDEO in, etc.

Por lo general 0=TV, 1=video in, 2=svideo in. El módulo saa7134 actual tiene un bug que permuta la entrada TV con SVIDEO.

Si bien la estructura es video_channel, esto no es para cambiar los canales de TV (canal 13, 7, etc) sino con el canal de entrada. Los módulos de las cámaras webcam solo proveen de 1 canal de entrada.

```
Cambiar canal de entrada:
```

```
struct video_channel chan;
chan.channel=0; /* elegir TV */
ioctl(fd,VIDIOCSCHAN,&chan);

Preguntar cual es el canal de entrada:

struct video_channel chan;
ioctl(fd,VIDIOCGCHAN,&chan);
printf("Canal de entrada actual: %d\n",chan.channel);
```

B.4. Estándares

V4L1 solo trabaja con 3 tipos de estándares de TV análoga.

VIDEO_MODE_PAL Sistema pal.

VIDEO_MODE_NTSC Sistema NTSC, usado en Chile.

VIDEO_MODE_SECAM Sistema SECAM

B.5. Controles

La estructura video_picture nos permite consultar y setear valores para los valores de hue, colour (ahí se demuestra el origen británico de los programadores de v4l1) y contrast.

Consulta los valores actuales:

```
struct video_picture pic;
ioctl(fd,VIDIOCGPICT,&pic);
printf("Brillo actual: %d\n", pic.brightness);

Setear un valor:

struct video_picture pic;
pic.brightness=32500; /*setear brillo a la mitad*/
ioctl(fd,VIDIOCSPICT,&pic);
```

B.6. Formatos de captura

La estructura video_picture nos permite consultar y setear valores para los valores del formato de la imagen con que se captura. Específicamente la estructura tiene el campo palette que es la que define en que espacio de color haremos la captura.

```
Ejemplo 1:
struct video_picture pic;
pic.palette = VIDEO_PALETTE_RGB24; /* setear captura en modo RGB24 */
ioctl(fd,VIDIOCSPICT,&pic);

ejemplo 2:
struct video_picture pic;
```

```
pic.palette = VIDEO_PALETTE_YUV422; /* setear captura en modo YUV422 */
ioctl(fd,VIDIOCSPICT,&pic);
```

B.7. Captura metodo read

La captura con el método read se hace simplemente leyendo desde el file descriptor. El tamaño de la imagen depende de la resolución y del tipo de imagen elegido (RGB24 son 3 bytes por pixel, YUV422 son 2 bytes por pixel). La variable buffer_length tiene que contener el porte del frame.

```
char * buffer;
int buffer_length;
read(fd,buffer,buffer_length);
```

En buffer queda almacenada la imagen recién capturada por el dispositivo.

B.8. Captura método mmap

Para capturar en mmap con v4l1 tenemos que ir recorriendo en forma secuencial los buffers de la interfaz, para eso definimos un arreglo buffers[] que fue "mmapeado" con los buffers de la interfaz.

```
struct video_mmap v_mmap;
// numero de buffer en la capturadora, son minimo 2
int n_buffers=0; // primer buffer
```

```
v_mmap.frame=n_buffers;
// deja disponible el buffer
ioctl(fd,VIDIOCSYNC,&v_mmap);

// aqui la imagen capturada esta disponible para la aplicacion

// terminada la lectura del buffer, se da la orden de que se actualice
ioctl(fd,VIDIOCMCAPTURE,&v_mmap);
n_buffers++;
// itero entre todos los buffer de captura
if(n_buffers>=mbuf.frames) n_buffers=0;
```

Apéndice C

Api v4l2

La segunda versión de Video4Linux es bastante más compleja, la documentación de esta api son 255 páginas. Al momento de escribir esta memoria, la especificación de v4l2 es el Draft 0.24.

C.1. Abrir y cerrar dispositivos

Para capturar video desde un dispositivo lo primero que tenemos que hacer es abrirlo. Para ello se utilizan las librerías estándar de C:

```
int fd;
char * devname="/dev/video0";
fd = open (dev_name, O_RDWR, 0);
```

C.2. Consulta de capacidades

Lo primero que hacemos antes de capturar es preguntar lo que el dispositivo es capaz de hacer. La estructura v4l2_capability tiene los campos char driver [16], card [32], bus_info [32] y int version,capabilities.

```
struct v4l2_capability cap;
ioctl (fd, VIDIOC_QUERYCAP, &cap))
printf("Nombre del dispositivo: %s\n",cap.card);
```

Aplicando OR al campo capabilities con estas definiciones y otras, es que consultamos al dispositivo.

V4L2_CAP_VIDEO_CAPTURE El dispositivo soporta captura.

V4L2_CAP_VIDEO_OUTPUT El dispositivo soporta salida de video (no es usual).

V4L2_CAP_VIDEO_OVERLAY El dispositivo soporta overlay.

C.3. Entradas

Para las tarjeta PCI necesitamos elegir si queremos capturar de la entrada de TV, video in, SVIDEO in.

Por lo general 0=TV, 1=video in, 2=svideo in

```
Setear:
int valor=0 // selecciona entrada TV
ioctl(fd,VIDIOC_S_INPUT,&valor);
```

```
Consultar:
int valor;
ioctl(fd,VIDIOC_G_INPUT,&valor);
printf("La entrada actual es la numero: %d\n", valor);
```

C.4. Estándares

V4L2 soporta una gran cantidad de estándares de televisión (PAL_B,PAL_G,PAL_H, NTSC americano y japonés, SECAM_B, SECAM_D, y muchos más).

```
Setear para la entrada actual:

v4l2_std_id std_id;
std_id = V4L2_STD_NTSC_M;
ioctl (fd, VIDIOC_S_STD, &std_id))

consulta estandar para entrada actual:

v4l2_std_id std_id;
ioctl (fd, VIDIOC_G_STD, &std_id))
if (std_id == V4L2_STD_NTSC_M) printf("Esta seteado modo NTSC");
```

C.5. Controles

En v4l2 el tema de los controles es mucho más complejo que en v4l1. Hay controles de usuario (user controls) que son los típicos controles que conocemos (como los de v4l1) entre otros:

V4L2_CID_BRIGHTNESS

V4L2_CID_CONTRAST

V4L2_CID_SATURATION

V4L2_CID_HUE

Esta API permite enumerar los controles disponibles, agruparlos en grupos (por ejemplo el grupo MPEG, con todos los controles para trabajar en MPEG) y también permite controles extendidos para ciertos dispositivos que necesitan más controles que los básicos. Así las aplicaciones pueden desplegar menues de control que sean pertinentes a las capacidades del dispositivo.

Con la estructura v4l2_querycontrol consultamos los rangos de los controles. Y con la estructura v4l2_control los seteamos.

```
int maximum, minimum;
struct v4l2_control v4l2control;
struct v4l2_queryctrl v4l2querycontrol;

Consultar cotas:
v4l2querycontrol.id =V4L2_CID_BRIGHTNESS;
ioctl(fd,VIDIOC_QUERYCTRL,&v4l2querycontrol);
```

```
maximum=v4l2querycontrol.maximum;
minimum=v4l2querycontrol.minimum;

Setear control:
v4l2control.id =V4L2_CID_BRIGHTNESS;
v4l2control.value=100;
ioctl(fd,VIDIOC_S_CTRL,&v4l2control);
```

C.6. Formatos de captura

V4L2 define muchos tipos de formato para trabajar. Muchos módulos implementan mal las especificaciones, por lo que no es raro que un dispositivo entregue BGR24 en vez de RGB24.

```
struct v4l2_format fmt;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_BGR24;
ioctl(fd,VIDIOC_S_FMT, &fmt); /* setear modo bgr24 */
```

El ioctl VIDIOC_G_FMT funciona de forma similar, pero llena la estructura fmt con el formato vigente. El módulo v4l2 en caso de no soportar dicho formato, seteará el formato que él cree es más parecido al solicitado.

C.7. Captura método read

La captura con el método read se hace simplemente leyendo desde el file descriptor. El tamaño de la imagen depende de la resolución y del tipo de

imagen elegido (RGB24 son 3 bytes por pixel, YUV422 son 2 bytes por pixel).
La variable buffer_length tiene que contener el tamaño del frame.

```
char * buffer;
int buffer_length;
read(fd,buffer,buffer_length);
```

En buffer queda almacenada la imagen recién capturada por el dispositivo.

C.8. Captura método mmap

En v4l2 encolamos un buffer mmapeado para que el módulo lo actualice.
Para poder leer una imagen desde el dispositivo debemos deencolar el buffer.

```
struct v4l2_buffer buf;
ioctl(fd,VIDIOC_DQBUF, &buf) // decolo el buffer actual para trabajarlo

// aqui trabajo el buffer actual con mi aplicacion

ioctl(fd,VIDIOC_QBUF,&buf); // libero el buffer para que el dispositivo lo actuali
```

Informe escrito en LaTeX