UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

DESARROLLO DE UN SISTEMA DE ESTIMACIÓN DE MAPAS DE PROFUNDIDAD DENSOS A PARTIR DE SECUENCIAS REALES DE VIDEO 3D

EDUARDO SÁNCHEZ ROMERO

2014

Título del Proyecto: *Desarrollo de un sistema de estimación de mapas de profundidad densos a partir de secuencias reales de vídeo 3D*

Alumno:	Eduardo Sánchez Romero
Tutora:	Iris Galloso Guitard
Ponente:	Claudio Feijóo González

Tribunal

Presidente:	Carmen Sánchez Ávila
Vocal:	Asunción Santamaría Galdón
Secretario:	Claudio Feijóo González
Suplente:	Pablo Benítez Giménez

Fecha de lectura y defensa:

Calificación:

Resumen del Proyecto

Este proyecto fin de carrera describe el desarrollo de un sistema de estimación de mapas de profundidad densos a partir de secuencias reales de vídeo 3D. Está motivado por la necesidad de utilizar la información de profundidad de un vídeo estéreo para calcular las oclusiones en el módulo de inserción de objetos sintéticos interactivos desarrollado en el proyecto *ImmersiveTV*.

En el receptor 3DTV, el sistema debe procesar en tiempo real secuencias estéreo de escenas reales en alta resolución con formato *Side-by-Side*. Se analizan las características del contenido para conocer los problemas a enfrentar. Obtener un mapa de profundidad denso mediante correspondencia estéreo (*stereo matching*) permite calcular las oclusiones del objeto sintético con la escena. No es necesario que el valor de disparidad asignado a cada píxel sea preciso, basta con distinguir los distintos planos de profundidad ya que se trabaja con distancias relativas.

La correspondencia estéreo exige que las dos vistas de entrada estén alineadas. Primero se comprueba si se deben rectificar y se realiza un repaso teórico de calibración y rectificación, resumiendo algunos métodos a considerar en la resolución del problema. Para estimar la profundidad, se revisan técnicas de correspondencia estéreo densa habituales, seleccionando un conjunto de implementaciones con el fin de valorar cuáles son adecuadas para resolver el problema, incluyendo técnicas locales, globales y semiglobales, algunas sobre CPU y otras para GPU; modificando algunas para soportar valores negativos de disparidad.

No disponer de *ground truth* de los mapas de disparidad del contenido real supone un reto que obliga a buscar métodos indirectos de comparación de resultados. Para una evaluación objetiva, se han revisado trabajos relacionados con la comparación de técnicas de correspondencia y entornos de evaluación existentes. Se considera el mapa de disparidad como error de predicción entre vistas desplazadas. A partir de la vista derecha y la disparidad de cada píxel, puede reconstruirse la vista izquierda y, comparando la imagen reconstruida con la original, se calculan estadísticas de error y las tasas de píxeles con disparidad inválida y errónea. Además, hay que tener en cuenta la eficiencia de los algoritmos midiendo la tasa de cuadros por segundo que pueden procesar. Observando los resultados, atendiendo a los criterios de maximización de PSNR y minimización de la tasa de píxeles incorrectos, se puede elegir el algoritmo con mejor comportamiento.

Como resultado, se ha implementado una herramienta que integra el sistema de estimación de mapas de disparidad y la utilidad de evaluación de resultados. Trabaja sobre una imagen, una secuencia o un vídeo estereoscópico. Para realizar la correspondencia, permite escoger entre un conjunto de algoritmos que han sido adaptados o modificados para soportar valores negativos de disparidad. Para la evaluación, se ha implementado la reconstrucción de la vista de referencia y la comparación con la original mediante el cálculo de la RMS y PSNR, como medidas de error, además de las tasas de píxeles inválidos e incorrectos y de la eficiencia en cuadros por segundo. Finalmente, se puede guardar las imágenes (o vídeos) generados como resultado, junto con un archivo de texto en formato csv con las estadísticas para su posterior comparación.

Palabras clave:

Estereoscopía, 3DTV, visión artificial, correspondencia estéreo, *stereo matching*, imagen, vídeo, calibración, geometría epipolar, rectificación, correspondencia densa, mapa de disparidad, mapa denso, función de coste, métodos locales, optimización global, GPU, *block matching*, *belief propagation*, *graph cuts*, evaluación de algoritmos.

Agradecimientos

Como diría un buen amigo, « Pues esto es. Esto es el proyecto fin de carrera. » Porque todo lo que comienza tiene su final. Con este documento termina una etapa para comenzar otra nueva, la vida de ingeniero (o eso espero), que tanto esfuerzo ha costado alcanzar. Pero antes, quiero agradecer a todos aquellos que se han cruzado en mi camino, o han subido a mi autobús (como se prefiera), en las diferentes etapas vividas hasta ahora. Algunos continúan el viaje junto a mí mientras que otros se bajaron en alguna parada, pero todos han aportado algo a lo que soy.

En primer lugar a mi familia, sobre todo a mi madre, que me ha dado la vida y le debo todo, y a mi hermana que siempre nos acompaña. A mis abuelos, que van a ver a su primer y único nieto con carrera. A mis tíos y primos de ambos bandos por haber estado ahí cuando se les ha necesitado. Un abrazo fuerte a todos.

Por otra parte, algunos amigos que me acompañaron durante el colegio: Juancar, Víctor S., Manu, Dani, Cristóbal; y aquellos con los que todavía mantengo relación cercana: Tito, con sus chistes malos; José Luis, impulsivo y cabeza loca que dentro de nada se nos hace papá (junto a Cristóbal hacían el dúo joscris*); Laura Barragán, diciendo las cosas tal y como son; Revi, muy silencioso y discreto; Jorge, que Magisterio no es una carrera seria y Víctor Dela, de vida libertina que me da ejemplo de lo que no se debe hacer (con cariño).*

A los amigos de la infancia y adolescencia de la plaza, con los que viví en el mundo callejero y a los del instituto en bachillerato con los que pasé grandes ratos, algunos de los cuales seguimos saliendo de fiesta: Mario Lázaro, Mario Arranz, Javi Valcárcel y compañía. A los amigos del barrio que alguna vez estuvieron: Marcos, Alex, Víctor, Javi, Alba, Amanda; y a los que siguen ahí, que grandes risas nos echamos: Lorena, Bea, María Urbanos, María Vallano, Laura, Tamy, Eva, Sara Marín, Angela y Dani, ¡que dentro de nada nos vamos de boda! Por supuesto, otros amigos con los que he compartido y comparto ratos de distracción: Jorge, Sandra, Irache y Vicky. Y por último a los informáticos, que se unieron en los últimos tiempos: Jorge, Manu, Sergio, Javi y Valen, con los que ha habido momentos divertidos.

A Miriam, una persona especial con la que he compartido el último año de mi vida, gran parte de los cuales he dedicado a escribir este proyecto, que me ha ayudado a desconectar del estrés y que gracias a ella he aprendido que en la vida hay cosas buenas a parte de lo malo. Con ella he descubierto que tengo mucha paciencia y por todo ello, merece mi respeto y cariño.

En teleco hay algunos con los que comienzas el camino y conoces en las primeras clases: Héctor, Mónica, Marta, Luis Úbeda y cía. Otros comenzaron, pero se quedaron en el camino: Sheila, Pablo Martín; y a otros te los vas encontrando por el camino y se convierten en grandes compañeros: Luis Picó, Guillermo Doncel, David G. Regodón, Bea Catalán, Marta Delgado y su grupillo. Pero al final encuentras grandes amigos con los que compartir viajes y experiencias inolvidables. A Eva, mi segunda hermana, que siempre ha estado ahí y me ha ayudado cuando lo he necesitado, hemos compartido agobios, estrés y clases aburridas. A Pablo, uno de los mejores amigos que he tenido siempre, al principio pensaba que no iba a tratar mucho con él pero luego ha sido casi inseparable, es una de las personas más inteligentes que conozco y tiene un gran corazón, es un bonachón. Dani, con su filosofía y visión particular de la vida. Las inolvidables conversaciones en el coche de Dani te hacen reflexionar sobre las relaciones y estupidez humana. Las eternas conversaciones de política (y otros temas trascendentes) entre Dani y Pablo son míticas. Juan, compañero inseparable de Dani (en paseos y carreras) con el don de la ubicuidad, está en todas partes y en ningún sitio. Es capaz de aparecer y desaparecer cuando menos te lo esperas en el lugar más inesperado. Además, tiene mucha suerte y es capaz de ganar un smartphone con sólo asistir a una conferencia. A Sara, otra de mis chicas de teleco, ha sido muy especial y buena compañera de laboratorio. Muy responsable y capaz de compaginar los estudios con una vida social

muy ajetreada. A Víctor G., siempre promocionando La Bañeza (León), es un teleco viajero y solidario al que le encantan los planes al aire libre en la naturaleza. Caminatas incompatibles con Dani, que prefiere el asfalto. A Luis Delgado, teleco por el mundo, muy inquieto y siempre con una sonrisa, tiene buenas anécdotas que contar. A Josevi, que está muy loco, yo no sé cómo lo hace para tener tanto éxito entre las féminas, serán las rastas. Al principio me daba miedo, pero luego se ha convertido en un buen compañero de fiestas. A Víctor Cano, de Badajoz, que quede claro. No soportaba escuchar los resultados al salir de los exámenes. También participaba en las discusiones de política. Alberto, el de Santa Eugenia, amante fiel de U2. Otro caído en batalla, ahora le va bien en eso de ADE. Y por último a Jaime y Bea Agudo, que también nos han acompañado en alguna que otra aventurilla, buena gente.

Por supuesto, al CeDInt por haberme ofrecido la oportunidad hacer el proyecto y dentro del cual he aprendido muchísimas cosas y he conocido a gente muy diferente. En especial al grupo de Realidad Virtual, con Iris a la cabeza, preocupándose siempre por que todo quede bien bien documentado. A mis compis del labo: Fran (FLO), triatleta semiprofesional, que es para mí un referente capaz de resolver todos los problemas que se le presentan; David, que siempre está concentrado en su trabajo y creo que se está volviendo un poco loco porque discute con su pantalla y ha inventado su propio vocabulario; César, que con un comentario breve es capaz de destruirte, pero tiene su corazoncito y cuida de los cactus; Luca, también una referencia y que ha tenido que aguantar que le consulte cosas constantemente y haciéndome sugerencias a las que podría hacer caso o no, pero a cambio yo le enseñaba palabras en español, y Ricardo, que es el chico para todo, si tienes un problema siempre tiene la respuesta o conoce una solución, y no suele equivocarse. Que te vaya todo muy bien con tu nuevo compromiso. A Belén, que se nos va a Japón y se va a perder mi defensa, la única mujer dentro del laboratorio, nos pone rápido a todos en vereda con su buen orden ¿qué vamos a hacer sin ella?. Y a los que ya no están, como Fran (C.) que aunque le llamaban niño y se comportaba como tal, yo le considero como el padre del laboratorio.

Y hablando de los que ya no están, he querido dejar para el final a mi padre Ángel, al cual le debo mi existencia y quien ha dedicado absolutamente toda su vida para que, entre otras muchas cosas, yo pueda estar ahora mismo escribiendo estas palabras. Porque tu recuerdo no sólo permanecerá sobre esta página sino que también estará siempre dentro de mí, en los genes, que esos sí que son eternos propagándose a través de las generaciones. Porque sé que estabas orgulloso de mí, y para que lo sigas estando, a tí va dedicado todo este trabajo.

Índice general

Índice de Figuras

Índice de Tablas

Siglas y Acrónimos

2D	**2 D**imensiones
3D	**3 D**imensiones
3DTV	**3D T**ele**v**ision
AD	**A**bsolute **D**ifferences
AML	**A**ttainable **M**aximum **L**ikelihood
BM	**B**lock **M**atching
BM_CPU	**B**lock **M**atching **C**entral **P**rocessing **U**nit
BM_GPU	**B**lock **M**atching **G**raphics **P**rocessing **U**nit
BP	**B**elief **P**ropagation
BP_GPU	**B**elief **P**ropagation **G**raphics **P**rocessing **U**nit
BPVISION	Efficient **B**elief **P**ropagation for Early **VISION**
BSD	**B**erkeley **S**oftware **D**istribution
BT	**B**irchfield-**T**omasi
CAVE	**C**ave **A**utomatic **V**irtual **E**nvironment
CbC	**C**olumn-**b**y-**C**olumn
CeDInt	**Ce**ntro de **D**omótica **I**ntegral
CSBP	**C**onstant **S**pace **B**elief **P**ropagation
.csv	Extensión del formato **c**omma-**s**eparated **v**alues
CUDA	**C**ompute **U**nified **D**evice **A**rchitecture
CUR	**C**urvature **M**etric
CPU	**C**entral **P**rocessing **U**nit
DIO	**D**istancia **I**nter**O**cular
DLP	**D**igital **L**ight **P**rocessing
DP	**D**ynamic **P**rogramming
DSI	**D**isparity **S**pace **I**mage

DSM	**D**istinctiveness **S**imilarity **M**easure
DTS	**D**istinctiveness
DVB	**D**igital **V**ideo **B**roadcasting
ELAS	**E**fficient **LA**rge **S**cale library
ENT	**E**ntropy metric
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
FC	**F**rame **C**ompatible
FHD3D	**F**ull **H**igh **D**efinition **3D**
fps	**f**rames **p**or **s**egundo
FTV	**F**ree-viewpoint **T**ele**v**ision
GC	**G**raph **C**ut
GPL	**G**eneral **P**ublic **L**icense
GNU	**G**NU is **N**ot **U**nix!
GPU	**G**raphics **P**rocessing **U**nit
GT	**G**round **T**ruth
GUI	**G**raphic **U**ser **I**nterface
HD	**H**igh **D**efinition
HDMI	**H**igh **D**efinition **M**ultimedia **I**nterface
HDTV	**H**igh **D**efinition **T**ele**v**ision
HMD	**H**ead **M**ounted **D**isplay
ICM	**I**terated **C**onditional **M**odels
IRD	**I**ntegrated **R**eceiver **D**ecoder
LAN	**L**ocal **A**rea **N**etwork
LbL	**L**ine-**b**y-**L**ine
LCD	**L**iquid **C**rystal **D**isplay
LCS	**L**iquid **C**rystal **S**hutter
LGPL	**L**esser **G**eneral **P**ublic **L**icense
LRC	**L**eft-**R**ight **C**onsistency
LRD	**L**eft-**R**ight **D**ifference
MAP	**M**inimum **A**-**P**osteriori
MLM	**M**aximum **L**ikelihood **M**etric
MMN	**M**aximum **M**argin
MRF	**M**arkov **R**andom **F**ields

MSE	**M**ean **S**quared **E**rror
MSM	**M**atching **S**core **M**etric
MSR-SCLA	**M**icrosoft **R**esearch **S**ource **C**ode **L**icense **A**greement
MVC	**M**ultiview **V**ideo **C**oding
NCC	**N**ormalized **C**ross-**C**orrelation
NEM	**N**egative **E**ntropy **M**etric
NLM	**Nonl**inear **M**argin
NOI	**N**umber of **I**nflection **P**oints
MVD	**M**ultiview **V**ideo plus **D**epth
NP-hard	**N**on-deterministic **P**olynomial-time **hard**
OpenCV	**Open C**omputer **V**ision library
PDP	**P**lasma **D**isplay **P**anel
PFC	**P**royecto **F**in de **C**arrera
PKR	**P**eak **R**atio
PKRN	**P**eak **R**atio **N**aive
PP	**P**unto **P**rincipal
PRB	**P**robabilistic metric
PSNR	**P**eak **S**ignal-to-**N**oise **R**atio
RAM	**R**amdom **A**ccess **M**emory
RGB	Modelo de color **R**ed **G**reen **B**lue
RMS	**R**oot **M**ean **S**quared error
SAD	**S**um of **A**bsolute **D**ifferences
SbS	**S**ide-**b**y-**S**ide
SC	**S**ervice **C**ompatible
SD	**S**tandard **D**efinition
SD	**S**quared **D**ifferences
SGBM	**S**emi**G**lobal **B**lock **M**atching
SGM	**S**emi**G**lobal **M**atching
SGRAD	**S**uma de **GRAD**ientes
SHD	**S**um of **H**amming **D**istances
SIFT	**S**cale **I**nvariant **F**eature **T**ransform
SA	**S**imulated **A**nnealing
SAMM	**S**elf-**A**ware **M**atching **M**easure

SM	**S**tereo **M**atching
SO	**S**canline **O**ptimization
SSD	**S**um of **S**quared **D**ifferences
STB	**S**et-**t**op **B**ox
StereoVAR	**Stereo VAR**iational
SURF	**S**peeded-**U**p **R**obust **F**eature
SVC	**S**calable **V**ideo **C**oding
TaB	**T**op-**a**nd-**B**ottom
ToF	**T**ime-**o**f-**F**light
TRW-S	**T**ree-**R**e**w**eighted **S**equential
UPV	**U**niversidad **P**olitécnica de **V**alencia
V+D	**V**ideo plus **D**epth
WMN	**W**inner **M**argin
WMNN	**W**inner **M**argin **N**aive
WTA	**W**inner-**T**ake-**A**ll
ZNCC	**Z**ero-mean **N**ormalized **C**ross-**C**orrelation
ZSAD	**Z**ero-mean **S**um of **A**bsolute **D**ifferences
ZSSD	**Z**ero-mean **S**um of **S**quared **D**ifferences

Nomenclatura

Símbolo	Significado	Unidad
$1080i$	Formato HD de resolución 1920x1080 escaneo entrelazado	px
$1080p$	Formato HD de resolución 1920x1080 escaneo progresivo	px
$720p$	Formato HD de resolución 1280x720 escaneo progresivo	px
α_x (α_y)	Distancia focal horizontal (vertical) medida en píxeles	px
γ	Coeficiente de distorsión radial	
θ	Ángulo entre los ejes de la imagen	rad
b	Distancia entre centros ópticos (línea de base)	m
C	Centro de proyección (origen de coordenadas de la cámara)	uds
c	Centro de imagen	px
$C(x, y, d)$	Función de coste en la posición (x, y) para la disparidad d	
$d(x_k)$	Disparidad del píxel de coordenada horizontal x de la imagen k	px
d_{max}	Disparidad máxima	px
d_{min}	Disparidad mínima	px
$D_p(f_p)$	Data cost de asignar la etiqueta f al píxel p	[**E**]
$[E]$	Matriz esencial	
e_l (e_r)	Epipolo en la imagen izquierda (derecha)	px
$[F]$	Matriz fundamental	
f	Distancia focal	m
f_l/f_r	Distancia focal de la cámara izquierda (derecha)	m
f_p/f_q	Etiqueta asignada al píxel p (q)	
I_{dif}	Imagen de error normalizada	
$\bar{I}_k$	Valor medio de las intensidades de la imagen k	
$\hat{I}_k(i)$	Valor interpolado de un píxel de la imagen k en la posición horizontal i	
$\tilde{I}_k$	Imagen k reconstruida (o predicción de la imagen k)	

$[K]$	Matriz de parámetros intrínsecos de la cámara	
$[K]_l$ ($[K]_r$)	Matriz de parámetros intrínsecos de la cámara izda. (dcha.)	
L (R)	Plano de proyección izquierdo (derecho)	
$\mathbf{L}$	Conjunto de etiquetas o *labels*	
l_l (l_r)	Linea epipolar en la imagen izquierda (derecha)	px
$[M]$	Matriz de proyección perspectiva de la cámara	
m_x/s_x/k	Tamaño efectivo de los píxeles en dirección horizontal	px/m
m_y/s_y/l	Tamaño efectivo de los píxeles en dirección vertical	px/m
m_{pq}^t	Mensaje del nodo p al nodo vecino q en el instante t	[**E**]
$\mathbf{N}$	Vecindad o entorno de nodos vecinos en un grafo	
$N(p)$	Vecindad del nodo p	
$N(p) \setminus q$	Conjunto de nodos vecinos de p que no lo son de q	
n_{disp}	Número de niveles de disparidad	px
n_{iter}	Número de iteraciones	iter.
n_{levels}	Número de niveles	niv.
O	Origen de coordenadas globales en el espacio	uds
O_L (O_R)	Centro óptico de la cámara izquierda (derecha)	uds
P	Punto en el espacio o escena 3D	uds
$\mathbf{P}$	Conjunto de píxeles	
P_e	Tasa de píxeles erróneos	%
P_{err}	Tasa de píxeles incorrectos	%
P_i	Tasa de píxeles inválidos	%
P_l (P_r)	P visto por la cámara izda.(dcha.) en coordenadas espaciales	uds
p	Píxel o proyección de un punto P en el plano de imagen	px
p_l (p_r)	Píxel de la imagen izquierda (derecha)	px
PP_L (PP_R)	Punto principal de la imagen izquierda (derecha)	px
q	Píxel	px
$[R]$	Matriz de rotación de la cámara	
$\overrightarrow{t}$	Vector de traslación de la cámara	uds
T	Posición del afijo de $\overrightarrow{t}$ en el espacio	uds
$\mathbf{T}$	Número de iteración	
t	Tiempo	s
$V(f_p, fq)$	Discontinuity Cost entre píxeles vecinos p y q	

win_{size}	Ancho de ventana	px
$\|x\|$	Parte entera de x	
$(X, Y, Z)^T$	Coordenadas de un punto P en el espacio	uds
$(X, Y, Z, 1)^T$	Coordenadas homogéneas de un punto P en el espacio	uds
$(x, y)^T$	Coordenadas de un píxel p	px
$(x, y, 1)^T$	Coordenadas homogéneas de un píxel p	px
(x_0, y_0)	Coordenadas del punto principal	px
Z	Profundidad o distancia de un objeto a la cámara	m

Capítulo 1

Introducción

1.1 Presentación y objetivos del proyecto

Hace apenas cinco años apareció la **televisión de alta definición** y muchas empresas trabajaron en su definición empujadas por el interés en innovar y atraer a un mayor número de usuarios, buscando **nuevas formas de llegar al espectador**. Una manera de lograr este acercamiento ha sido intentar inducir en el espectador la sensación de pertenecer al escenario que se visualiza aumentando la inmersividad. Por este motivo, en los últimos años se ha incrementado el esfuerzo en investigación y desarrollo de la **tecnología 3D**, capaz de producir dicha sensación de encontrarse dentro de la escena. Sin embargo, el espectador mantiene un rol pasivo observándola y es necesario eliminar las barreras que lo separan de la escena visualizada.

En este contexto, se define la **sensación de inmersión** como un mecanismo de percepción relacionado con el estado mental del observador por el cual siente la relación entre la autoconsciencia y el entorno que le rodea [1]. Por lo tanto, la industria tiende a la **convergencia** de los medios tradicionales con la realidad virtual y aumentada, apareciendo los sistemas inmersivos. Un paso más allá es el desarrollo de programas basados en la **interactividad**. Permitiendo a los espectadores interactuar con el contenido en tiempo real, la audiencia se involucra activamente para vivir el evento según sus preferencias, dejando de ser un elemento pasivo. Se considera interacción ocultar y superponer detalles o información adicional, modificar objetos 3D insertados en la escena (mover, escalar, ocultar), consultar estadísticas y otros datos textuales o crear vistas personalizadas del evento mediante selección del punto de vista.

La interactividad se aplica con éxito en entornos sintéticos, donde la información 3D está disponible como parte del modelo de la escena. Sin embargo, los flujos 2D tradicionales presentan limitaciones como el punto de vista fijo y la carencia de definición 3D completa de la escena, que se resuelven parcialmente bajo condiciones bien definidas como las de los **vídeos estereoscópicos**. Aunque la profundidad estéreo no proporciona una idea completa de la geometría de todos los objetos 3D (se define como 2.5D), puede ser utilizada para permitir la interacción con modelos 3D integrados en el flujo original.

El trabajo realizado para este PFC forma parte del proyecto *ImmersiveTV* [2], concebido para **mejorar la experiencia** de usuario de los espectadores de televisión 3D mediante el aumento de la **sensación de presencia** y permitiendo un mayor grado de **interactividad** con el contenido. *ImmersiveTV* tiene como objetivo aprovechar el potencial inmersivo e interactivo del sistema de difusión de televisión HD durante la emisión de un evento 3D en directo, para lo cual plantea dos escenarios de prueba.

El primer escenario se basa en un **entorno inmersivo de tipo CAVE™** instalado en el CeDInt [3, 4] en el cual se visualiza un panorama estéreo de la ciudad de Barcelona. En este escenario, la interactividad se logra con ayuda de una interfaz que permite al usuario seleccionar y mover, haciendo uso de un *Flystick*®, objetos sintéticos 3D que se insertan en el vídeo. **El segundo escenario**, el **salón inmersivo** [4, 5], representa un salón doméstico del futuro, donde la inmersividad se consigue a través de audio binaural y de la creación de un ambiente inmersivo con ayuda de dispositivos actuadores que generan efectos sensoriales sincronizados con el contenido audiovisual. Se visualiza un partido de fútbol en el cual se insertan objetos 3D interactivos que se hacen visibles al producirse un evento concreto durante la emisión. Además, se pueden consultar estadísticas y contenido adicional en tiempo real con ayuda de una segunda pantalla, que puede tratarse de una tablet o dispositivo móvil, aumentando la interactividad.

Según la descripción anterior, el elemento común a ambos escenarios es la aparición de **objetos sintéticos interactivos que se introducen en el vídeo**. Para lograrlo, se ha desarrollado un módulo de inserción de dichos objetos que necesita conocer la información de profundidad de la escena para calcular las oclusiones con los elementos que la componen. Concretamente, **el trabajo descrito en esta memoria se ocupa de resolver el problema de extracción de información de profundidad** a partir

del vídeo difundido a través de la infraestructura de televisión y **que se muestra en el escenario salón**.

Por lo tanto, el **objetivo principal** de este proyecto es **encontrar una solución al problema de estimación** de la información **de profundidad** del contenido estereoscópico que se plantea en el proyecto *ImmersiveTV*. Esta información es necesaria para que el módulo de inserción de objetos sintéticos interactivos sea capaz de calcular correctamente las oclusiones de los objetos 3D con los elementos de la escena. Además, el sistema de estimación deberá ser capaz de procesar vídeo estereoscópico en tiempo real al tratarse de un módulo situado en un sistema de recepción de televisión tridimensional. Para realizarlo, se plantea el siguiente conjunto de objetivos parciales:

- **Análisis de los requisitos** del módulo de inserción de objetos sintéticos interactivos y del tipo de contenido utilizado en el demostrador del proyecto *ImmersiveTV* para localizar los problemas a enfrentar.
- **Revisión y análisis** del estado del arte **y selección de las técnicas de correspondencia estéreo.** Atendiendo a los requisitos de la aplicación, seleccionar y evaluar de forma práctica un subconjunto de dichas técnicas y algoritmos con el fin de determinar cuáles pueden ser adecuadas para resolver el problema planteado.
- **Desarrollo de un sistema de estimación de mapas de profundidad densos a partir de secuencias reales de vídeo 3D.** Implementación de una herramienta software capaz de, dadas dos secuencias de vídeo correspondientes a las vistas del ojo izquierdo y derecho, estimar el mapa de disparidad de la escena con la calidad suficiente para llevar a cabo la inserción de objetos sintéticos interactivos en escenas de vídeo 3D.
- **Análisis comparativo del comportamiento y de los resultados** de los algoritmos de correspondencia aplicados al problema real. Revisión del trabajo existente en el campo de la evaluación de algoritmos de correspondencia con el fin de encontrar un conjunto de criterios que permitan una comparación fiable. Diseño e implementación de un módulo software capaz de extraer estadísticas de los resultados para facilitar la evaluación atendiendo a los criterios establecidos. Selección de la técnica o algoritmo que más se ajusta a las necesidades del módulo de inserción de objetos sintéticos interactivos.

1.2 Estructura del documento

La estructura de capítulos que componen esta memoria es la siguiente:

Capítulo 1. Introducción

En este capítulo introductorio se define el contexto de nuestro trabajo mediante la descripción detallada del proyecto *ImmersiveTV* y sus escenarios de prueba con el fin de establecer los objetivos y definir el problema concreto que debemos resolver. Debido a las características propias del contenido y del proceso de estimación de profundidad (*stereo matching*) nos encontraremos con problemas asociados que debemos identificar.

Capítulo 2. El ciclo de vida de la 3DTV: fundamentos teóricos y tecnologías

Se introducen los fundamentos de la estereoscopía y la visión estereoscópica, definiendo términos como la estereopsis, la visión binocular o el paralaje, con el objetivo de entender cómo afectan a la percepción de la profundidad. Por otra parte, se presenta el ciclo de vida de los contenidos tridimensionales, haciendo un recorrido por las fases que resultan de mayor interés aplicadas a la cadena de valor de la televisión 3D: generación de contenidos, formatos de transmisión del contenido y tecnologías de visualización.

Capítulo 3. Calibración y rectificación

El primer requisito de los algoritmos de correspondencia es que las imágenes de entrada estén alineadas horizontalmente, cumpliendo la restricción epipolar. En primer lugar, hay que comprobar si el contenido con el que se va a trabajar está alineado. En caso negativo, es necesario aplicar rectificación. En este capítulo se explican los conceptos en torno a este proceso, desde el modelo de geometría proyectiva y formación de imagen, pasando por los parámetros de la cámara, la geometría epipolar y las matrices que definen la transformación geométrica proyectiva que compone la rectificación. Adicionalmente, se realiza una revisión de algunas implementaciones, analizando cuáles pueden ser las más adecuadas en el contexto de nuestra aplicación.

Capítulo 4. Correspondencia y reconstrucción

En este capítulo se define el concepto de disparidad, que determina la profundidad percibida de los objetos de la escena, y se plantean los fundamentos de la correspondencia estéreo, también conocida como *Stereo Matching*. Se presenta un conjunto representativo de algoritmos de correspondencia densa con el fin de comparar las características

de los diversos enfoques posibles para enfrentar el problema propuesto. Con el objetivo de analizar cuál es el más adecuado para nuestra herramienta, se realiza un estudio de las implementaciones de algunos algoritmos, además de un análisis de los entornos de evaluación más habituales para compararlos de forma fiable y objetiva. Finalmente, se describe, de manera teórica, la solución adoptada.

Capítulo 5. Implementación y funcionalidades del sistema

En este capítulo se describen las modificaciones realizadas sobre las implementaciones de los algoritmos utilizados. También se explica en profundidad la interfaz gráfica de usuario (GUI), para facilitar al lector la utilización de la herramienta, y se realiza un recorrido detallado a través de los módulos que componen la herramienta desarrollada para estimar mapas de disparidad densos, presentando sus funcionalidades.

Capítulo 6. Pruebas y evaluación de resultados

En este capítulo se recopilan los resultados de los experimentos realizados sobre un conjunto de capturas de prueba con el fin de comparar la calidad, tanto subjetiva como objetiva, y la eficiencia de los diferentes algoritmos de correspondencia, justificando la selección del algoritmo más adecuado.

Capítulo 7. Presupuesto

Se presenta una tabla que resume los costes asociados al desarrollo de este Proyecto de Fin de Carrera.

Capítulo 8. Conclusiones y trabajo futuro

En el capítulo final se resumen las conclusiones alcanzadas a lo largo del proyecto y se plantean las posibles líneas de trabajo futuro para la mejora y continuación del trabajo propuesto.

Anexo I. Código de las modificaciones sobre los algoritmos

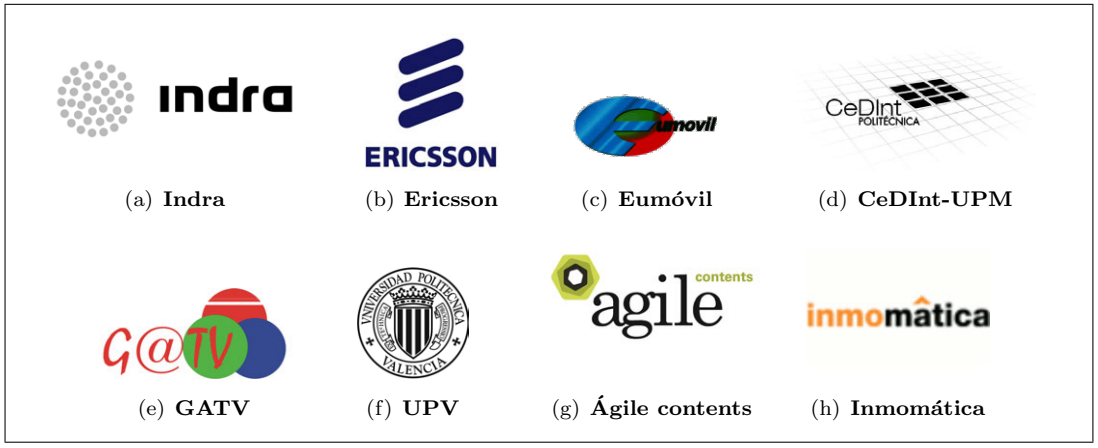
En este anexo se recoge el código de ejemplo de las principales modificaciones o adaptaciones realizadas sobre las implementaciones de algoritmos de correspondencia estéreo.

Anexo II. Instalación del DVD

En este anexo se explica el contenido del DVD adjunto a este documento y el procedimiento de instalación.

1.3 Contexto del PFC: El proyecto *ImmersiveTV*

ImmersiveTV es un proyecto tractor, cofinanciado por el Ministerio de Industria, Turismo y Comercio en el marco de Avanza Competitividad, con duración 2010-2013. Los socios participantes en el proyecto aparecen en la figura 1.1 [2]. Su objetivo es **investigar sobre las tendencias de producción de contenidos digitales inmersivos y toda la cadena de valor** involucrada (producción, transmisión y recepción), desarrollando la interactividad e inmersividad más allá de la actual 3D, que es meramente bidimensional. De este modo se pretende que el usuario final experimente la sensación de pertenecer al propio escenario que está visualizando pudiendo interactuar con él en cualquier dirección espacial.



(a) **Indra** (b) **Ericsson** (c) **Eumóvil** (d) **CeDInt-UPM**

(e) **GATV** (f) **UPV** (g) **Ágile contents** (h) **Inmomática**

FIGURA 1.1 – **Socios del proyecto *ImmersiveTV* [2]**

A continuación describimos los dos escenarios que se contemplan dentro del proyecto y pasaremos a explicar con cierto nivel de detalle las características técnicas del contenido utilizado para realizar pruebas y experimentos.

1.3.1 Escenarios de prueba

Los **ambientes virtuales inmersivos** son espacios tridimensionales reales o imaginarios generados por ordenador con los que el usuario puede interactuar y que son capaces de producir la sensación de encontrarse situado en el interior. La **sensación de presencia** se genera al integrar varios elementos, como la generación de un gran número de imágenes de alta calidad por segundo que cubran un amplio grado del campo de visión del usuario, que son resultado de la interacción con la escena por medio

del movimiento, modificándose el entorno y, además, la utilización de sonido espacial relacionado con todo el ambiente.

En este ambiente se busca la **interacción** de la forma más **natural** posible. Para ello, se utilizan dispositivos como guantes, sistemas de rastreo o seguimiento del movimiento, o interfaces de entrada más específicos vinculados al ambiente en el que se trabaja. Por ejemplo, volantes y palancas de cambio de velocidades para un simulador de conducción, o simuladores de instrumentos y herramientas de cirujía.

Las experiencias inmersivas exploradas dentro de *ImmersiveTV* han sido seleccionadas y desarrolladas con el objetivo de alcanzar los criterios de **inmersividad** e **interactividad**. Dicho objetivo se alcanza a través de dos escenarios complementarios: **CAVE** [6–8], y **Salón Interactivo**, que se describen brevemente a continuación.

1.3.1.1. Entorno CAVE™

El primer escenario de prueba y validación del *ImmersiveTV* hace uso de la CAVE instalada en el CeDInt [3, 4]. Puesto que las instalaciones de la CAVE no contemplan la posibilidad de recibir **contenido** por ningún medio, se ha decidido que este sea **no directo**.

La CAVE permite desarrollar experiencias inmersivas en las que es posible cambiar el punto de vista de los contenidos que se muestran por pantalla según los movimientos del usuario y la orientación de su mirada. Esto es posible gracias al uso de la información de posición de la cabeza del usuario capturada por el **sistema de tracking óptico** de la CAVE™. Si se colocan los objetos de forma que aparentemente salen de la pantalla (dentro del volumen real que abarca la CAVE™), este sistema permite al usuario verlos desde distintas perspectivas. Haciendo uso de esta característica y de las variaciones de profundidad proporcionadas por los contenidos 3D, se busca enriquecer el contenido real con información adicional como texto, imágenes, vídeo o audio.

En la figura 1.2 se presenta la arquitectura del demostrador. En el diagrama mostramos la **estructura de la CAVE**, que consiste en un paralelepípedo formado por 5 pantallas (lateral izquierdo, frontal, lateral derecho, inferior y superior) de resolución 1400x1050 píxeles y dimensiones 3,2 x 2,4 metros, con sus respectivos proyectores estereoscópicos activos. A causa de esta configuración de pantallas consecutivas, es preferible

proyectar un **contenido panorámico**. Por otro lado se dispone de un conjunto de estaciones de trabajo que controlan cada uno de los proyectores y en las cuales se almacenan los contenidos pregrabados de su vista correspondiente. Un equipo central se encarga de sincronizar y comunicar todas estaciones de trabajo y otro equipo gestiona el seguimiento del movimiento del usuario. La visualización puede ser con obturación activa normal o de tecnología Infitec.

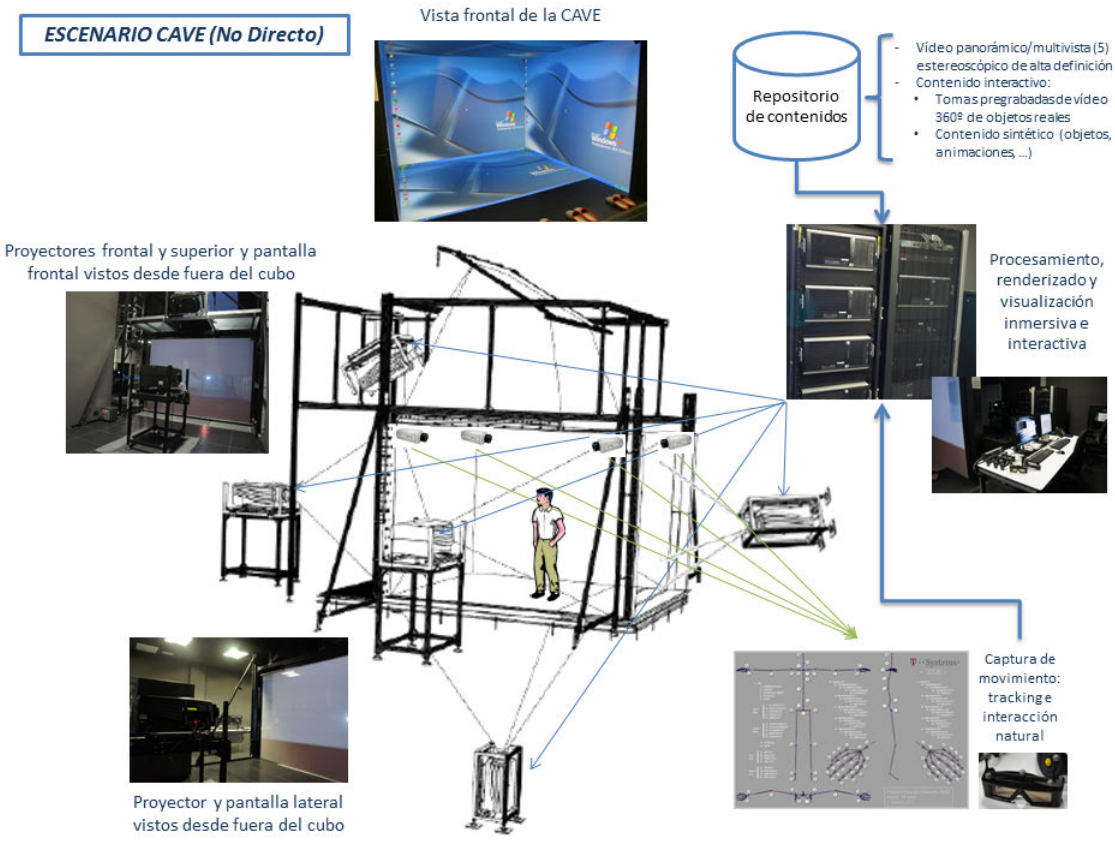


FIGURA 1.2 – **Arquitectura del escenario CAVE** (CeDInt)

1.3.1.2. Salón inmersivo

El proyecto *ImmersiveTV* pretende realizar una emisión real de contenido en directo, abarcando todos los eslabones de la cadena de valor de una transmisión de televisión 3D. Para que el resultado sea fiable es preciso validar cada uno de ellos. El segundo escenario de prueba y validación del *ImmersiveTV* trata de representar un **salón doméstico del futuro**, recibiendo contenidos vía difusión (*broadcast*) y a través de banda ancha (*broadband*).

Uno de los contenidos que más seguimiento tiene en televisión es la **retransmisión de eventos deportivos**. Por ello, este escenario [4, 5] reflejará la retransmisión *broadcast* de un partido de fútbol en directo en una pantalla de 46 pulgadas. El contenido que se debe generar para este escenario, por tanto, es un partido completo de fútbol profesional, grabado mediante tecnología estereoscópica.

El hecho de presentar el contenido en 3D conlleva un **aumento de la inmersividad** en la experiencia de visualización. Para acentuar aún más la inmersividad se plantean tres aproximaciones diferentes que pueden aplicarse por separado:

- **Interacción** con elementos 3D generados por ordenador que se integran con el contenido de vídeo: son contenidos de **realidad aumentada**, objetos 3D que aparecen superpuestos sobre la imagen original cuando ocurre un evento de interés en un momento dado a lo largo del partido. El usuario tiene la opción de visualizar o no la capa de información correspondiente a dicho contenido enriquecido e interactuar con él.
- Creación de un **ambiente inmersivo** mediante la modificación de las condiciones de la sala de forma síncrona con el contenido de vídeo. Para lograrlo se utilizan **elementos domóticos** como sensores, sistemas de audio 3D, iluminación de ambiente e iluminación de acento, difusores de aromas y difusores de humo.
- Visualización en tiempo real de **servicios adicionales en una segunda pantalla**: la **interactividad** en este escenario viene dada por el contenido adicional que se recibe a través de banda ancha, visualizado en un dispositivo con navegador de Internet, como puede ser una tablet o un dispositivo móvil, que permite la interacción individual sin afectar al resto de televidentes de la sala. La información adicional interactiva que recibe el usuario en tiempo real son eventos relacionados con el desarrollo del partido, como pueden ser estadísticas, la alineación de cada equipo participante, repeticiones de jugadas importantes o selección de vistas adicionales de la misma jugada.

Este planteamiento sigue el concepto de televisión interactiva conocido como ***HybridTV***, caracterizado por los receptores de reproducir la señal proviente de las redes de difusión e internet (banda ancha). En recepción, se demultiplexa y enruta adecuadamente el contenido según el tipo de terminal de cada usuario utilizando la LAN doméstica.

En la figura 1.3 se presenta la **arquitectura del demostrador**, donde se puede observar que el dispositivo central es el STB, que es capaz de recibir tanto la señal multimedia como la señalización para los dispositivos domóticos y por otra parte acceder a través de Internet al resto de contenido adicional. Los dispositivos de presentación son una pantalla de gran tamaño para el contenido de televisión 3D y una tablet para el contenido interactivo adicional. Además, se dispone de un mando de videoconsola para interactuar con los objetos integrados en el vídeo.

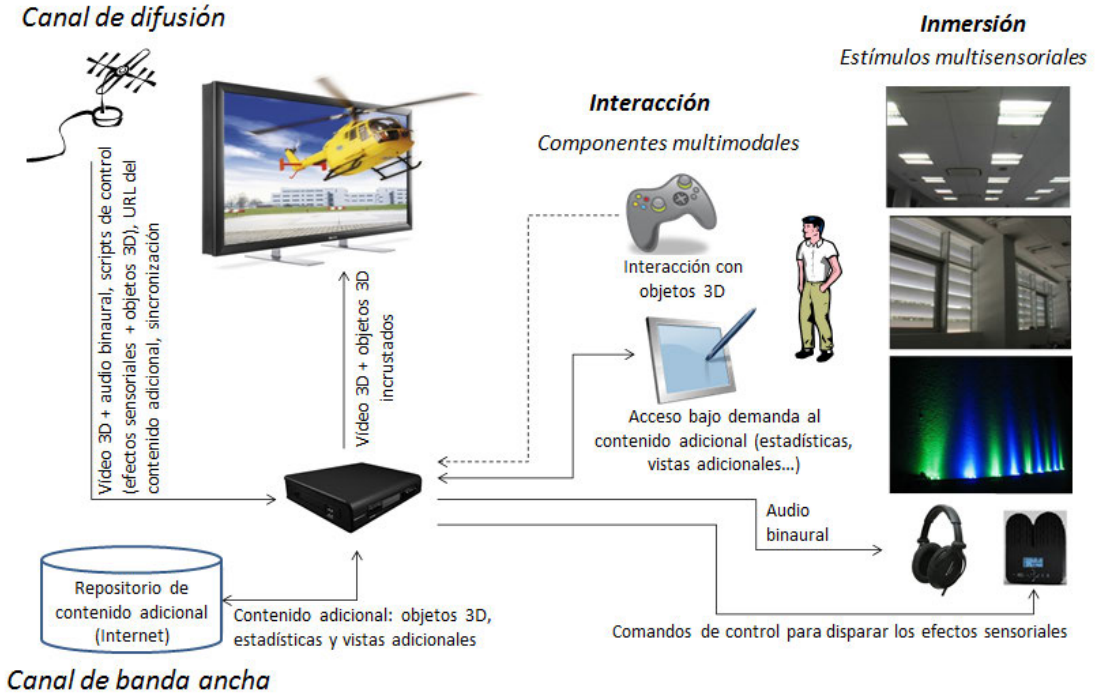


FIGURA 1.3 – **Arquitectura del escenario salón** [5]

1.3.2 Descripción del contenido

1.3.2.1. Entorno CAVE™

En este escenario se utiliza una **grabación estereoscópica** de resolución *full HD* 1920x1080 píxeles para cada ojo, realizada desde la azotea del edificio Imagina de Mediapro en la Avenida Diagonal de **Barcelona** durante 24 horas. El contenido consiste en **cuatro vistas** estereoscópicas **parcialmente solapadas** que, en conjunto, abarcan un ángulo de visión de 270° (figura 1.4) con el objetivo de generar una **vista panorámica** estéreo (figura 1.5, arriba).

FIGURA 1.4 – **Captura de cuatro vistas de Barcelona** (fuente: Eumóvil)

La vista panorámica estéreo se obtiene mediante una técnica conocida como ***stitching* (o cosido de imágenes)**, que es una variante de la formación de imágenes en mosaico. Se visualiza en las tres caras verticales de la CAVE (laterales y frontal) junto con una interfaz gráfica interactiva para poder mostrar todas las posibilidades del demostrador.

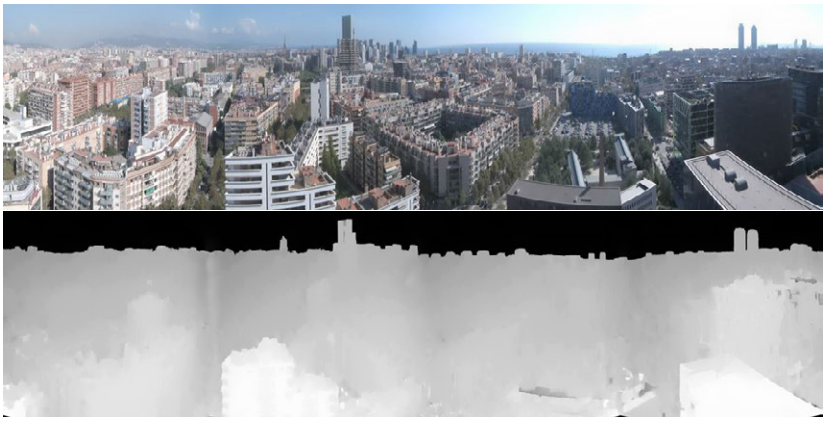


FIGURA 1.5 – **Vista panorámica** (arriba) y **mapa de profundidad** (abajo)

Cada una de las vistas se tomó con un conjunto de dos cámaras Panasonic AG-HPX371E colocadas en un rig estéreo con estructura *Side-by-Side*. Se trata de una configuración de cámaras en paralelo con separación entre los centros ópticos (interaxial) de 680 milímetros. El eje de rotación se sitúa en el centro de las dos cámaras físicas, pero los sensores se sitúan 152 milímetros por delante del mismo. Para poder capturar la calle desde lo alto del edificio, las cámaras tienen un ángulo de inclinación con respecto al suelo de $-10,55°$. Para no producir deformaciones, se establece una configuración sin zoom. Debido a la gran distancia que separa la cámara de la escena, la lente de cada cámara tiene configuración de foco en el infinito, que es el punto a partir del cual el enfoque permanece constante, es decir, todos los objetos mas allá de la distancia mínima de enfoque deberían aparecer perfectamente claros y sin distorsión.

Sobre el contenido descrito, el demostrador incluye una serie de **contenidos adicionales** (figura 1.6) destinados a complementar la experiencia de usuario. Como funcionalidad, se introduce la visualización selectiva (a demanda) de dichos contenidos, que son realizados de forma previa a la transmisión y almacenadas en un repositorio local. Pueden ser de tres tipos:

- **Modelos 3D:** un menú de interacción permite seleccionar entre varios modelos sintéticos asociados con la ciudad de Barcelona durante la reproducción del vídeo. Se trata de edificios representativos de la ciudad que el usuario puede observar desde cualquier posición y ángulo mediante las interacciones con el *Flystick*® de la CAVE™. Además, se pueden integrar dentro del panorama de vídeo mediante la acción correspondiente en el menú de interacción.
- **Audio:** además de la pista de audio general que se reproduce junto al panorama de vídeo, cada uno de los modelos 3D seleccionables tienen a su vez una pista de audio descriptiva. Estas se reproducen a partir del momento en que se selecciona alguno de los edificios 3D disponibles, pudiendo en cualquier momento pausar o reanudad la reproducción mediante las interacciones con el menú de selección.
- **Fotografías 3D:** al seleccionar un edificio 3D, sobre el panorama aparece un visor que muestra fotografías relacionadas con el modelo. La secuencia de fotografías y el tiempo que permanecen en escena vienen determinados automáticamente por el audio asociado que se encuentre reproduciendo en ese momento. Además, se habilita la interacción del usuario con este visor permitiendo avanzar o retroceder la secuencia, o bien mostrar y ocultar el visor.

FIGURA 1.6 – **Ejemplo del demostrador CAVE.** Vista del demostrador en funcionamiento con el menú interactivo donde se muestra un modelo del estadio Camp Nou insertado en el vídeo (parte izquierda) y una presentación de fotografías relacionadas con el modelo 3D (parte derecha). Para interactuar se utiliza el *Flystick*®.

1.3.2.2. Salón inmersivo

El contenido multimedia que se mostrará en este demostrador corresponde a una escaleta de un **partido completo de fútbol profesional**, grabado mediante tecnología estereoscópica. El encuentro se realizó en el estadio Camp Nou de Barcelona entre el Fútbol Club Barcelona y el Real Madrid durante la jornada 35 de la liga BBVA, el 21 de abril de 2012. La captura ha sido realizada utilizando una configuración de cámaras habitual en cualquier retransmisión futbolística, siguiendo un esquema como el de la figura 1.7.

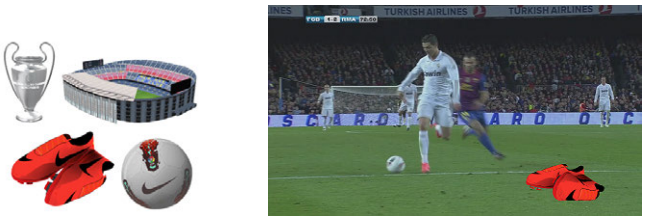
FIGURA 1.7 – **Esquema de cámaras del partido:** (1) principal SbS, (2-3) cortos SbS, (4-5) detrás de las porterías SbS, (6) lateral TaB, (7) vista de las gradas, (8) *steady* hacia banquillos, (9-10) fueras de juego.

La cámara principal, etiquetada como 1 es apoyada por las cámaras 2 y 3, utilizadas para tomar planos cortos de los jugadores en el terreno de juego. Las cámaras 4 y 5 están situadas detrás de cada portería para tomar planos cortos durante las jugadas que se producen cerca de las porterías y la cámara 6 recorre la banda lateral. También se dispone de dos cámaras (9 y 10) para detectar y comprobar las situaciones de fuera de juego. La cámara 8 está colocada sobre un soporte de tipo *steadycam* y apunta hacia los banquillos, mientras que la cámara 7 graba hacia las gradas durante el tiempo en el que se llena y se vacía el estadio.

FIGURA 1.8 – **Ejemplo de cuadro de vídeo para el escenario Salón** (Eumóvil)

Las **cámaras HD** utilizadas son del modelo GV LDK 8000, excepto la situada tras la portería derecha, que se trata de una cámara 3D Panasonic AG-3DA1 que captura cada vista en *Full HD* y SbS. Todos los rigs están configurados en *Side-by-Side*, excepto el de la banda lateral, que es *Top-and-Bottom*. La vista de cada ojo se captura en formato **720p** (1280x720 píxeles formato progresivo), por lo que el resultado del montaje final es un vídeo en formato ***HD Ready Side-by-Side*** , es decir un cuadro de dimensiones 2560x720 píxeles que contiene las vistas de resolución completa una al lado de la otra.

Como hemos explicado en el apartado 1.3.1.2, se utiliza un conjunto de **modelos tridimensionales** que **se insertan** en el vídeo como contenido adicional en momentos determinados del partido. En la figura 1.9(a) se muestran todos los objetos que se han modelado y que están directamente relacionados con el contenido del vídeo. En la figura 1.9(b) vemos la captura de un cuadro en el que se ha insertado uno de los objetos.



(a) **Modelos 3D** (b) **Superposición de un modelo**

FIGURA 1.9 – **Superposición de contenidos de realidad aumentada** (CeDInt)

Por otra parte, sabemos que es posible **interactuar** con los modelos 3D dentro del vídeo, como en la figura 1.10, donde se ilustra el desplazamiento de posición de un modelo en el vídeo, pudiéndose observar el **efecto de las oclusiones** entre elementos situados a **diferente profundidad** como ocurre con el techo del estadio que se sitúa por delante de la copa (imagen superior).

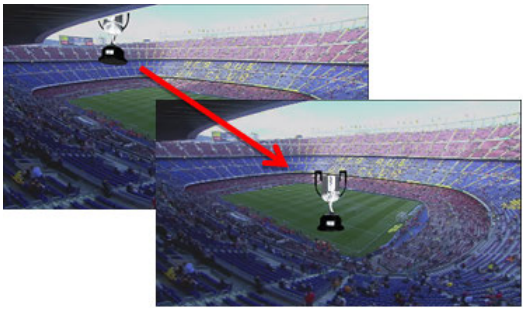


FIGURA 1.10 – **Ejemplo de interacción en el escenario Salón** (fuente: CeDInt)

Otra posible forma de interacción hace uso de las **vistas de las diferentes cámaras** con el objetivo de incrementar el nivel de implicación del usuario, de forma que tenga cierto control al poder seleccionar la que más le interese. Así, los espectadores se convierten en verdaderos realizadores de su propio contenido. Esta posibilidad supone numerosos retos a nivel de producción. Por un lado, se debe planificar cada una de las distintas posiciones de cámara estereoscópica y por otro, almacenar de forma coherente las diferentes fuentes de vídeo. En este sentido, se presenta un reto importante en cuanto al tamaño que ocupa el contenido en términos de compresión sin alterar excesivamente la calidad final y en cuanto a la adecuada sincronización entre las vistas. Por otra parte, se debe prever el método más efectivo para dotar al usuario final de la capacidad de ir alternándolas.

1.4 Especificación del problema a resolver

1.4.1 Descripción general del problema

En este Proyecto Fin de Carrera nos centramos en la **estimación de profundidad del contenido visualizado en el escenario salón**. Se trata de un vídeo estéreo en formato *Side-by-Side* correspondiente a la retransmisión de un evento deportivo en directo, por lo tanto, disponemos de la información de dos vistas, una correspondiente a cada ojo. Cuando se visualiza el contenido, cada vista alcanza el ojo correspondiente y, gracias a la diferencia de posición de un punto de la escena visible en ambas vistas, el cerebro es capaz de interpretar la información de profundidad [9–12]. De manera similar, en visión artificial, las técnicas de correspondencia estéreo o *stereo matching* aprovechan las pequeñas diferencias existentes entre ambas vistas para determinar la distancia que separa la cámara de un punto cualquiera de la escena, esto es, su valor de profundidad. Si las imágenes de entrada están alineadas, la diferencia entre una vista y la otra es un desplazamiento horizontal conocido como disparidad. Este desplazamiento está relacionado con la profundidad percibida de los objetos de la escena [13–15]. Por lo tanto, en el caso de imágenes rectificadas, **el problema reside en calcular la disparidad existente entre ambas imágenes**.

En este trabajo se describe el desarrollo de un sistema de estimación de mapas de profundidad **a partir de secuencias reales de vídeo 3D**. Para abordarlo, en primer lugar es necesario conocer los requisitos del módulo de inserción de objetos sintéticos interactivos para poder determinar las características del mapa de disparidad que se tiene que calcular. En este caso, es deseable un **mapa de disparidad denso**, es decir, aquel en el que todos los píxeles de la imagen tienen un valor de disparidad válido asignado, así no existirán agujeros al situar un objeto 3D sintético dentro de la escena. Por otra parte, **no es necesario obtener un mapa preciso** ya que es suficiente con distinguir los diferentes planos de profundidad de la escena, puesto que el módulo de inserción de objetos trabaja con distancias relativas sin conocer el valor exacto de profundidad de los objetos. Una vez conocidos los requisitos y, asumiendo que se dispone de un contenido rectificado o que se puede rectificar, es necesario hacer una **revisión de las técnicas de correspondencia estéreo** [16] para preseleccionar un conjunto de muestra que pueda ser útil para nuestra herramienta. El resultado de aplicar los

algoritmos de correspondencia **depende de las características del contenido** y, por consiguiente, también la selección de algoritmos. Otro factor limitante en la búsqueda de algoritmos de correspondencia es la capacidad de trabajar en tiempo real, por lo que **resulta de interés utilizar implementaciones basadas en GPU**.

Finalmente, para elegir adecuadamente el algoritmo que cumple mejor los requisitos de la aplicación, **se debe realizar una comparación de los resultados**. Existen muchos trabajos en torno a la evaluación de algoritmos de correspondencia atendiendo a aplicaciones concretas o centrados en alguna de las etapas que componen el proceso de correspondencia [16–22], en ocasiones, dando lugar a entornos de evaluación online [23, 24] o bases de datos de contenidos de prueba [16, 25–27]. En nuestro caso, la principal dificultad es que **no se dispone de *ground truth* de referencia**, por lo que se debe buscar un criterio objetivo de comparación indirecta [28–30]. Como resultado de la revisión de estos trabajos, se propone diseñar un **entorno de evaluación** de algoritmos de correspondencia utilizando un conjunto de **criterios de comparación indirecta** basados en la reconstrucción de la imagen de referencia a partir del mapa de disparidad obtenido.

1.4.2 Introducción al problema de *stereo matching*

Partiendo de dos imágenes tomadas desde diferentes puntos de vista de una misma escena correspondientes a la vista del ojo izquierdo, que será referida como **imagen de referencia** en la mayoría de los casos, y del ojo derecho, que llamaremos imagen objetivo o imagen de comparación, el problema de *stereo matching* o correspondencia estéreo consiste en encontrar e identificar en dichas imágenes aquellos píxeles que representan al mismo punto de la escena tridimensional. De esta manera se puede estimar la profundidad de cada punto de la escena a partir de un **mapa de disparidad** como el de la figura 1.11(c), resultante del proceso de correspondencia.

En las especificaciones [31, 32] se define la **disparidad** como la diferencia entre las posiciones horizontales de un píxel que representa al mismo punto en el espacio en las vistas derecha e izquierda. La disparidad positiva (según el convenio en el que la coordenada horizontal derecha es mayor que la coordenada horizontal izquierda) implica una posición detrás del plano de pantalla, y la disparidad negativa implica una posición

(a) Vista ojo izquierdo (b) Vista ojo derecho (c) Mapa de disparidad

FIGURA 1.11 – Ejemplo de problema de correspondencia estéreo para la imagen de prueba Tsukuba [16, 23]

delante de la pantalla. Esta definición coincide con la definición de paralaje de la sección 2.2, donde se representan los tipos de paralaje en la figura 2.3.

Para realizar una buena correspondencia es imprescindible que las imágenes estén bien rectificadas, como se explicará más adelante, y una escena en la que se puedan reconocer bien los elementos gracias a una buena iluminación con ausencia de reflejos o sombras excesivas. También nos encontramos con el problema de las oclusiones, situación en la cual un conjunto de píxeles de una imagen no aparece en la otra debido a la posición relativa de los objetos en la escena, lo cual dará lugar a errores en el proceso de cálculo de la correspondencia. Todos estos problemas y restricciones relacionados tanto con el tipo de contenido como con el proceso de correspondencia se comentarán en más detalle en la próxima sección 1.4.3 y en el capítulo 4.

Cuando hablamos de *stereo matching* o correspondencia estéreo en sentido amplio, nos referimos al conjunto de las tres fases que lo componen:

1. Problema de calibración y rectificación (capítulo 3).
2. Problema de correspondencia (capítulo 4).
3. Reconstrucción o reproyección (capítulo 4).

En los capítulos posteriores profundizaremos en el problema de *stereo matching*. En el capítulo 3 presentaremos los fundamentos teóricos de la calibración y rectificación previas al proceso de correspondencia estéreo y dedicaremos el capítulo 4 para describir métodos y algoritmos de resolución del problema de correspondencia, analizados mediante evaluaciones y también para explicar cómo obtener el valor de profundidad en la fase de reconstrucción.

1.4.3 Problemas a enfrentar

Como se ha planteado en los apartados anteriores, una parte fundamental del proyecto *ImmersiveTV* es la **inserción de objetos sintéticos interactivos dentro del video y la posibilidad de interacción por parte del usuario**. Para ello es necesario utilizar un módulo de superposición gráfica. Este módulo debe cumplir una serie de requisitos que se enumeran a continuación, junto con algunas tareas que debe ser capaz de realizar:

1. Búsqueda de **correspondencias estéreo** y generación de mapa de profundidad (problema de *stereo matching* presentado en la sección 1.4.2).
2. **Formato de datos 3D** capaz de describir la geometría, posición y acciones de los objetos sintéticos interactivos.
3. **Inserción de objetos 3D:** cálculo de oclusiones y fusión del objeto con el entorno.
4. **Reproducción de gráficos sobre vídeo.**
5. **Interacción con objetos:** ejecución de las acciones asociadas a los objetos.
6. **Sincronización** entre pantallas en el caso de sistemas de proyección multipantalla.

En este trabajo nos vamos a centrar únicamente en el punto 1 de la lista, ya que es necesario inferir **información de profundidad** para calcular correctamente las oclusiones de los objetos con el contenido del vídeo. Como sabemos, en el proyecto *ImmersiveTV* se trabaja con una fuente de vídeo estereoscópico cuya salida es un par de imágenes estéreo correspondientes a la perspectiva del ojo derecho e izquierdo respectivamente. A partir de la disparidad entre los píxeles de las dos imágenes y los parámetros de la cámara, es posible inferir la profundidad de la escena que se está visualizando y por lo tanto regenerar la imagen que se muestra para incluir datos sintéticos a una determinada profundidad. La inclusión de este tipo de información incrementa las posibilidades de interacción del usuario con los contenidos tradicionales de la televisión.

Para abordar la extracción de información 3D fiable, necesitamos un algoritmo de correspondencia estéreo capaz de enfrentarse a algunos **problemas** que pueden presentarse y que vienen asociados, por un lado a las **características del contenido** (en este caso vídeo deportivo) y por otro, al propio **proceso de cálculo de correspondencias**. La tabla 1.1 presenta un resumen de los más comunes, clasificados por su naturaleza.

Tipo de problema	Problema	Causas
Características de eventos deportivos	Tiempo de procesado	Eventos en directo, contenido real, imágenes o vídeos de alta resolución
	Cambios en parámetros de cámara	Zoom, cambios (repentinos) entre cámaras
	Cambios en el rango de disparidad	Zoom, cambios (repentinos) entre cámaras
	Disparidad pequeña o indetectable	Distancia a la escena, ángulo de captura
	Error residual de movimiento	Movimientos rápidos de jugadores y objetos
Proceso de correspondencia estéreo	Ausencia de textura	Regiones homogéneas o de bajo contraste
	Discontinuidades	Bordes de objetos, planos 3D diferentes
	Oclusiones	Movimientos, objetos en diferentes planos, entorno
	Condiciones de iluminación	Tiempo, exterior/interior, día/noche, reflexiones, diferentes reflexiones entre vistas

TABLA 1.1 – Ejemplo de problemas que se plantean al trabajar con el proceso de correspondencia estéreo sobre vídeos de eventos deportivos.

1.4.3.1. Problemas derivados del contenido

Las características propias del contenido de vídeo sobre el que se va a aplicar el cálculo de correspondencia estéreo influyen directamente sobre el rendimiento y la calidad del resultado. Por un lado, es fundamental que el **tiempo de procesado** sea el mínimo posible ya que estamos planteando trabajar con eventos en directo en los que la **latencia** y el **retardo** son factores críticos. Sin embargo, al tratar con imágenes o vídeos de **alta resolución con contenido real** donde la estructura de la escena no está controlada, el tiempo de computación se incrementa con respecto al caso de información de menor resolución, por lo que habrá que lograr un compromiso entre la calidad y la eficiencia del algoritmo de correspondencia.

Las **condiciones de captura** determinan cómo será el contenido adquirido, por lo que sus características también afectan al resultado. En concreto, factores como el **zoom** o los **cambios más o menos bruscos entre distintos puntos de vista** alteran o modifican los **parámetros de cámara**, que suele ser necesario conocer para obtener la información de profundidad, y pueden cambian el **rango de disparidad** (máxima y mínima disparidad posible entre vistas). Sin embargo, al tratarse de un acontecimiento

deportivo, se puede conocer previamente la posición de las cámaras permitiendo sortear algunos de estos problemas. Además, la **distancia a la escena**, que generalmente es grande excepto en los planos cortos y primeros planos, puede dar lugar a **valores de disparidad muy pequeños o casi indetectables** obligando a que el algoritmo de correspondencia estéreo tenga una resolución mayor. Esto no siempre es posible, ya que la resolución del mapa de disparidad es una limitación dada por los parámetros de los algoritmos de correspondencia estéreo.

Otro tipo de aspectos relacionados con la captura son la **distorsión de la lente de cada cámara**, que da lugar a una aberración distinta para cada ojo, y aquellas desviaciones asociadas al proceso de digitalización como son el **error de cuantificación**, que produce diferencias de intensidad en cada vista, y los **efectos del submuestreo**, que generan artefactos distintos en cada imagen, aliasing o píxeles desplazados. El **ruido** tanto de la propia escena, que se puede presentar en forma de niebla o sombras, como el del sensor, que da lugar a patrones de ruido distintos en cada vista, dificulta la búsqueda de correspondencias.

Finalmente, trabajar con **vídeo** presenta otro tipo de problemas relacionados principalmente con el movimiento. A parte de los cambios de plano, cámara o punto de vista que ya hemos comentado, el movimiento de las cámaras y de los elementos de la escena, como los jugadores u objetos (el balón), dan lugar a **ruido o error residual de movimiento** que se manifiesta en forma de halos, emborronamientos u objetos fantasma. También se producen cambios en las condiciones de iluminación, obligando a modificar los parámetros del algoritmo de correspondencia en tiempo real.

1.4.3.2. Problemas derivados del *stereo matching*

El proceso de inferencia de información de profundidad por medio de técnicas de *stereo matching* tiene ciertas **restricciones** que limitan su eficiencia y calidad. Las principales restricciones que caracterizan la correspondencia estéreo se definen en el capítulo 4 (sección 4.2.2), por ahora comentaremos en qué medida el contenido afecta a algunas de ellas:

- Las zonas con **ausencia de textura** que pueden encontrarse en la superficie sobre la que se practica el deporte (e.g. hierba, tarima, hielo o agua) dan lugar a ambigüedades en el resultado.
- Se producen **discontinuidades** o **saltos de disparidad** en los bordes de todos los objetos que no forman parte del estadio (e.g. público, jugadores o balón).
- Se producen **oclusiones** debido al movimiento de los jugadores y objetos (balón) que ocultan otros elementos de la escena. Se puede producir la situación en la que dos jugadores con la misma equipación y próximos en la imagen sean considerados uno solo.

Otro tipo de cuestiones a tener en cuenta están relacionadas con la elección de la configuración y los parámetros de los algoritmos de correspondencia, como puede ser el tipo de métrica utilizada para comprobar la semejanza (función de coste), tamaño de ventana, rango de búsqueda o asumir superficies fronto-paralelas. Profundizaremos en estos conceptos más adelante.

Las condiciones de captura también influyen sobre los resultados. Por ejemplo, el **paralaje vertical** producido por una mala alineación vertical de las cámaras impide el correcto funcionamiento de una gran cantidad de algoritmos de correspondencia. Por supuesto, las **condiciones de iluminación** son ampliamente variables y dependen de muchos factores como las condiciones meteorológicas, la hora del día (día o noche) o si la competición se realiza en interior o exterior, que exige un control sobre el tiempo de exposición de la cámara. Se denomina **distorsión radiométrica** a la variación de intensidad luminosa de la vista de un ojo con respecto de la del otro, que habitualmente viene dada por una constante multiplicativa (ganancia) y una variación aditiva (*offset*). Este fenómeno exige la normalización de iluminación. Además, habitualmente se asumen **superficies lambertianas**, es decir, aquellas en las que la luz se refleja uniformemente en todas direcciones.

Una vez planteado el problema que nos ocupa con todas sus características, en los siguientes capítulos describiremos la solución adoptada.

Capítulo 2

El ciclo de vida de la 3DTV: fundamentos teóricos y tecnologías

2.1 Introducción

La retransmisión de eventos 3D en directo es una actividad que ha salido recientemente del terreno experimental, debido a los altos costes de producción y escasa madurez de la tecnología existente. En los últimos años, los avances técnicos han empezado a ofrecer la posibilidad de que este tipo de eventos sea posible y que, en un futuro próximo, llegue a ser una realidad cotidiana. La reducción de los costes de producción será uno de los principales factores que impulsen el desarrollo de mejores tecnologías facilitando la introducción del 3D en la mayoría de salas de cine y en la televisión. Por lo general, en la actualidad el tipo de eventos en directo que se producen en 3D son los **deportes** y los **conciertos**, pero la técnica es fácilmente generalizable, siendo posible retransmitir en directo todo tipo de programas en 3D.

Por otra parte, desde 2010 ya existe gran variedad de contenidos 3D producidos y grabados. La aparición de las primeras cámaras 3D domésticas junto con los sistemas de conversión 2D a 3D han permitido al usuario generar sus propios contenidos. Además, la existencia de soportes de almacenamiento como el Blu-Ray han impulsado el mercado de cine y videojuegos en 3D para el hogar. Por todo esto, las tecnologías 3D son objeto de estudio y están en continuo desarrollo y evolución.

Para comprender un poco cómo funciona el 3D, en este capítulo comenzamos definiendo algunos términos como la estereoscopía y estereopsis, en los que se basan algunas

herramientas de visión artificial para resolver determinados problemas relacionados con la información tridimensional de una escena visual. A continuación, se introduce el tema de la televisión 3D, haciendo un recorrido por algunas de las fases del ciclo de vida, como la generación y producción de contenidos, los formatos de transmisión y los modos de visualización de contenidos.

2.2 Fundamentos de estereoscopía y visión estereoscópica

Se denomina **estereopsis** (de στερεός (*stereos-*), sólido y ὄψις (*-opsis*), visión o vista) al fenómeno de percepción visual por el cual, a partir de dos imágenes ligeramente diferentes de la misma escena proyectadas en la retina de los dos ojos, el cerebro es capaz de percibir sensación de profundidad [12].

Con la **visión binocular** se captan dos imágenes distintas de la escena debido a que los ojos ocupan posiciones diferentes en la cabeza. La separación entre los ojos se denomina **distancia interocular (DIO)** o interpupilar y oscila entre 45 y 75 mm., tomando como valor medio típico 65 mm. La diferencia entre ambas imágenes se conoce como **disparidad binocular**, **disparidad horizontal** o **disparidad retiniana** ya que la proyección de un punto sobre la retina de un ojo está desplazada en la dirección horizontal con respecto a la proyección sobre la retina del otro ojo [10, 33]. La sensación de profundidad depende de la disparidad, y la disparidad de la DIO: cuanto mayor es la distancia interocular, mayor es la disparidad y se capta mejor la profundidad de objetos lejanos; sin embargo, a menor DIO, menor disparidad y mejor captación de la profundidad de los objetos cercanos.

El término estereopsis se usa generalmente como abreviatura de visión binocular, percepción binocular de profundidad o percepción estereoscópica de profundidad aunque, estrictamente hablando, la sensación de profundidad asociada con la estereopsis se puede obtener bajo otras condiciones como, por ejemplo, en el caso de un observador que ve la escena con un único ojo mientras se mueve [11]. El movimiento del observador produce diferencias en la imagen retiniana simple a lo largo del tiempo de manera similar a la disparidad binocular. Este efecto se conoce como **paralaje de movimiento**. Otras señales visuales que el cerebro utiliza para percibir volumen, distancia y profundidad son [9]:

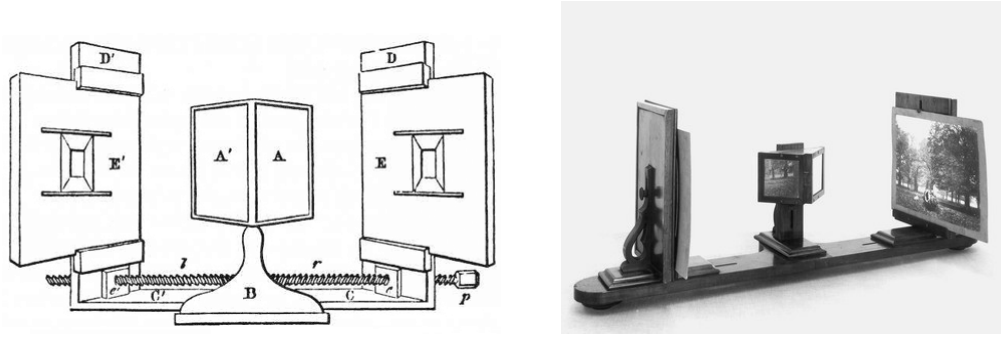
- **Superposición:** un objeto superpuesto a los demás se interpreta como situado más cerca del observador.
- **Perspectiva o puntos de fuga:** mecanismo descrito por Leonardo Da Vinci que consiste en la capacidad de calcular la distancia entre los objetos en base a la perspectiva. Por ejemplo, si observamos unas vías de tren que, según la perspectiva, parecen juntarse en el infinito y con árboles en los laterales, podemos tener una idea de la distancia entre árboles y la distancia que nos separa de ellos.
- **Tamaño de objetos conocidos:** se trata de la información que ya tenemos interiorizada acerca de algunos objetos, como el tamaño relativo entre diferentes objetos.
- **Tamaño de la imagen sobre la retina:** el cerebro calcula la distancia a un objeto en función del tamaño de la proyección sobre nuestra retina. Por ejemplo, el tamaño de un objeto en la retina aumenta a medida que disminuye la distancia que nos separa de él.
- **Convergencia:** se trata del punto en el que intersecan los ejes ópticos. También se llama **punto de divergencia cero (o disparidad cero)**. Cuanto más cercano sea un objeto, mayor será la convergencia. Este fenómeno determina la estereopsis, puesto que a partir de cierta distancia no se puede distinguir la profundidad (según la persona, entre los 60 y 100 metros) ya que en puntos muy alejados los ejes ópticos son casi paralelos.

Como observación importante, la estereopsis no está presente cuando se observa una escena con un solo ojo sin movimiento o cuando se ve la imagen plana de una escena con ambos ojos. Habitualmente, tampoco está presente cuando una persona con visión binocular anormal (estrabismo) mira la escena con ambos ojos. Incluso en los tres casos anteriores donde no existe estereopsis, los humanos son capaces de percibir relaciones de profundidad. Sin embargo, la visión binocular es la mejor fuente de información que el cerebro utiliza para calcular profundidad en la escena visual.

Se denomina **estereoscopía** (de στερεός (*stereos-*), sólido y σκοπέω (*-skopeō*), mirar o ver) a la técnica capaz de recoger información visual tridimensional y/o crear o realzar la ilusión de profundidad a partir de una imagen estereográfica, un estereograma o una imagen tridimensional [10, 33]. Se trata de inferir información de profundidad a partir de las dos imágenes bidimensionales que representan un par estéreo basándose en la estereopsis o visión binocular, es decir, presentando una imagen separada para cada

ojo tal y como veríamos la escena en la realidad. Como ya sabemos, las dos imágenes bidimensionales **se fusionan** en el cerebro otorgando la percepción de profundidad 3D.

La primera fotografía estereoscópica se capturó con una sola cámara realizando una traslación horizontal de 6 cm para simular la DIO. Para ver una imagen estereoscópica es necesario un medio capaz de presentar cada imagen al ojo correcto para que el cerebro interprete adecuadamente la información. El primer dispositivo que permitió la visualización de este tipo de imágenes es el **estereoscopio**. Según la RAE [34], « *aparato óptico en el que, mirando con ambos ojos, se ven dos imágenes de un objeto, que, al fundirse en una, producen una sensación de relieve por estar tomadas con un ángulo diferente para cada ojo* ». Típicamente, un estereoscopio dispone de una lente para cada ojo que aumenta el tamaño y la distancia aparentes de la imagen vista a través de ella. Además, desplaza la posición horizontal de forma que los bordes de ambas imágenes se funden en una ventana estéreo que parece contener a los objetos en su interior.



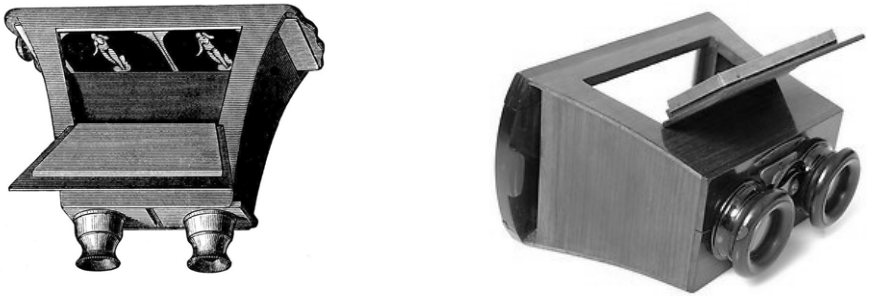
(a) **Boceto** (fuente: [35]) (b) **Estereoscopio**[1]

FIGURA 2.1 – **Estereoscopio de Wheatstone**

El científico e inventor inglés Sir. Charles Wheatstone inventó el estereoscopio, que supuso una revolución en su época. De hecho, fue Wheatstone quien definió oficialmente la estereopsis dentro de su estudio sobre la visión binocular en 1838 [36], por el cual recibió un premio de la *Royal Society* de Inglaterra. El **estereoscopio de Wheatstone** consta de dos espejos girados 45º con respecto a los ojos del usuario (etiquetados como A y A' en la figura 2.1), cada uno de los cuales reflejan una imagen colocada en el lateral del ojo correspondiente (etiquetadas como E y E').

[1]Imagen tomada de: http://www.ssplprints.com/image/129529/reflecting-stereoscope-originally-used-by-charles-wheatstone-19th-century

Posteriormente, Sir. David **Brewster** presentó en 1845 una herramienta similar introduciendo lentes correctivas para enfocar las imágenes a una distancia más cercana, reduciendo el tamaño del dispositivo [37]. Este equipo utilizaba pequeños pares de transparencias en lugar de las grandes láminas laterales del invento de Wheatstone, tal y como se observa en la figura 2.2.



(a) **Boceto**[2] (b) **Estereoscopio**[3]

FIGURA 2.2 – **Estereoscopio de Brewster**

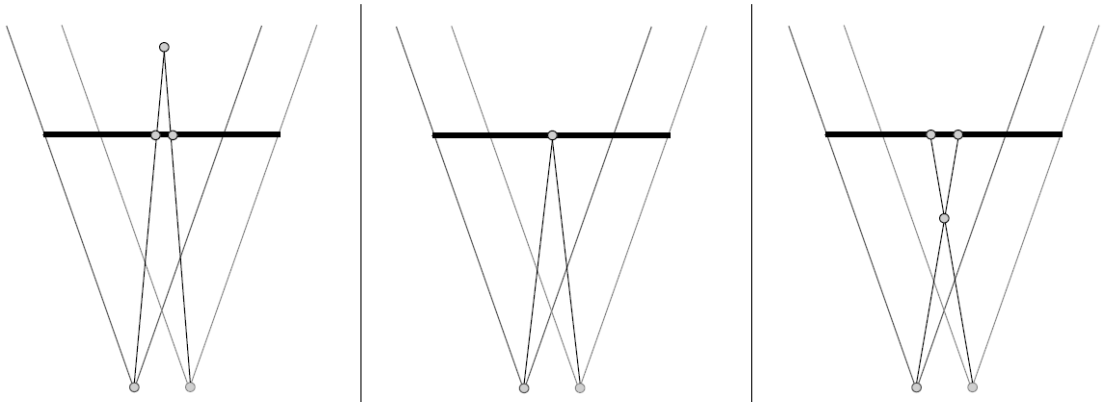
Se denomina **campo visual** a la porción del espacio que cada ojo es capaz de ver y se mide en grados. La **acomodación** es el enfoque óptico de un punto, es decir, hacer coincidir dicho punto en el plano de proyección (en el caso de la visión, la retina), y la **fusión** es el proceso conjunto de acomodar un punto y hacer converger los ejes ópticos en él, lo que significa hacer que los ejes ópticos intersequen (punto de disparidad cero) coincidiendo con el plano de proyección [10, 33]. En estos términos, definimos la **imagen estéreo** como la fusión de la imagen del ojo derecho con la del ojo izquierdo. Cada uno de los dos elementos que componen la imagen estéreo recibe el nombre de campo.

Un concepto fundamental de la estereoscopía es el **paralaje** (*paralax*). Se trata de la distancia entre las dos proyecciones de un mismo punto sobre el plano de proyección (figura 2.3). El paralaje determina la disparidad y depende de la distancia interocular, como ya se ha comentado, a mayor DIO, mayor paralaje y disparidad [15]. A continuación se enumeran los tipos de paralaje que podemos encontrarnos:

- Según el eje: puede ser horizontal o vertical. Es un tipo de paralaje que no debe producirse y no haremos demasiado hincapié.

[2]Fuente: http://en.wikipedia.org/wiki/File:PSM_V21_D055_The_brewster_stereoscope_1849.jpg
[3]Fuente: http://www.ebay.com/itm/RARE-EARLY-UNIS-FRANCE-BREWSTER-TYPE-WOOD-ANTIQUE-3D-STEREO-VIEWER-STEREOSCOPE-/151040299088

- Según la distancia: consiste en la separación de las proyecciones sobre la pantalla.
  - **Paralaje cero:** los puntos de imagen homólogos de ambas imágenes coinciden en la misma posición sobre el plano de proyección (figura 2.3(b)).
  - **Paralaje positivo:** se puede dar el caso en el que los ejes de ambos ojos son paralelos, para objetos lejanos al observador, además de aquel en el que los ejes ópticos convergen. La imagen que se produce parece estar **detrás del plano de imagen** (figura 2.3(a)). Cuando la distancia entre las imágenes es mayor que la DIO, los ejes ópticos divergen, pero esto no ocurre al visualizar objetos en el mundo real.
  - **Paralaje negativo:** los ejes ópticos se cruzan y los objetos parecen estar **delante del plano de imagen**, es decir, entre la pantalla y el observador (figura 2.3(c)).



(a) **Paralaje Positivo** (b) **Cero Paralaje** (c) **Paralaje Negativo**

FIGURA 2.3 – Tipos de paralaje [10].

Antes de finalizar esta sección, enumeramos algunos de los problemas a tener en cuenta en el campo de la estereoscopía [10]:

- **Relación acomodación/convergencia:** como ya sabemos, el punto de enfoque no es el mismo que el punto de convergencia. Tal y como se ha definido, la fusión consiste en hacer coincidir ambos puntos mediante la acomodación en la pantalla. Los puntos de paralaje cero no presentan este problema, pero para el resto de puntos existe dificultad en la fusión produciendo incomodidad. Esta relación depende de las características del observador, del tamaño de la pantalla (preferiblemente grande), de la distancia d al plano de proyección (mejor cuanto mayor sea) y del tipo de paralaje que sea positivo, tratando de no superar los 1,5º siguiendo la expresión: $paralax_{max} = 2 \cdot d \cdot tan\left(\frac{1{,}5^\circ}{2}\right)$.

- **Imágenes congruentes:** las dos imágenes que muestran la misma escena deben tener brillo congruente e igual intensidad y contraste. El incumplimiento de estos requisitos dificulta la fusión.
- **Bordes:** las imágenes que no están contenidas totalmente en la pantalla generan problemas sobre todo en el caso de las imágenes con paralaje negativo, cuyos objetos parecen estar delante de la pantalla. Se aprecia un efecto de incomodidad visual en la intersección de los objetos con los bordes de la pantalla. Por lo general, se tiene más tolerancia al recorte horizontal que al vertical.
- **Imágenes cruzadas:** cuando un ojo capta parte o la totalidad de una imagen que no le corresponde, se produce un efecto de imagen fantasma o *ghosting*. La causa suele ser tecnológica: problemas con filtros debido al movimiento de la cabeza del espectador, problemas de sincronización entre dispositivos o problemas de latencia de la pantalla o proyectores.
- **Paralaje vertical y/o divergente:** como hemos comentado antes, nunca debe producirse. Provoca dificultades en la fusión, incomodidad e incluso malestar y obliga al ojo a realizar un movimiento antinatural. Deriva principalmente de los movimientos del usuario y de la utilización de algoritmos equivocados.

La **visión artificial** o ***computer vision*** es la disciplina tecnológica que se encarga de adquirir, analizar y comprender las imágenes mediante un conjunto de herramientas informáticas y matemáticas. Comprende un gran abanico de campos, entre los que se incluye la **correspondencia estéreo** (o ***stereo matching***) o la **reconstrucción 3D basada en imagen** (***image-based 3D rendering***), que explicaremos a lo largo de los siguientes capítulos.

2.3 Televisión 3D

2.3.1 Definición y ciclo de vida de la televisión tridimensional

Según [38], la televisión 3D (3DTV por sus siglas en inglés) es la adición de profundidad o volumen a las imágenes de televisión. Los actuales sistemas trabajan con dos imágenes (izquierda y derecha) dispuestas de manera que el espectador vea cada una de ellas con el ojo correspondiente. El cerebro es engañado para interpretar las diferencias

en las imágenes (**disparidad binocular**) como profundidad en la escena, ya que vería las mismas dos imágenes si la profundidad estuviera realmente presente.

El aumento de la **interactividad** del usuario con el contenido y la **inmersividad** más allá del actual 3D son objeto de investigación con el objetivo de que el usuario final experimente la sensación de pertenecer al propio escenario que está visualizando al poder interactuar con él. La **realidad virtual** permite un alto grado de interactividad ya que el usuario es capaz de modificar la escena visualizada con ayuda de herramientas y dispositivos auxiliares. Por otro lado, los **contenidos en tres dimensiones** aumentan la inmersividad logrando que el usuario perciba el entorno de forma cercana a como se hace en la realidad. Por este motivo, se busca generar unos buenos contenidos tridimensionales y preservar la máxima calidad a lo largo de la cadena de valor de difusión de contenidos televisivos.

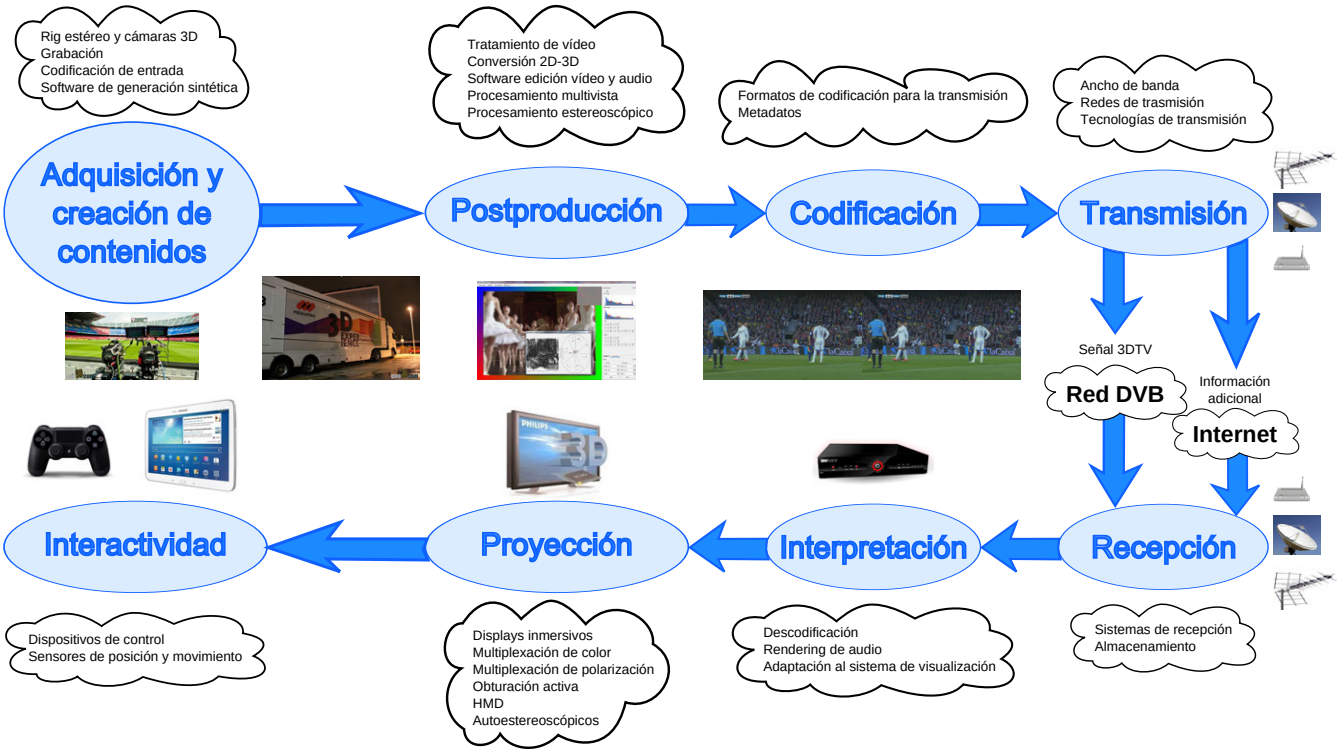


FIGURA 2.4 – **Ciclo de vida del contenido en 3DTV.**

Las principales fases del ciclo de vida del contenido de televisión tridimensional (figura 2.4) son las siguientes:

- **Adquisición y creación de contenidos:** obtención del contenido, ya sea por medio de la grabación con cámaras de eventos en directo para su posterior difusión, como de la creación de contenidos a través de herramientas de diseño.

- **Postproducción:** postprocesado del contenido para corregir errores, añadir información adicional o elementos como rótulos de texto, aplicar filtros como la normalización o la ecualización del histograma, y transformaciones como la rectificación de imágenes.
- **Codificación:** composición del vídeo que se va a enviar al canal de transmisión. Habitualmente suele comprender dos etapas: procesado de fuente para organizar y agrupar las vistas izquierda y derecha, y codificación del vídeo 2D resultante utilizando los estándares ya existentes.
- **Transmisión:** encapsulado del contenido 3D para obtener el flujo de datos que es difundido a través de las infraestructuras de cable, terrestres, satelitales y canales de banda ancha.
- **Recepción:** desencapsulado y recuperación del contenido 3D.
- **Interpretación:** descodificación de las vistas izquierda y derecha a partir del vídeo 3D.
- **Proyección o visualización:** presentación del contenido al usuario.
- **Interactividad:** funcionalidades que permiten al telespectador modificar elementos de la escena visualizada.

En este capítulo nos centramos en las fases que consideramos más importantes para el desarrollo de este trabajo. La primera de ellas es la **generación de contenido**, que se engloba dentro de la etapa de adquisición y creación de contenidos. En la sección 2.3.2 se describe brevemente la configuración de las cámaras y dispositivos habituales de **captura de contenidos**. La segunda etapa considerada forma parte de la transmisión. La sección 2.3.3 explica los **formatos de transmisión** para comprender cómo es la señal que llega al receptor. Finalmente, la sección 2.3.4 muestra algunas de las principales técnicas de **presentación del contenido** al usuario.

2.3.2 Generación de contenidos 3D

En este trabajo nos centraremos en los **contenidos inmersivos capturados o generados para la televisión 3D**. El contenido de televisión está compuesto generalmente por uno o varios puntos de vista de una escena, que se alternan mediante cambios de cámara, además de distintas escenas que van cambiando y que disponen a su vez de diversos puntos de vista. De esta manera, en un televisor se presenta un punto de vista

de una escena en un determinado momento. En televisión tridimensional, cada punto de vista puede ser generado como una imagen estéreo compuesta por una vista para cada ojo, o bien por una vista RGB convencional junto con información de profundidad. Existen básicamente tres **técnicas de creación de contenido**:

1. Generación de imágenes sintéticas por ordenador.
2. Filmación digital de varias vistas de la escena mediante una o varias cámaras.
3. Conversión del material de vídeo 2D existente a 3D.

En la fase de adquisición y creación de contenido tridimensional se debe tener en cuenta una serie de factores que afectan a la experiencia del usuario en términos de sensación de profundidad. En primer lugar, se debe cuidar las condiciones de iluminación, evitando la existencia de brillos y sombras en las superficies de la escena, además de la obtención de color realista. Por otra parte, es necesario que las señales de vídeo y de contenidos 3D adicionales, que se generan en una etapa de preproducción, estén debidamente sincronizadas en tiempo de producción.

En el caso del contenido que lleva asociado información de profundidad, en la práctica, esta se almacena en forma de **mapa de profundidad**. Se trata de una imagen en escala de gris en la que el valor de intensidad de cada píxel está relacionado con la distancia del objeto a la cámara. Es decir, la imagen representa de manera visual la distancia comprendida entre la cámara y cada punto de la escena. Según el convenio escogido, los valores de menor intensidad (más oscuros) pueden representar distancias más lejanas y los de mayor intensidad (más brillantes) distancias cercanas, o viceversa. Adicionalmente, se debe especificar la relación entre intensidad y distancia (e.g. definiendo el valor más cercano y más lejano del rango de intensidad). En la figura 2.5 se muestra una vista de una escena y su mapa de profundidad asociado.

(a) Vista izquierda (b) Mapa de profundidad

FIGURA 2.5 – Vista izquierda de la escena *cones* y su mapa de profundidad [23, 39].

Como veremos en la sección 2.3.3.5, la utilización de formatos de transmisión basados en la información de profundidad facilita la grabación de las imágenes y simplifica la postproducción, ya que sólo se necesita manipular una imagen. Sin embargo, como inconveniente, aumenta la complejidad en el receptor ya que se debe reconstruir la segunda vista a partir de la primera y el mapa de profundidad.

Para obtener la información de profundidad en una escena se puede capturar directamente utilizando una **cámara de profundidad** (*depth camera*) o de rango (*ranging camera*), también conocida como cámara RGB-D, Z-cam, tiempo de vuelo (*time-of-flight* (ToF)) o LIDAR (radar láser sin escáner mecánico) [40], que consiste en una cámara RGB convencional con un dispositivo acoplado que proyecta patrones de luz estructurada infrarroja, de manera que, por medio de un sensor, es capaz de calcular la distancia a los objetos basándose en la velocidad de la luz y el tiempo que tarda en regresar. Un ejemplo de cámara de profundidad de bajo coste es el conocido dispositivo Kinect^{TM}de Microsoft. Otra opción es la obtención de información 3D a partir de información 2D, como en el caso del vídeo monoscópico, mediante técnicas de **reconstrucción a partir del movimiento (*structure from motion*)** [41]. Sin embargo, por lo general el contenido 3D para televisión es capturado mediante cámaras estereoscópicas, de manera que es necesario otro tipo de tratamiento para la obtención de profundidad como la utilización de técnicas de visión artificial en el campo de la **correspondencia estéreo (*stereo matching*)**.

Actualmente, la mayoría del material 3D que se difunde ha sido generado usando una cámara con configuración dual, proporcionando un par estéreo, donde las vistas del ojo izquierdo y derecho están grabadas por separado desde dos perspectivas ligeramente diferentes. La obtención de los **parámetros de las cámaras mediante calibración** es un aspecto que trataremos en el capítulo 3.

Los parámetros de grabación como la **distancia base de la cámara** (separación entre las dos cámaras), la **distancia de convergencia** (distancia desde la línea de base hasta la intersección de los ejes ópticos) y la **distancia focal** de las lentes de las cámaras, se pueden utilizar para escalar la disparidad horizontal y, por lo tanto, el rango de profundidad percibida [13, 14]. Se puede escoger entre dos tipos de configuración: la configuración de **cámaras en paralelo** y la configuración de **cámaras convergentes**. En muchas ocasiones se recomienda la configuración en paralelo para evitar distorsiones

geométricas, como la distorsión trapezoidal (de piedra angular o *keystone distortion*) o la curvatura de plano de profundidad (*depth plane curvature*) [42]. Por otra parte, la posibilidad de ajustar los pares de imágenes en postproducción es limitada y requiere gran cantidad de tiempo [43].

FIGURA 2.6 – **Rig estéreo: configuración de cámaras en paralelo [44]**

Las configuraciones de cámara doble se instalan sobre unos soportes llamados **rigs**, como el de la figura 2.6 que consiste en una configuración de cámaras en paralelo. Existen multitud de fabricantes con distintas configuraciones, precios y prestaciones. Para los eventos 3D en directo, es prácticamente obligatorio el uso de rigs robotizados, cuyos movimientos se pueden aplicar a las dos cámaras por igual con total precisión, así como lentes robotizadas, que hacen lo propio con los ajustes de apertura (obturación), foco o zoom. Es posible usar rigs fijos en las producciones 3D en directo, pero su uso impide la modificación de los parámetros de estereoscopía y del zoom, limitando el aprovechamiento del potencial de la cámara.

Existen en el mercado **cámaras de doble óptica** como la de la figura 2.7(a), con **interaxial fija**, es decir aquellas en las que la distancia entre los ejes ópticos permanece constante. Este tipo de cámaras obtienen resultados aceptables para televisión, aunque cuentan con una limitación en la distancia mínima de enfoque o captura, que oscila entre 2 y 2,5 metros, ya que a menor distancia producen problemas de visión. Por este motivo, no pueden ser utilizadas en cualquier posición de un evento en directo, reduciéndose su uso al de cámara de apoyo o para obtener planos con estabilizadores de cámara del tipo *steadycam* (figura 2.7(b)). Sin embargo, estas cámaras tienen una configuración más fácil de manejar que el caso del rig con dos cámaras.

Los ajustes de posición de la cámara, como son las modificaciones de interaxial o de angular (convergencia), son distintos en función del destino para el que se generan los contenidos (televisión 3D, pantalla de cine,...) ya que el tamaño de la pantalla es muy

(a) **Panasonic AG-3DP1 [45]** (b) ***Steadycam* [46]**

FIGURA 2.7 – Cámara de doble óptica (izquierda) y estabilizador de cámara (derecha)

diferente y el paralaje máximo tolerable varía en cada caso. En el caso de contenidos creados para varios tipos de pantalla, los ajustes se hacen siempre para la pantalla más grande, limitando la experiencia inmersiva de los usuarios de la pantalla más pequeña.

2.3.3 Formatos de transmisión

Pese a que el mercado de la televisión 3D está preparado para un enorme despegue, las tecnologías de transmisión son de muy reciente creación por lo que aún no se ha establecido un estándar de uso general para la difusion de contenidos 3D, aunque la especificación DVB-3DTV es un claro candidato a ocupar dicho puesto. Por este motivo, permanece la incertidumbre sobre los formatos y métodos de transmisión que se impondrán en un futuro a medio y largo plazo. Por ahora, a pesar del gran número de técnicas diferentes que se usan para codificar vídeos 3D, todos los televisores de alta definición 3D son capaces de procesar la señal tridimensional entrante y recodificarla sobre la marcha usando una especificación y formato que les permite presentar el contenido en pantalla.

Debido a la cantidad de formatos 3D que existen, entender la terminología y cómo trabajan todos ellos puede parecer una tarea imposible. En esta sección explicamos algunos de los formatos más habituales que nos podemos encontrar.

2.3.3.1. El estándar DVB-3DTV

El consorcio que forma el ***Digital Video Broadcasting Project* (DVB)** ha formalizado un estándar de difusión de contenidos 3D para acelerar su adopción por parte de la industria audiovisual centrada en el hogar. El 17 de febrero de 2011 se publicó la especificación para la televisión en 3D en el *BlueBook A154* con el nombre *Frame Compatible Plano-Stereoscopic 3DTV*, más conocido como DVB-3DTV [31]. Esta especificación fue enviada al **ETSI** para su estandarización en noviembre de 2012 [32].

DVB-3DTV incluye técnicas y procedimientos para difundir una señal de vídeo 3D a través de la infraestructura de televisión HD (HDTV) ya desplegada que comprende sistemas por cable, terrestres, satelitales y canales de banda ancha, incluyendo a los usuarios con receptores (o ***set-top boxes* (STB)**) de HDTV. El canal de 3DTV cumple similares requisitos de tasa de bits que un canal HDTV comprimido sin pérdidas.

El estándar DVB-3DTV propone una implantación escalonada en dos fases:

- **Fase 1:** busca la compatibilidad hacia delante de los actuales sistemas HD, capaces de recibir señales 3DTV, tras la actualización de *firmware* del STB. Los formatos de señalización o distribución deben extenderse para soportar los nuevos formatos de codificación 3D. Se propone la utilización de formatos plano-estereoscópicos *frame compatible* (FC), explicados en la sección 2.3.3.2. Algunos operadores como Sky3D [47] o Canal+3D [48] ya utilizan esta tecnología, pensada principalmente para televisión de pago.
- **Fase 2:** plantea el uso de técnicas de **codificación de vídeo escalable (SVC)** y **codificación de vídeo multivista (MVC)** para evitar transmitir simultáneamente información 2D y 3D, ocupando solo un canal HDTV mediante la transmisión de un único flujo de vídeo bidimensional de base y una capa ajustable con información de profundidad 3D, de manera que los STB actuales no son compatibles. Se busca que los sistemas HDTV sean capaces de mostrar una versión 2D utilizando esquemas *service compatible* (SC), explicados en la sección 2.3.3.3. Este sistema está pensado principalmente para las cadenas en abierto.

DVB-3DTV es independiente del tipo de tecnología de presentación al usuario. La especificación además presenta opciones para señalizar la posición de los subtítulos de cada personaje de la escena en diferentes profundidades.

2.3.3.2. Formatos *frame compatible*

La especificación DVB-3DTV [31, 32] propone la adopción de los formatos 3D plano-estereoscópicos *frame compatible* en su primera fase de implantación. Los formatos ***frame compatible*** [49] son aquellos en los que la señal estéreo es el resultado de multiplexar las vistas izquierda y derecha en un único cuadro HDTV. Han tenido una buena acogida por parte de la industria ya que facilitan la introducción de servicios estereoscópicos a través de la infraestructura y equipamiento existentes. En la mayoría de los casos, permite también reutilizar los receptores (STB) del usuario ya desplegados para servicios HD.

La especificación **HDMI v1.4a** [50] ha sentado prececentes adoptando los formatos 3DTV plano-estereoscópicos *frame compatible* que son soportados por los dispositivos de visualización en funcionamiento. Por este motivo, los formatos especificados para servicios 3DTV se corresponden con los formatos que puede transmitir la conexión HDMI, que está sujeta a las resctricciones de los sistemas de difusión de HDTV existentes.

Los formatos *frame compatible* realizan primero un submuestreo espacial o temporal sobre las imágenes izquierda y derecha originales de resolución completa y a continuación empaquetan o multiplexan las subimágenes resultantes en un único cuadro que puede ser tratado por el demodulador y el descodificador del receptor como una imagen HDTV convencional. DVB-3DTV [31, 32] **contempla dos tipos de submuestreo y multiplexado espacial** (horizontal para *Side-by-side* y vertical para *Top-and-Bottom*). El flujo de vídeo *frame compatible* se ajusta a los requisitos del formato de vídeo HDTV [51], con relación de aspecto 16:9 y códec H.264/AVC [52], de manera que el codificador de las imágenes de vídeo no necesita conocer el formato de vídeo 3DTV plano-estereoscópico *frame compatible*, a excepción de la señalización en la capa de vídeo para diferenciar el tipo de flujo y poder alternar entre 2D y 3D. DVB-3DTV también especifica los tipos de cuadro admitidos (e.g. 720p, 1080i, etc), diferenciando cuáles pueden aplicarse en sistemas de vídeo de 25Hz y de 30Hz. Los flujos de audio y contenido auxiliar tienen el mismo formato que los de los servicios HDTV DVB.

En el formato ***Side-by-Side (SbS)***, o lado a lado (figura 2.8), las dos imágenes se **submuestrean horizontalmente a la mitad** de su resolución y a continuación se empaquetan en un cuadro conjunto de resolución HD cuya mitad izquierda contiene la

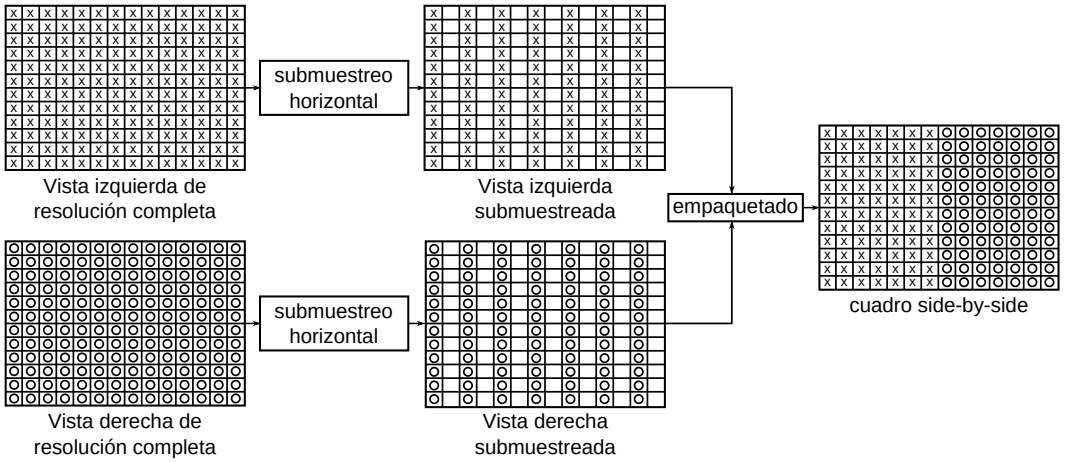


FIGURA 2.8 – **Formato *side-by-side*:** submuestrea horizontalmente las imágenes y las multiplexa en un único frame.

vista izquierda, dedicando su otra mitad para la vista derecha. El receptor desempaquetará las imágenes submuestreadas y las escalará, recuperando las resoluciones originales para generar de nuevo una secuencia de cuadros completos de vídeo para cada ojo. Este formato de transmisión es el más común por lo que es ampliamente utilizado por operadores de teledifusión, sobre todo en Europa.

El formato ***Top-and-Bottom* (TaB), *over-under* o encima-debajo** (figura 2.9) realiza un **submuestreo vertical** de las imágenes **a la mitad** de resolución y las empaqueta en un cuadro HD que contiene la vista izquierda en su mitad superior y la vista derecha en la inferior. El receptor demultiplexa las versiones diezmadas y las escala para obtener las resoluciones originales, generando la secuencia de cuadros completos. Este formato es utilizado para la difusión de contenido, sobre todo en norteamérica, y es apropiado para las retransmisiones deportivas por tener mayor resolución horizontal.

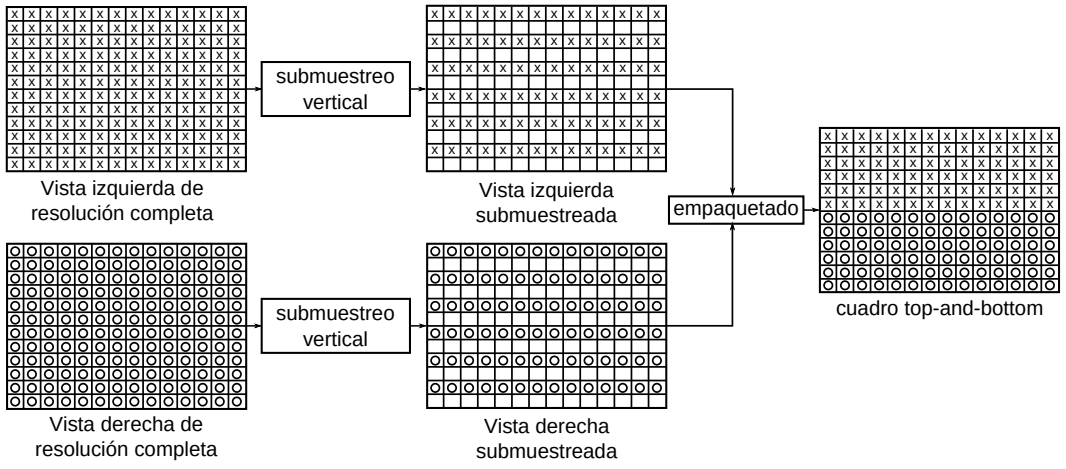
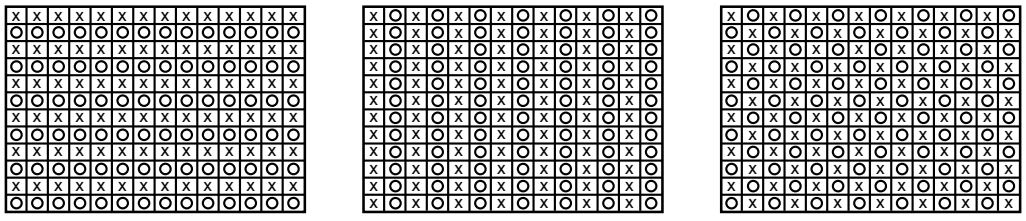


FIGURA 2.9 – **Formato *top-and-bottom* (over-under):** diezma verticalmente cada imagen las empaqueta en un único frame.

El diezmado o submuestreo puede realizarse preservando las muestras de las mismas columnas (o filas) en ambas vistas (diezmado común), eliminando las pares o las impares de las dos imágenes, o las columnas (o filas) complementarias (diezmado complementario), conservando las pares en una imagen y las impares en la otra. Hay que aclarar que, aunque por lo general utilizaremos los términos *Side-by-Side* y *Top-and-Bottom* para referirnos a las técnicas de las figuras 2.8 y 2.9 (submuestreo y empaquetado), estos términos sólo describen la disposición espacial del múltiplex tras el empaquetado (una mitad para cada vista) pudiendo ser combinados con cualquier tipo de submuestreo.

Además de los considerados por DVB-3DTV existen más formatos *frame compatible* como resultado de la combinación de diferentes tipos de submuestreo y de multiplexado. La figura 2.10 recoge tres ejemplos habituales de empaquetado FC. El primero (2.10(a)) es conocido como ***Line-by-Line* (LbL) o línea a línea**, que empaqueta las imágenes submuestreadas verticalmente, entrelazando las líneas de manera alterna entre las dos vistas. Análogamente, el formato ***Column-by-Column* (CbC) o columna a columna** (2.10(b)) entrelaza las columnas de píxeles de las imágenes submuestreadas horizontalmente. Por último, el formato ***Checkerboard*** (2.10(c)) entrelaza píxel a píxel las imágenes submuestreadas. En este caso el submuestreo puede ser tanto horizontal como vertical, o incluso tomando directamente el píxel de la misma posición.



(a) ***Line-by-Line*** (b) ***Column-by-Column*** (c) ***Checkerboard***.

FIGURA 2.10 – **Otros formatos de empaquetado *frame compatible***

Sin embargo, estos formatos introducen altas frecuencias y disminuyen la correlación entre las muestras, obligando a aumentar el ancho de banda. Los estudios de rendimiento y calidad de formatos *frame compatible* [49, 53–55] buscan técnicas de muestreo que conserven el balance entre la resolución vertical y horizontal, mejorando la calidad de imagen. Como ejemplo, el **submuestreo quincuncial o escalonado** consiste en un submuestreo horizontal y vertical simultáneo preservando la mitad de los píxeles en su posición original formando una estructura de tipo *checkerboard* obteniendo una mejora visual con respecto a los formatos anteriores con el mismo ancho de banda. No existe

un método genérico de submuestreo quincuncial por lo que han aparecido códecs propietarios como SENSIO® Hi-Fi 3D [54], que realiza un submuestreo quincuncial seguido del multiplexado en estructura *side-by-side* (figura 2.11, parte superior). Pero también se puede empaquetar en forma de estructura *checkerboard*, como se muestra en la parte inferior de la figura 2.11.

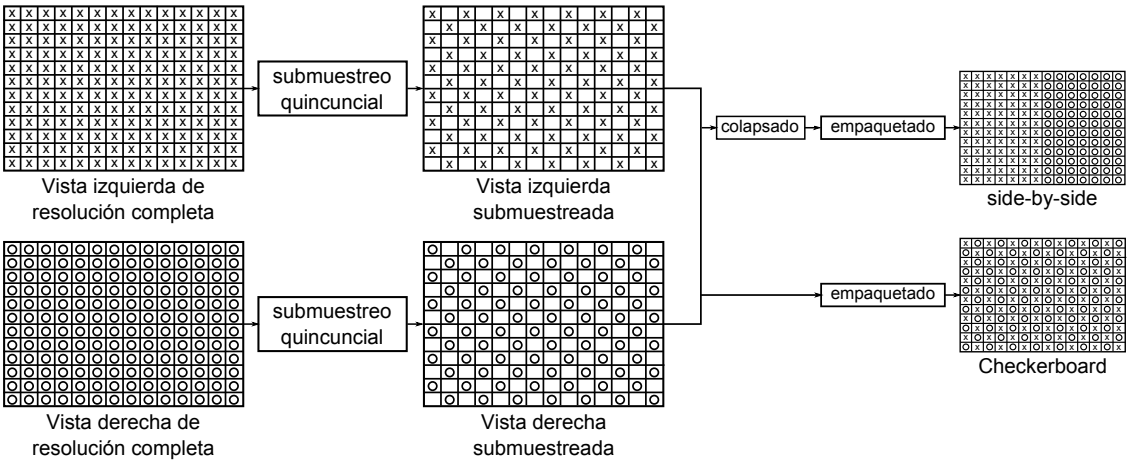


FIGURA 2.11 – **Formato basado en muestreo quincuncial:** cada imagen se submuestrea siguiendo el esquema quincuncial y se empaquetan juntas en un único frame utilizando alguno de los formatos anteriores.

Aunque se preserve la resolución horizontal y vertical, el submuestreo quincuncial pierde resolución diagonal. El principal problema que tiene es que para generar la imagen compuesta FC, los píxeles deben ser desplazados de su posición original para alinearlos y como consecuencia se pierde correlación, dando lugar a bordes dentados o abruptos que en la etapa de codificación pueden generar artefactos de compresión o requerir una mayor tasa de bit para mantener una calidad satisfactoria.

También es posible considerar formatos basados en el submuestreo temporal, como ***field sequential*** o secuencia de campos, que genera un flujo de vídeo entrelazado en el que los campos impares corresponden a una vista y los pares a la otra. Es un buen método para transmitir información estéreo sin duplicar el ancho de banda necesario a costa de perder resolución temporal ya que el vídeo resultante tiene la mitad de frecuencia que el original. Es una técnica bastante antigua que tiene problemas de *flickering* o parpadeo cuando se utiliza con televisión estándar porque cada ojo solo recibe la mitad de la tasa de campos total.

2.3.3.3. Formatos *service compatible*

Constituye la segunda etapa de la fase de implantación de DVB-3DTV [56, 57]. Consiste en disponer las imágenes en el múltiplex de manera que un receptor de HDTV existente sea capaz de extraer una versión 2D del contenido de vídeo a partir de un servicio 3DTV plano-estereoscópico *frame compatible*. Además de la retrocompatibilidad tecnológica, estos formatos tienen como ventaja la utilización del mismo ancho de banda que los actuales servicios HDTV, pero como contrapartida se tiene que transmitir simultáneamente la información 2D y 3D.

Los servicios 3DTV *Service Compatible* (SC) adoptan el formato de vídeo **MVC *Stereo High Profile***, que se trata de un perfil de MVC desarrollado específicamente para vídeo estereoscópico de dos vistas. El servicio de vídeo 3D contiene dos flujos elementales de vídeo:

- **Capa Base Estéreo MVC** o **Capa Independiente**, compatible hacia atrás con los flujos de vídeo HDTV, de manera que un IRD no compatible con 3DTV lo descodifica como si se tratase de un servicio HDTV convencional.
- **Capa Dependiente Estéreo MVC** que complementa a la capa base estéreo para proporcionar la segunda vista de un par estéreo a una tasa de bit reducida en comparación con una codificación independiente para un nivel de calidad de imagen equivalente.

Otro formato válido para servicios SC es el **SVC (*Scalable Video Coding*)** al disponer de una capa base compatible con los perfiles de H.264/AVC utilizados en los actuales servicios HD. La baja eficiencia que ofrecen ambos perfiles para vídeo entrelazado, como es el caso de 1080i, que no supera el 25 % con respecto a la transmisión simultánea de ambas vistas, lleva a la utilización de tecnologías como *2D + Depth* (sección 2.3.3.5) de manera que los dispositivos 2D puedan mostrar una vista con buena calidad y los compatibles con 3D pueden reconstruir ambas vistas.

Además de los considerados por DVD-3DTV existen múltiples códecs propietarios como el **formato 3D en mosaico *service compatible*** de Sisvel Technology [58] (figura 2.12). Este sistema está diseñado para vídeo generado a 720p y permite almacenar las dos vistas en un único cuadro 1080p. Como se puede observar la vista izquierda permanece inalterada, mientras que la derecha se divide en tres fragmentos (R_1, R_2 y R_3) de

manera que ocupan parte del espacio restante del frame de 1080p tras colocar la primera vista. Una versión posterior (formato 3DZ en mosaico) permite añadir información de profundidad en el fragmento que queda vacío en la esquina inferior derecha. Se utiliza un parámetro de señalización del flujo H.264, llamado *cropping rectangle* (rectángulo de recorte) que indica al descodificador qué parte de la trama de datos tiene que ser mostrada en la pantalla, de este modo un receptor HDTV puede mostrar correctamente la imagen de la vista izquierda.

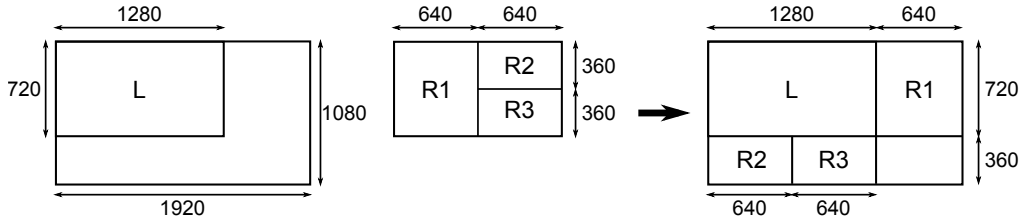


FIGURA 2.12 – Ejemplo de formato 3D en mosaico ***service compatible***

2.3.3.4. Formatos *full 3D*

Son formatos 3D que mantienen la resolución original de captura. La principal desventaja es que necesitan cambios en la infraestructura de transporte de la señal. Los sistemas de 3DTV de resolución *full HD* requieren dos flujos completos de contenido 1080p para ser capaces de mostrar una imagen HD 1080p completa a cada ojo de manera secuencial.

El formato ***Frame Sequential*** (*page flipping* o secuencia de cuadros) consiste en una secuencia alternada de cuadros que contienen la imagen correspondiente a cada ojo. Es decir, si un cuadro contiene la imagen del ojo izquierdo, el siguiente contiene la imagen del derecho (figura 2.13). La secuencia de vídeo progresivo generada duplica la tasa de transmisión de *frames* original de cada vista, duplicando el ancho de banda necesario.

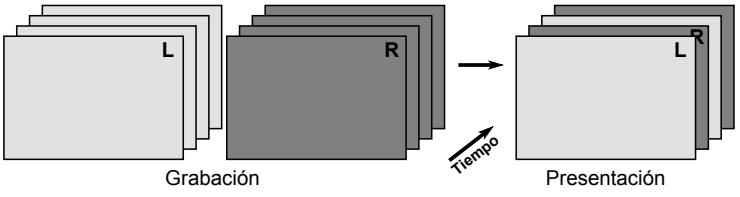


FIGURA 2.13 – **Formato *frame sequential*:** la imagen de cada ojo se presenta alternadamente en el receptor.

Este formato aparece en la especificación 3D para Blu-Ray y es muy popular, ya que es apropiado para los sistemas de televisión compatibles con 3D basados en obturación activa, capaces de mostrar de forma alterna las imágenes del ojo izquierdo y derecho secuencialmente. Así pues, se presenta directamente sobre la pantalla sin necesidad de procesado adicional en el caso de las pantallas activas.

Frame Packing no es un formato propiamente dicho, sino una configuración de empaquetado de los cuadros. El término se usa para referirse al contenido 3D cuyas subimágenes para el ojo izquierdo y derecho se combinan en un único cuadro con estructura *side-by-side* o *top-and-bottom* sin reducir la resolución original de cada imagen a la mitad. Por este motivo, *Frame Packing* preserva la fidelidad y calidad del vídeo ya que cada subimagen permanece con la resolución HD completa, obteniendo como resultado un cuadro 3D cuyo contenido ocupa el doble de tamaño del contenido HD bidimensional.

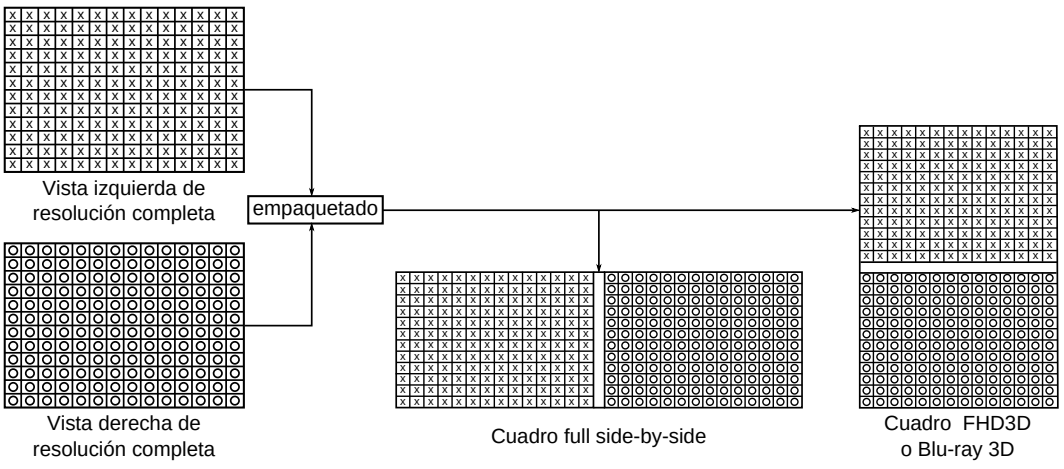


FIGURA 2.14 – **Formato *frame packing*:** las imágenes de ambos ojos se empaquetan en único frame 3D manteniendo la resolución HD original.

Se denomina ***Full High Definition* 3D (FHD3D)** al formato de empaquetamiento de cuadros *top-and-bottom* 1080p. Según la especificación HDMI 1.4 [50], la estructura *top-and-bottom* 3D es el método preferible y obligatorio para el empaquetado de cuadros como se describe en la especificación de FHD3D, también conocida como *Blu-Ray 3D*. FHD3D es el único formato 3D sin pérdidas que proporciona verdadera alta definición. Aunque se puede usar el empaquetamiento *side-by-side*, donde cada subimagen mantiene la resolución completa 1080p o 720p, tal formato no es obligatorio en HDMI 1.4 de manera que los fabricantes no tienen por qué soportarlo (figura 2.14).

La mayoría de los discos *Blu-ray* de películas en 3D de 1080p y juegos 3D de PS3 de 720p utilizan esta técnica. Para representar el contenido *frame packing*, el dispositivo de televisión compatible con HDMI 1.4 debe ser capaz de diferenciar una señal en formato *frame packing* y separar los cuadros individuales para cada ojo sin intervención del usuario, mostrando el contenido en pantalla en formato *frame sequential*.

2.3.3.5. Formatos para la conversión 2D a 3D

El formato ***2D + Depth*** (o 2D+Z) transmite una de las vistas con su resolución original, junto con el mapa de profundidad. Ambas imágenes se pueden empaquetar en un cuadro HD utilizando cualquiera de los formatos descritos anteriormente. La figura 2.15 representa un cuadro que se transmite con este formato, donde las vistas se presenta en formato FHD3D.

FIGURA 2.15 – ***2D+Depth***[4]

El estándar ISO MPEG-C [59, 60] permite que el mapa de profundidad sea tratado como un flujo de video auxiliar, compatible con los estándares 2D. El formato *2D+depth* no depende de la tecnología de captura o presentación del contenido siendo compatible con la mayoría de algoritmos de conversión 2D a 3D. La eficiencia de compresión puede alcanzar una ganancia del 80 % frente a la transmisión simultánea de ambas vistas, resultando atractivo para entornos con restricciones de ancho de banda. Otra característica de flexibilidad es el control del usuario sobre el rango de profundidad global.

Por último, el formato ***2D + Delta*** facilita la codificación de secuencias multivista de modo eficiente, explotando la redundancia presente entre vistas. Surgió como un método propietario para la codificación de vídeo estereoscópico que utiliza una de las imágenes como base 2D. La diferencia optimizada o disparidad (*delta*) entre esa vista y la segunda se inyecta en el flujo de vídeo como flujo auxiliar. La información *delta* puede ser disparidad espacial, predicción temporal o compensación de movimiento bidireccional u optimizada. Pertenece a la extensíon MVC *Stereo High Profile* definida en H.264 [52] y mejora la transmisión simultánea de las dos vistas en torno al 30 %-50 %, por lo que requiere un ancho de banda aproximado 1,5 veces superior al de una secuencia 2D.

[4]Fuente: http://en.wikipedia.org/wiki/File:2D_plus_depth.png

2.3.3.6. Formatos avanzados y *Free Viewpoint Television*

Gracias a la tecnología MVC, se han diseñado esquemas avanzados de codificación de contenido multivista como *Multiview video plus depth* (MVD), un paso más allá del formato *2D+depth*, que añade los mapas de profundidad a la codificación MVC. Aprovecha la redundancia existente mediante codificación predictiva y eficiente entre vistas tomadas por cámaras vecinas correladas ahorrando ancho de banda. El receptor puede reconstruir más vistas intermedias con ayuda de la información de profundidad.

Este tipo de formatos permiten el desarrollo de tecnologías como la ***Free Viewpoint Television*** **(FTV)** o **televisión de libre punto de vista**, que otorga al telespectador la capacidad de elegir libremente el punto de vista desde el que se muestra la escena. Por otra parte, la codificación multivista junto con la reconstrucción de vistas intermedias, aplicadas a los sistemas autoestereoscópicos, aumentan el área de movimiento del observador e incrementan el número de usuarios que pueden visualizar el contenido a la vez. El trabajo realizado dentro del campo FTV tiene el objetivo de dar a la audiencia de televisión la oportunidad de experimentar más allá del 3D convirtiendo al usuario en un agente activo capaz de interactuar con el contenido de televisión.

2.3.4 Visualización de contenido estereoscópico

Tras la recepción y descodificación, los dispositivos de visualización estereoscópica deben presentar en pantalla el contenido 3D de manera que el usuario vea cada vista con el ojo correspondiente. Los rasgos que caracterizan estos dispositivos son los siguientes:

1. El método aplicado para separar la vista del ojo izquierdo y derecho.
2. La capacidad de visualizar contenido multivista.
3. El número de espectadores que pueden ver el contenido simultáneamente.

Actualmente se utilizan dos estrategias de filtrado y separación de imágenes para cada ojo, distinguiendo entre dispositivos de visión **estereoscópica** y **autoestereoscópica** [61]. En el primer caso, el espectador lleva un dispositivo óptico para dirigir las imágenes izquierda y derecha al ojo apropiado (**visualización asistida**) mientras que en el segundo caso, la técnica para separar ambas vistas está integrada en el dispositivo de visualización (**visualización libre**) siendo innecesario el uso de gafas u otro complemento.

Los dispositivos con visualización asistida están ampliamente extendidos y pueden mostrar ambas imágenes simultáneamente en una o dos pantallas (*time parallel*) o secuencialmente (*time sequential*). Las tecnologías comunes de presentación de contenido estereoscópico se pueden clasificar de la siguiente forma [62, 63]:

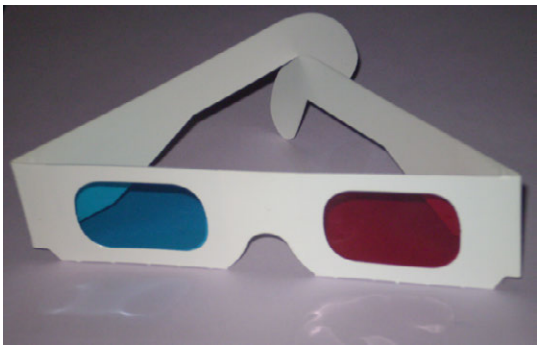
- **Visualización asistida:** uso de filtros o lentes. Sistemas basados en:
  - **Multiplexación de color:** uso de anaglifos, con filtros de color pasivos.
  - **Multiplexación de polarización:** con filtros de polarización pasivos.
  - **Obturación activa:** con gafas de obturación activa.
  - **Visor montado sobre la cabeza (HMD):** cascos de visualización, con una pantalla independiente, separada para cada ojo y lentes para relajar el foco.
- **Visualización libre:** sistemas autoestereoscópicos. Pueden basarse en:
  - **Dirección multiplexada:** barreras de paralaje o lentes lenticulares.
  - Proyección de **campos de luz**.
  - **Holografía:** sistemas holográficos con modulación en amplitud, fase, o basados en reflexión mediante espejos.
  - **Visores volumétricos:** con proyección sobre el espacio físico tridimensional.

Estas tecnologías limitan la posición del espectador y, en ocasiones, son costosas o requieren que el usuario utilice dispositivos incómodos o antiestéticos. A continuación, describiremos las tecnologías aplicables a los contenidos de televisión pensados para el consumo en el hogar, excluyendo cascos o dispositivos holográficos y volumétricos.

2.3.4.1. Multiplexación de color

La multiplexación de color es una de las técnicas más primitivas para la separación de imágenes. Se denomina **anaglifo 3D** al efecto estereoscópico 3D que se logra al codificar las imágenes de cada ojo utilizando diferentes filtros de color, por lo general, opuestos cromáticamente (o complementarios). Típicamente se usa rojo y cian, aunque pueden ser rojo y azul, amarillo y violeta o rojo y verde, por ejemplo. Las imágenes para anaglifo 3D se forman superponiendo ambas imágenes filtradas, como en la figura 2.16(a). Cuando se visualiza a través de las gafas de anaglifo (figura 2.16(b)) codificadas por color, cada imágen alcanza un ojo, revelando una imagen estereoscópica integrada. El córtex visual del cerebro lo fusiona interpretando la diferencia de color creando la percepción de una escena o composición tridimensional.

(a) **Imagen multiplexada por color**[5]



(b) **Anaglifos**[6]

FIGURA 2.16 – Visualización basada en **multiplexación por color**

La principal ventaja que ofrece es su bajo coste, ya que este tipo de contenido se puede reproducir en cualquier monitor o televisor siendo necesario únicamente el uso de las gafas de anaglifo para filtrar correctamente las imágenes. Sin embargo, presenta como desventaja la llamada **falta de extinción** que consiste en un efecto de difusión de un ojo a otro, es decir, el proceso de generación de las imágenes coloreadas no es perfecto y tras el filtrado, hay elementos de una imagen que son vistas por el ojo contrario. Además, este tipo de filtrado produce una alteración de los colores, pédida de luminosidad y cansancio visual tras un uso prolongado.

Como se ha comentado, existen múltiples versiones de esta técnica utilizando diferentes combinaciones de filtros de color debido a la constante búsqueda del compromiso entre el efecto estereoscópico y la precisión de los colores. La tecnología **Infitec** ofrece un aumento significativo de la calidad de imagen. Está basada en la multiplexación de longitud de onda teniendo en cuenta la naturaleza del ojo humano, que se caracteriza por su sensibilidad a los colores primarios rojo, verde y azul. En la figura 2.17 los filtros utilizados para cada ojo presentan tres bandas de transmisión estrechas (una por cada color) no solapadas entre ambos ojos, dentro del espectro visible. De esta manera se logra una separación óptima entre los canales y soporte para todos los colores.

FIGURA 2.17 – **Gafas basadas en tecnología Infitec** [64]

[5]Fuente: http://commons.wikimedia.org/wiki/File:Dusk_on_Desert.jpg
[6]Fuente: http://es.wikipedia.org/wiki/Archivo:Anaglyph_glasses.png

2.3.4.2. Multiplexación de la polarización

La visualización basada en multiplexación de la polarización se nutre de los principios físicos de **polarización de la luz**. Sobre una pantalla o monitor, se proyectan las imágenes de cada ojo **polarizadas ortogonalmente**, superponiéndolas de manera sincronizada o mostrándolas secuencialmente. La polarización puede ser lineal o circular. El espectador debe utilizar unas gafas polarizadas de manera que cada filtro deja pasar una sola imagen a cada ojo bloqueando la luz polarizada en el sentido opuesto.

Los sistemas de **polarización lineal** (figura 2.18) utilizan dos filtros de polarización lineal ortogonales entre sí (e.g. horizontal y vertical, o $+45^o$ y -45^o), uno para cada vista. La utilización de estos filtros exige que el observador mantenga la inclinación de su cabeza para evitar el efecto *ghosting* o imagen fantasma por el que parte de la luz correspondiente a un ojo alcanza el contrario (mezcla de canales).

En el caso de la **polarización circular**, se utiliza la polarización circular levógira (o izquierda, en sentido antihorario desde el punto de vista del observador) para la imagen de un ojo y dextrógira (o derecha, en sentido horario) para la otra. A diferencia de la polarización lineal permite al espectador inclinar la cabeza, manteniéndose la separación correcta de ambas imágenes, por lo que es más utilizada.

Como alternativa, la tecnología **RealD** no necesita dos proyectores sincronizados, sino un proyector único que presenta las imágenes del ojo izquierdo y derecho tres veces cada una, a una frecuencia de 144 cuadros por segundo (seis veces la tasa de transmisión normal). El proyector realiza un filtrado electrónico que conmuta entre polarización horaria y antihoraria síncronamente con la emisión de los cuadros para cada ojo.

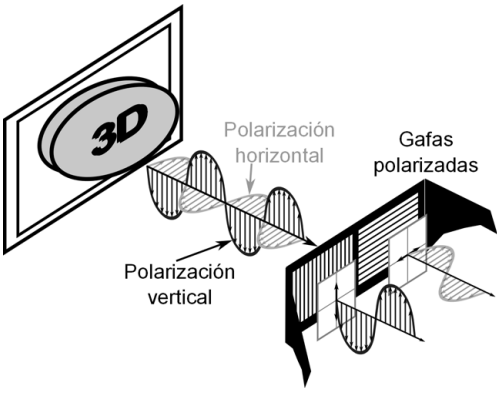


FIGURA 2.18 – Visualización basada en **multiplexación de polarización** lineal

Las gafas de polarización (figura 2.19(a)) son relativamente baratas (de 1 a 10 €) y tienen mayor aceptación que las basadas en anáglifo ya que presentan una apariencia más parecida a las gafas de sol. Además, este tipo de tecnología admite un gran número de espectadores visualizando el contenido simultáneamente, únicamente limitado por la cantidad de gafas disponibles. Como contraposición, los filtros de polarización atenúan la intensidad de la luz, por lo que el dispositivo de visualización debe compensar este efecto aumentando la luminosidad, aumentando los costes de producción.

2.3.4.3. Sistemas de obturación activa

Los sistemas 3D de **obturación activa** (alternancia de cuadros o campos) se basan en la presentación secuencial mediante la activación de la vista izquierda o derecha de forma alternada. Esta multiplexación temporal muestra los cuadros en formato *frame sequential*, aprovechando que el sistema visual humano percibe como una sola imagen estéreo dos imágenes consecutivas percibidas en un intervalo de menos de 50 ms.

Análogamente a RealD, sólo requieren un proyector capaz de trabajar a una tasa al menos dos veces superior a la normal para presentar dos imágenes en el mismo intervalo de tiempo. Como requisito, la transición y refresco de la pantalla del televisor debe ser rápida para no producir solapamiento entre las imágenes de ojos distintos. Las pantallas LCD, de plasma (PDP) y de procesado digital de la luz (DLP) cumplen este requisito.

Para la visualización del contenido, el espectador debe utilizar gafas de obturación activa, que dejan paso a la imagen correspondiente para un ojo bloqueando la visualización para el ojo contrario, alternadamente. Es decir, permiten el paso de la luz para el ojo izquierdo mientras lo impiden para el ojo derecho y viceversa, de forma sincronizada con la presentación de cada imagen en la pantalla.

(a) **Gafas pasivas RealD** [65] (b) **Gafas activas CrystalEyes** [66]

FIGURA 2.19 – Tecnologías de visualización: gafas pasivas de polarización (izquierda) y gafas de obturación activa (derecha)

Las gafas de obturación son habitualmente de cristal líquido o LCS (figura 2.19(b)), que tiene la propiedad de volverse opaco cuando se le aplica una tensión, manteniéndose transparente en cualquier otro caso. Además, una señal de temporización sincronizada con la tasa de refresco de la pantalla controla el sincronismo de las gafas con la pantalla a través de un enlace inalámbrico infrarrojo o de radiofrecuencia (e.g. **bluetooth**). Por este motivo las gafas de obturación son más costosas que las anteriores (de 30 a 50 €).

La frecuencia de refresco debe ser superior a 100 Hz para evitar **parpadeo**, ya que las gafas reducen la frecuencia efectiva de visualización a la mitad. Además, requieren mayor luminosidad porque las lentes dejan pasar la luz durante la mitad de tiempo, atenuándola incluso en el modo transparente. Sin embargo, no se produce atenuación cuando se utiliza la pantalla para visualizar contenidos en 2D sin gafas.

2.3.4.4. Sistemas autoestereoscópicos

Los sistemas de **visualización autoestereoscópica** son aquellos que proporcionan sensación tridimensional sin necesidad de accesorios adicionales como gafas. Las pantallas autoestereoscópicas están diseñadas de manera que proyectan imágenes diferentes cuando se observan desde ángulos distintos. Si el observador está colocado correctamente en el área de visualización, recibe una imagen diferente en cada ojo.

Estos sistemas son adecuados para televisión 3D ya que soportan **visualización libre** mientras que las gafas limitan la libertad de movimiento de los espectadores. En este apartado nos centramos en los dispositivos de dirección multiplexada, ya que los holográficos y volumétricos no se aplican a la televisión 3D.

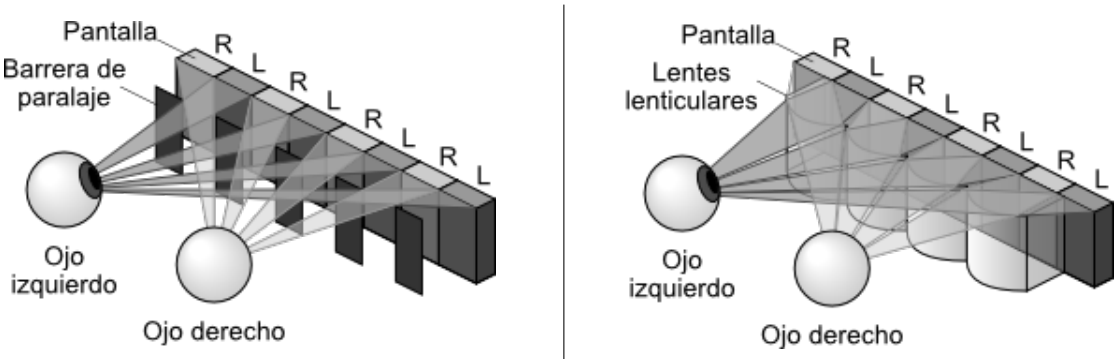
Los dispositivos basados en **dirección multiplexada** aplican técnicas basadas en el efecto óptico producido por un fenómeno físico como la difracción, refracción, reflexión u oclusión para dirigir cada imagen al ojo correspondiente. Existen dos técnicas autoestereoscópicas: las barreras de paralaje y las lentes lenticulares. En ambos tipos, la resolución espacial de la imagen base en 2D se reduce con el propósito de conseguir el efecto estéreo deseado.

- **Dispositivos de barrera de paralaje:** se basa en oclusiones. La barrera de paralaje se sitúa sobre la pantalla y actúa como máscara capaz de direccionar la luz de columnas alternas de píxeles a cada uno de los ojos (figura 2.20(a)). Estos

sistemas permiten conmutar de dinámicamente entre visualización 2D y 3D si la barrera se fabrica con una capa de cristal líquido capaz de volverse completamente transparente, permitiendo funcionar como un monitor 2D convencional.

- **Dispositivos de lentes lenticulares:** basada en la refracción, obtiene el mejor rendimiento. La dirección de la luz se controla mediante lentes cilíndricas (figura 2.20(b)) que disponen de una región central, de entre 10 y 15 grados de amplitud dentro de la cual se observa el efecto correctamente y con zonas adicionales a los lados. Estas zonas adicionales permiten que varios usuarios puedan visualizar las imágenes de forma simultánea si se sitúan correctamente dentro de cada región. También permiten un cierto grado de paralaje horizontal a través del movimiento, es decir la escena cambia según se desplaza horizontalmente la vista del usuario pero con un efecto perceptible de salto entre las vistas.

Los principales problemas que presentan los sistemas autoestereoscópicos explicados en esta sección es la limitación de la posición del espectador y el máximo número de observadores simultáneos, debido a que el ángulo horizontal de observación permitido es bastante limitado. Sin embargo, se puede conseguir un ángulo más amplio utilizando dispositivos de visualización multivista, donde se presenta un número discreto de vistas dentro del área de visualización. También se puede usar tecnologías de seguimiento de posición del usuario para ajustarse a los movimientos de las pupilas o de la cabeza.



(a) **Barrera de paralaje** (b) **Lentes lenticulares**

FIGURA 2.20 – Visualización autoestereoscópica de **dirección multiplexada**[7]

[7]Modificación de la figura obtenida de: http://en.wikipedia.org/wiki/Autostereoscopy

Capítulo 3

Calibración y Rectificación

3.1 Introducción

Como se ha descrito en la presentación del problema (Capítulo 1) uno de los requisitos de partida del sistema de estimación de mapas de profundidad ha sido que el vídeo estereoscópico de entrada se encuentre correctamente calibrado y rectificado. Este requisito está motivado por la restricción de búsqueda a lo largo de una línea horizontal impuesta por la mayoría de los algoritmos de correspondencia estéreo.

En el proyecto ImmersiveTV [3, 4] el socio encargado de la generación del contenido de vídeo estereoscópico era también responsable de que el par de imágenes proporcionadas estuviera correctamente alineado. Para ello, dicho socio realizó una calibración exahustiva de las cámaras estereoscópicas utilizadas para la grabación del contenido, además de prestar especial atención a la calidad del estéreo en las fases de producción y post-producción.

Dada la calidad del contenido proporcionado, solo ha sido necesario comprobar que el vídeo se encontraba correctamente calibrado para la consiguiente aplicación de un algoritmo de correspondencia estéreo. Para ello, hacemos uso de una herramienta denominada *brief_match_test*, disponible en OpenCV [67]. Esta herramienta toma dos imágenes, encuentra puntos característicos en ambas, busca las correspondencias entre ellos y las representa sobre las imágenes originales.

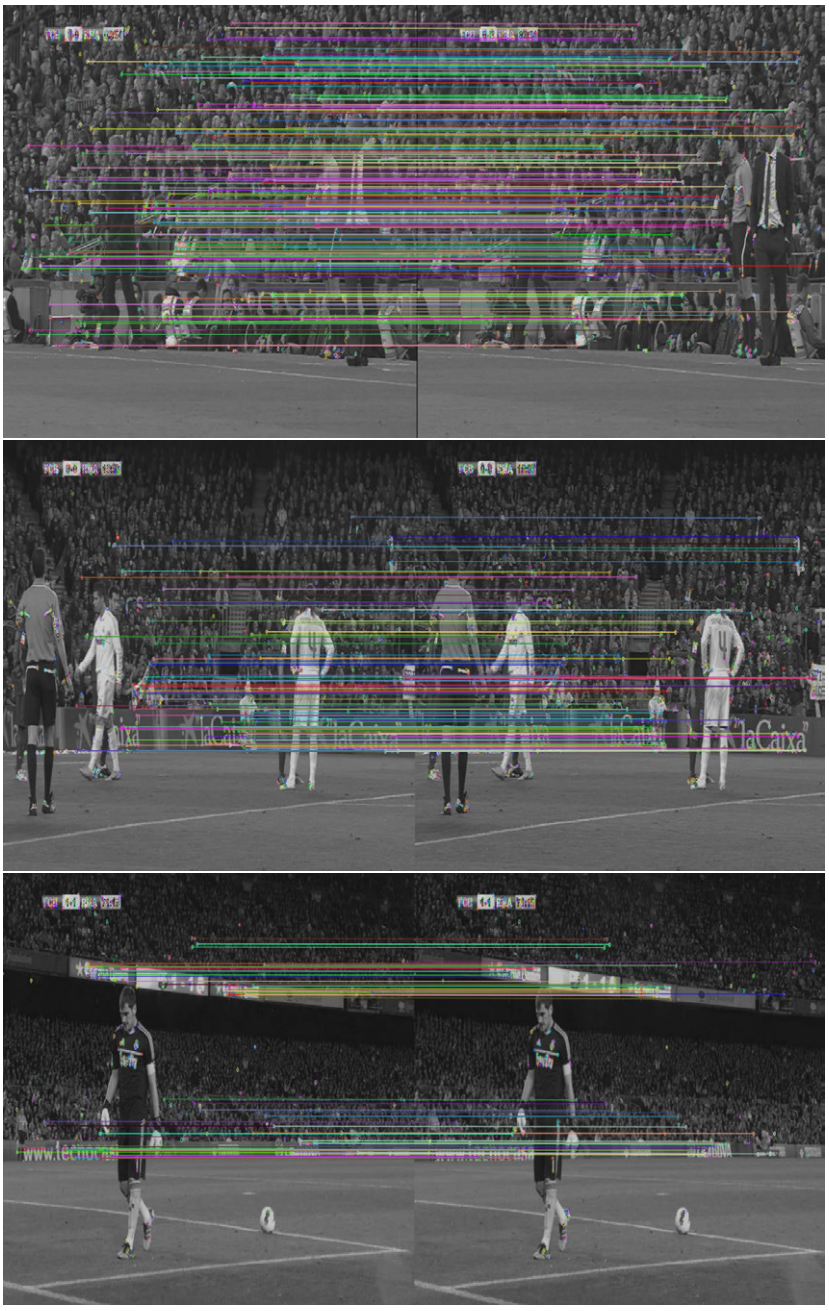


FIGURA 3.1 – **Comprobación de imágenes rectificadas**

La figura 3.1 muestra el resultado de *brief_match_test* para algunas capturas de nuestro contenido. Como se puede observar, las líneas que unen los puntos de correspondencia son horizontales. Esto quiere decir que las correspondencias se pueden encontrar a lo largo de líneas horizontales y por tanto, nuestro contenido está rectificado.

Sin embargo, teniendo en cuenta que en la práctica resulta difícil garantizar unas condiciones tan estrictas para la producción de vídeo estereoscópico y con el fin de poder alcanzar conclusiones sobre la escalabilidad de la solución propuesta, se han analizado

los fundamentos teóricos y algunas de las soluciones de calibración y rectificación más relevantes que puedan ser aplicables al problema.

Teniendo en cuenta lo anterior, en el presente Capítulo se describe los procesos de calibración y la rectificación, explicando en primer lugar el proceso de formación de imagen y los parámetros que definen una cámara, a continuación se extiende la teoría para dos cámaras y se explica la geometría estéreo, además de comentar brevemente las herramientas disponibles y su utilidad para nuestro problema [67, 68].

3.2 Fundamentos teóricos

En este apartado nos centraremos en las características puramente geométricas de la cámara. Para poder comprender el proceso de calibración y rectificación es necesario introducir el modelo de formación de imagen mediante la proyección de los rayos luminosos. Una vez conocido en términos generales el proceso de formación de imagen, podemos pasar a describir el modelo de cámara mediante sus parámetros intrínsecos y extrínsecos. Por último, explicamos el modelo geométrico para un sistema binocular como el que se utiliza para generar el contenido estereoscópico sobre el que se va a aplicar la rectificación.

3.2.1 Introducción a la geometría proyectiva. El modelo *pinhole*

El modelo *pinhole*, también conocido como **cámara oscura** o **estenopeica** es una representación simplificada del sistema óptico de la cámara, que está compuesto por lentes, aperturas y demás elementos diseñados para hacer que los rayos procedentes del mismo punto P en el espacio converjan en un único punto p de la imagen. Este modelo permite describir y entender de manera más fácil la geometría de los rayos luminosos que se proyectan a través de la cámara generando una imagen. El modelo de proyección perspectiva *pinhole*, también conocida como **perspectiva central** fue propuesta inicialmente por Brunelleschi a principios del siglo XV y, a pesar de su simplicidad, ofrece una aproximación bastante aceptable del proceso de formación de imagen [68].

Se basa en la idea de reducir la apertura de la cámara a un único punto, llamado ***pinhole*** o **estenopo** y está formado por una **lámina** imaginaria que bloquea todos los

rayos luminosos procedentes de la escena salvo aquellos que son capaces de atravesar una **apertura** infinitamente pequeña situado **en su centro**. De esta manera solo llega a la cámara un único rayo procedente de cada uno de los puntos de la escena real (correspondencia uno a uno entre los puntos de la escena y los de la imagen). Estos rayos intersecan con un plano denominado **plano de proyección, proyectivo** o **plano de imagen** que está situado detrás del plano de la lámina que contiene el orificio, demominado **plano *pinhole***, como se puede observar en la figura 3.2. Todo el espacio comprendido entre el plano *pinhole* y el de proyección se encuentra aislado o protegido de la luz exterior de la misma manera que está construida una cámara oscura, la cual basa su funcionamiento en este modelo.

Supongamos, como ya se ha comentado, que la luz proviene de la escena o de un objeto distante pero solo un único rayo procedente de cada punto físico entra en la cámara. En una cámara *pinhole* física, toda proyección de un punto sobre el plano de imagen se encuentra en el **foco** y el tamaño de dicha imagen relativa al objeto distante viene dada por la **distancia focal** de la cámara. Por ello, en nuestro modelo idealizado de la cámara la distancia entre la apertura estenopeica y el plano de proyección es la distancia focal (f). Todo esto se ilustra en la figura 3.2, donde f es la distancia focal de la cámara, Z es la distancia desde la cámara al objeto, H es la longitud del objeto y h es la imagen del objeto en el plano de proyección.

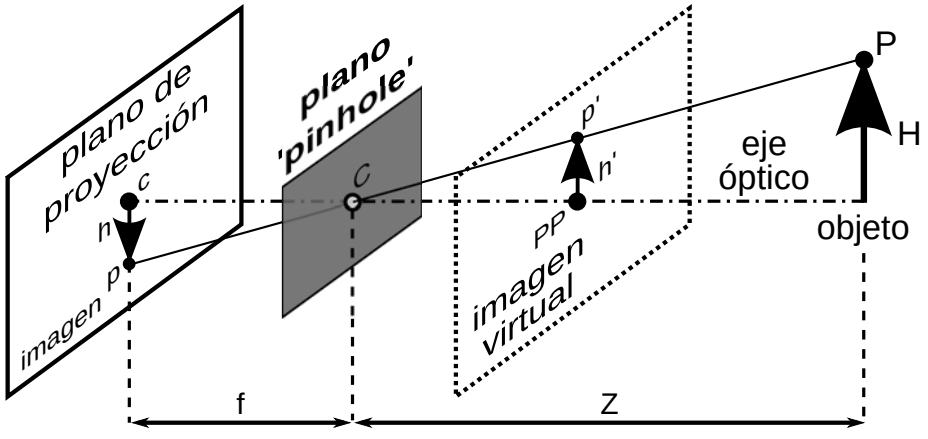


FIGURA 3.2 – **Modelo de proyección:** funcionamiento de una cámara estenopeica. Los rayos de luz provenientes de un objeto atraviesan una pequeña apertura para formar la imagen.

En la figura 3.2, podemos ver por semejanza de triángulos:

$$-\frac{h}{f} = \frac{H}{Z} \implies -h = f\frac{H}{Z} \tag{3.1}$$

donde h es una altura negativa, es decir, la imagen aparece invertida debido a la trayectoria rectilínea de la luz.

Consideremos ahora un sistema de coordenadas asociado a la cámara, cuyo origen (C) se encuentra en el *pinhole*. Se denomina **eje óptico** a la línea perpendicular al plano de imagen que pasa por el origen, y el punto (c) de intersección con el plano de imagen se denomina **centro de imagen**. Este punto se puede utilizar como origen de coordenadas en el plano de proyección y juega un papel fundamental en el proceso de calibración de la cámara.

Sea P un punto de la escena de coordenadas $(\mathrm{X,Y,Z})^{\mathrm{T}}$ y p su imagen con coordenadas $(\mathrm{x,y,z})^{\mathrm{T}}$ (usaremos letras mayúsculas para denotar puntos en el espacio y minúsculas para sus proyecciones en la imagen). Como p se encuentra en el plano de imagen, se tiene que z=f. Puesto que los tres puntos P, C y p son colíneales, se tiene que $\overrightarrow{Cp} = \lambda\overrightarrow{CP}$ para algún λ (puede ser negativo), luego

$$\begin{cases} x = \lambda X \\ y = \lambda Y \\ z = f = \lambda Z \end{cases} \iff \lambda = \frac{x}{X} = \frac{y}{Y} = \frac{f}{Z} \tag{3.2}$$

y por tanto

$$\begin{cases} x = f\frac{X}{Z} \\ y = f\frac{Y}{Z} \end{cases} \tag{3.3}$$

estas relaciones no son líneales debido a la aparición del factor 1/Z.

La proyección perspectiva crea imágenes invertidas, y suele ser conveniente reorganizar nuestro modelo de manera que sea equivalente y los cálculos se simplifiquen. Consideramos una **imagen virtual** colocando el plano de imagen delante del *pinhole* en la posición simétrica (distancia f). La principal diferencia es que la imagen ahora no aparece invertida pero es equivalente a la original. El punto C en la apertura se reinterpreta como **centro de proyección**. En este contexto, todo rayo une un punto del objeto distante con el centro de proyección. El punto de intersección del plano de imagen

virtual con el eje óptico se denomina **punto principal** (PP). En este nuevo plano de imagen frontal, la imagen del objeto es exactamente del mismo tamaño que en el plano de proyección y se genera mediante la intersección de los rayos con el plano de imagen virtual. Esta modificación hace que la relación de semejanza de los triángulos de la figura 3.2 sea

$$\frac{h'}{f} = \frac{H}{Z} \implies h' = f\frac{H}{Z} \tag{3.4}$$

donde el signo negativo desaparece porque la imagen ya no está invertida.

Las imágenes obtenidas mediante este modelo son muy nítidas y sin distorsión. Sin embargo, dado que el tiempo de exposición es inversamente proporcional al cuadrado del diámetro de la apertura y que ésta es a su vez proporcional a la cantidad de luz que entra en la cámara, el modelo *pinhole* no se puede aplicar de forma estricta en la realidad porque no recoge suficiente luz en una exposición de corta duración. Por este motivo, al igual que nuestros ojos, las cámaras reales están equipadas con lentes que permiten captar más luz de la que habría disponible en un único punto y los sistemas ópticos pueden ser ajustados para trabajar con un rango muy amplio de condiciones de iluminación y tiempos de exposición. La principal desventaja derivada de la utilización de lentes es que introducen **distorsión**, complicando más aún todo procedimiento de corrección.

Otro aspecto a tener en cuenta son los efectos de la proyección perspectiva sobre el tamaño aparente de los objetos en relación a su distancia. Los objetos lejanos parecen más pequeños que los cercanos. Además, no se preserva la distancia entre puntos ni los ángulos entre rectas, y las imágenes de líneas paralelas intersecan en el horizonte.

3.2.2 Parámetros de la cámara

Las imágenes digitales son espacialmente discretas y están divididas en elementos de imagen rectangulares, que denominamos **píxeles**. En la práctica, el sistema de coordenadas global se relaciona con el de la cámara a través de un conjunto de parámetros físicos como la distancia focal de la lente, el tamaño de los píxeles, la posición del centro de la imagen o la posición y orientación de la cámara. Se llama **calibración** geométrica de la cámara al proceso de estimación de los parámetros antes mencionados.

Para reconstruir la estructura 3D de una escena se necesita relacionar las coordenadas de los puntos del espacio 3D con las de sus correspondientes puntos 2D en la imagen. La calibración es fundamental para relacionar medidas de la cámara con medidas en el mundo real. Además, en las escenas existen espacios físicos con unidades físicas, por lo que la relación entre las unidades naturales de la cámara (píxeles) y las unidades del mundo físico (e.g. metros) es una componente crítica en cualquier intento de reconstruir una escena tridimensional. También, mediante el uso de la calibración de cámaras, se puede corregir (matemáticamente) las principales desviaciones que introduce el uso de lentes respecto del modelo pinhole simple. Por lo tanto, el proceso de calibración de cámara nos devuelve tanto un modelo de la geometría de la cámara como un modelo de distorsión de la lente.

Para poder entender mejor los siguientes párrafos es necesario introducir el concepto de **coordenadas homogéneas**.

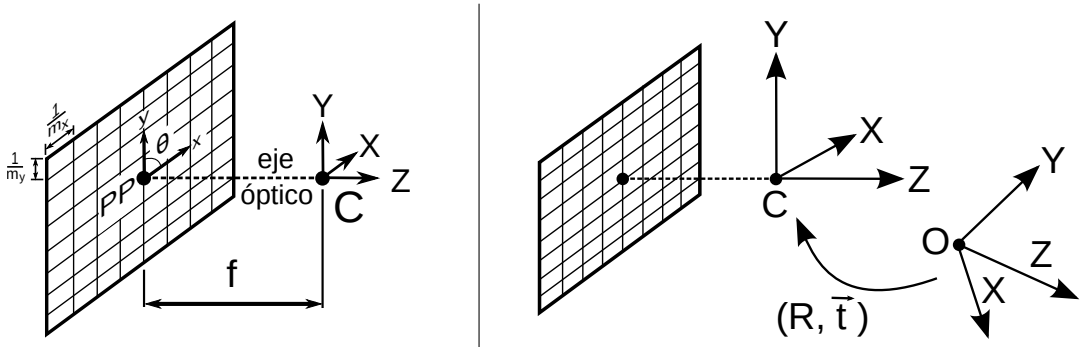
Como sabemos, la relación que transforma el punto P_i en el mundo físico de coordenadas $(X_i, Y_i, Z_i)^T$ con los puntos en el plano de proyección con coordenadas $(x_i, yi)^T$ se denomina **transformación proyectiva**. Para trabajar con estas transformaciones es conveniente utilizar las conocidas como **coordenadas homogéneas**. Las coordenadas homogéneas asociadas a un punto perteneciente a un espacio proyectivo de dimensión n suelen ser representadas por medio de un vector (n+1)-dimensional, con la restricción adicional de que cualesquiera dos puntos cuyos valores sean proporcionales, son equivalentes. En nuestro caso el plano de imagen es el plano proyectivo y tiene dos dimensiones, por ello representaremos los puntos de ese plano como vectores tridimensionales $p = (x, y, z)^T$. Teniendo en cuenta que todos los puntos que tienen valores proporcionales en el espacio proyectivo son equivalentes, podemos recuperar las verdaderas coordenadas del pixel, dividiendo por z.

Dado un punto P, su vector de coordenadas es $(X, Y, Z)^T$ en $\mathbb{R}^3$ y su vector de coordenadas homogéneas, $(X, Y, Z, 1)^T$ en $\mathbb{R}^4$. Las coordenadas homogéneas son una herramienta adecuada para representar varias transformaciones geométricas mediante productos de matrices, como por ejemplo el cambio entre sistemas de coordenadas mediante una matriz de rotación [R] de dimensiones 3x3 y un vector de traslación $\overrightarrow{t}$ en $\mathbb{R}^3$. También proporcionan una representación algebraica del proceso de proyección perspectiva en forma de una matriz [M] de dimensiones 3x4, tal que el vector de coordenadas

$(X, Y, Z, 1)^T$ de un punto P en un sistema fijado de coordenadas del mundo y el vector de coordenadas $(x, y, 1)^T$ de su imagen p en el plano de referencia de la cámara se relacionan mediante la **ecuación de proyección perspectiva**:

$$p = \frac{1}{Z} MP \tag{3.5}$$

A continuación distinguiremos entre **parámetros intrínsecos**, que relacionan el sistema de coordenadas de la cámara con el sistema de coordenadas idealizado utilizado en la sección 3.2.1, de los **parámetros extrínsecos**, que relacionan el sistema de coordenadas de la cámara con un sistema fijado de coordenadas del mundo y especifica su posición y orientación en el espacio.



(a) **Parámetros intrínsecos** (b) **Parámetros extrínsecos**

FIGURA 3.3 – **Parámetros de la cámara**

Los **parámetros intrínsecos** asocian las coordenadas en unidades de píxel de un punto de la imagen con sus correspondientes en el sistema de referencia de la cámara. Los describimos a continuación (figura 3.3(a)).

- **Distancia focal (f):** distancia desde el centro óptico hasta el plano de proyección. Viene dada en unidades de longitud (m o mm).
- **Relación de aspecto** de los píxeles con respecto a las distancias. Aunque en ocasiones se utiliza k y l [68] o s_x y s_y [67], en este trabajo llamaremos m_x y m_y al tamaño efectivo de los píxeles en las direcciones horizontal y vertical respectivamente. Estos coeficientes relacionan las unidades de distancia con unidades de píxel y se miden en píxeles por unidad de distancia (px/m o px/mm). De esta manera un píxel tendrá dimensiones $\frac{1}{m_x}$x$\frac{1}{m_y}$ (m).

Los dos parámetros anteriores suelen relacionarse entre sí, mediante las expresiones 3.6, dando lugar a α_x y α_y, que se corresponden con la **distancia focal medida en píxeles** en las direcciones horizontal y vertical, respectivamente. La distancia focal α_x (α_y) es el producto de la distancia focal física f (m o mm) por el tamaño m_x (m_y) de los elementos de imagen individuales (px/mm o px/m), por lo que α_x (α_y) está en las unidades de píxel requeridas,

$$\alpha_x = fm_x \qquad , \qquad \alpha y = fm_y \tag{3.6}$$

donde el signo es positivo, porque este modelo utiliza la imagen no invertida en el plano de proyección delante del *pinhole*.

Nótese que hemos introducido dos distancias focales diferentes. La razón es que los **píxeles** individuales en un sensor típico son **rectangulares** en vez de cuadrados. Es importante tener encuenta que m_x y m_y no pueden ser medidos directamente mediante un proceso de calibración. Tampoco la distancia focal es medible directamente. solo las combinaciones de la expresión 3.6 pueden ser halladas sin tener que desmontar la cámara y medir sus elementos directamente. Para lentes con zoom, la distancia focal puede variar con el tiempo, trasladando a su vez el centro de imagen cuando el eje óptico de la lente no es exactamente perpendicular al plano de imagen.

- **Punto principal (x_0,y_0):** proyección del centro óptico sobre el plano de proyección. Idealmente se encuentra en el centro de la imagen o punto de la imagen más cercano al centro de proyección. x_0 e y_0 son las coordenadas del punto principal en la imagen y definen la posición de c en unidades de píxel.

Podríamos pensar que el punto principal coincide con el centro de la imagen, pero esto implicaría que el sensor ha sido colocado con precisión exacta durante el proceso de fabricación. De hecho, el centro del chip habitualmente no está en el eje óptico. Por este motivo, se introducen dos nuevos parámetros x_0 e y_0 que modelan un posible **desplazamiento** (del eje óptico) **del centro de coordenadas** en el plano de proyección. El resultado es un modelo relativamente simple en el que un punto P del mundo físico, cuyas coordenadas son $(X, Y, Z)^T$, es proyectado sobre la pantalla en un píxel de posición dada por $(x, y)^T$ de acuerdo a las ecuaciones siguientes:

$$x = \alpha_x \frac{X}{Z} + x_0 \qquad , \qquad y = \alpha_y \frac{Y}{Z} + y_0 \tag{3.7}$$

- **Distorsión geométrica (γ):** coeficiente de distorsión radial debido a la distorsión geométrica que introduce la óptica. Representa la relación de distorsión entre los ejes de coordenadas (idealmente 0, ortogonales). En la figura 3.3(a) se representa θ como el ángulo formado por los dos ejes del sistema de coordenadas en el plano de imagen y se relaciona mediante: $\gamma = -\alpha_x \cot\theta$.

El uso de las coordenadas homogéneas permite organizar los parámetros que definen nuestra cámara en una única matriz 3x3, que llamaremos **matriz de parámetros intrínsecos** o **matriz de la cámara** [K].

$$K = \begin{bmatrix} \alpha_x & \gamma & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

La matriz de parámetros intrínsecos no depende de la escena sino que depende exclusivamente de la cámara y, una vez estimada, se puede reutilizar (siempre y cuando la distancia focal permanezca fija). La determinación de estos parámetros nos permitirá compensar posteriormente la distorsión introducida por las lentes de las cámaras. La proyección de los puntos en el mundo físico sobre la cámara se puede resumir mediante la siguiente ecuación:

$$p = KP, \quad p = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \quad P = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{3.9}$$

Hasta ahora hemos considerado la proyección de un punto referido al sistema de coordenadas de la cámara. Sin embargo, el sistema de referencia de la cámara está localizado dentro de otro sistema de referencia global. Los **parámetros extrínsecos** definen la localización y orientación del sistema de referencia de la cámara con respecto al sistema de referencia global y especifican la transformación entre ambos. Los resumimos en los dos descritos a continuación (figura 3.3(b)).

- **Matriz de rotación [R]:** es el producto de tres rotaciones elementales bidimensionales: $R_x(\psi)$, $R_y(\varphi)$ y $R_z(\theta)$, que se corresponden con la rotación alrededor de los ejes x, y, z respectivamente. La matriz [R] tiene la propiedad de ser ortogonal, es decir, su inversa coincide con su traspuesta: $R^T R = RR^T = I$.

- **Vector de traslación $\overrightarrow{t}$:** representa el desplazamiento u offset desde un sistema de coordenadas a otro sistema cuyo origen ha sido desplazado a otra posición, en otras palabras, el vector de traslación es la transformación de posición desde el origen de coordenadas del primer sistema de coordenadas hasta el origen del segundo sistema de coordenadas: $\overrightarrow{t}$=Origen$_{(\text{global})}$-Origen$_{(\text{cámara})}$.

Combinando ambas podemos obtener la expresión que transforma las coordenadas de un punto de la escena referidas al sistema de coordenadas global en coordenadas referidas al sistema de coordenadas de la cámara:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{Camara} = [R] \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}_{Global} + \overrightarrow{t} \tag{3.10}$$

Denominamos **matriz de parámetros extrínsecos** a la matriz conjunta compuesta por la matriz de rotación y el vector de traslación $[R \mid t]$. Esta matriz se utiliza para describir el movimiento de la cámara alrededor de una escena estática, o viceversa, el movimiento rígido de un objeto delante de una cámara fija. Es decir, $[R \mid t]$ transforma las coordenadas de un punto $(X, Y, Z)^T$ a algún sistema de coordenadas, fijado con respecto a la cámara.

De manera más general, la proyección de los puntos en el mundo físico sobre la cámara se puede resumir mediante la siguiente ecuación:

$$p = \frac{1}{Z} MP \tag{3.11}$$

donde M es la **matriz de proyección perspectiva** dada por:

$$M = K[R \mid t] \tag{3.12}$$

utilizando coordenadas homogéneas resulta:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_x & \gamma & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.13}$$

De esta manera, conociendo las coordenadas de varios puntos 3D en el sistema de referencia global y sus proyecciones sobre el plano de imagen de la cámara se puede calcular los parámetros de la cámara.

3.2.3 Geometría epipolar

La geometría básica de un sistema de imágenes estéreo se denomina **geometría epipolar**. En esencia, esta geometría combina dos modelos *pinhole* (uno para cada cámara) y añade dos puntos de especial interés llamados **epipolos** (ver figura 3.4). A continuación la describiremos y explicaremos la utilidad de dichos puntos.

Cada cámara tiene un centro de proyección asociado, O_L y O_R, y un plano de proyección correspondiente, L y R. El punto P en el mundo físico tiene una proyección sobre cada uno de los planos mencionados y las etiquetamos como p_l y p_r, respectivamente. Los puntos e_l y e_r se denominan **epipolos** de las cámaras. El epipolo e_l (resp. e_r) en el plano de imagen L (resp. R) es la proyección del centro óptico O_R (O_L) de la segunda (primera) cámara observada por la otra cámara.

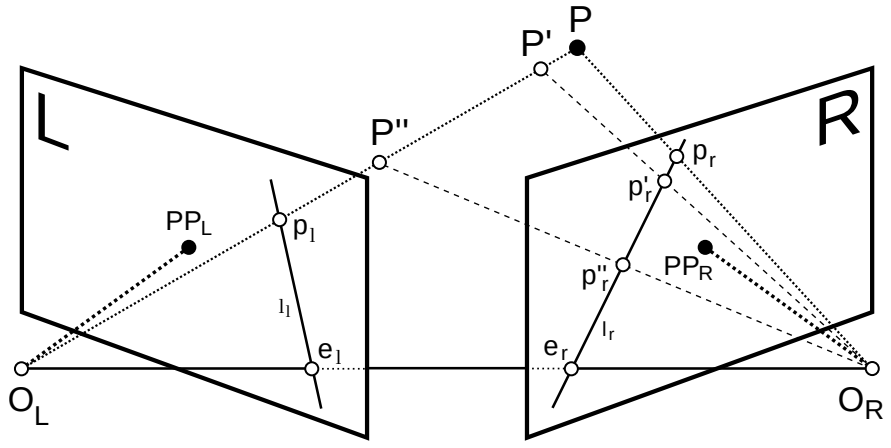


FIGURA 3.4 – **Relación epipolar:** el conjunto de correspondencias posibles para el punto p_l de la vista izquierda está restringido dentro de la línea epipolar asociada l_r en la vista derecha.

El plano en el espacio formado por el punto P y los dos epipolos e_l y e_r (o los dos centros de proyección O_L y O_R) se denomina **plano epipolar**, es decir los cinco puntos P, e_l, e_r, O_L y O_R además de las imágenes p_l y p_r pertenecen a un plano definido por los dos rayos $\overline{O_L P}$ y $\overline{O_R P}$ que intersecan. En particular, el punto p_l se encuentra en la línea l_l donde este plano corta con el plano de proyección L (análogamente, p_r se encuentra en l_r, la intersección del plano con R). Las líneas $\overline{p_l e_l}$ y $\overline{p_r e_r}$ se denominan

líneas epipolares. La línea l_r es la línea epipolar asociada al punto p_l, y pasa por el epipolo e_r, donde la línea de base que une los centros ópticos (O_L y O_R) corta con el plano R (análogamente, l_l es la línea epipolar asociada al punto p_r que pasa por la intersección e_l de la línea de base con el plano L).

Como se ha dicho, si p_l y p_r son imágenes del mismo punto, entonces p_r debe estar en la línea epipolar asociada con p_l. Esta restricción epipolar juega un papel fundamental en visión estéreo y análisis de movimiento.

Para entender la utilidad de los epipolos primero recordamos que, cuando vemos un punto del mundo físico proyectado sobre nuestro plano de imagen izquierdo (o derecho), dicho punto podría estar realmente situado en cualquier posición de la línea definida por el rayo que va desde O_L, a través de p_l (O_R a través de p_r) ya que, con una única cámara, no podemos conocer la distancia hasta el punto que estamos viendo. Más concretamente, tomamos como ejemplo el punto P visto desde la cámara izquierda. Puesto que esa cámara únicamente ve p_l (la proyección de P sobre L), el punto P real podría estar situado en cualquier posición de la línea definida por p_l y O_L. Esta línea obviamente contiene a P, pero también contiene un gran número de puntos (P', P”,...). Lo interesante es preguntarse qué aspecto tiene esa línea proyectada sobre el plano de imagen derecho R; de hecho, será la línea epipolar definida por p_r y e_r. En otras palabras, la imagen de todas las posibles posiciones de un punto visto desde una cámara es la línea que atraviesa el punto correspondiente y el epipolo en la otra cámara.

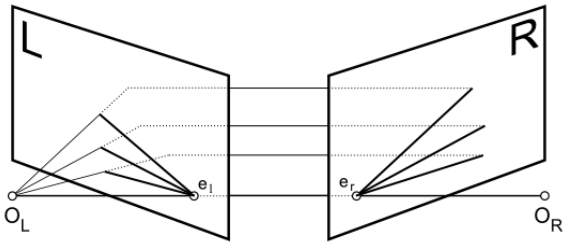


FIGURA 3.5 – **Planos epipolares:** las rectas sobre las que se encuentran los puntos y sus correspondencias forman un haz de planos cuya base es la recta que une los epipolos (línea de base).

A partir de ahora consideraremos que en nuestro sistema las cámaras están calibradas interna y externamente. La parte más complicada de construir un sistema artificial de visión estéreo es encontrar métodos efectivos para establecer correspondencias entre las dos imágenes, esto es, decidir qué puntos en la segunda imagen corresponden con

puntos de la primera. La restricción epipolar limita enormemente la búsqueda de estas correspondencias. De hecho, como asumimos que el *rig* está calibrado, las coordenadas del punto p_l determinan completamente el rayo que une O_L y p_l, y por tanto el plano epipolar asociado $O_L O_R p_l$ y la línea epipolar l_r. La búsqueda de correspondencias puede restringirse a esta línea en vez de la imagen completa.

A modo de resumen recogemos algunos aspectos de la geometría epipolar estéreo que nos resultan interesantes y que tendremos en cuenta más adelante.

- Cada punto 3D visto desde las cámaras está contenido en un plano epipolar que interseca con cada imagen en una línea epipolar.
- Dado un punto característico sobre una imagen, su vista correspondiente en la otra imagen se debe encontrar a lo largo de la línea epipolar respectiva. Esto se denomina **restricción epipolar**.
- La restricción epipolar implica que la posible búsqueda bidimensional para encontrar correspondencias en dos imágenes se convierte en una **búsqueda unidimensional** a lo largo de líneas epipolares, una vez conocida la geometría epipolar del *rig* estéreo. Esto no solo supone un ahorro en el coste computacional, sino que también nos permite descartar muchos puntos que en otros casos dan lugar a correspondencias espúreas.
- Se preserva el **orden**. Si dos puntos A y B son visibles en ambas imágenes y aparecen horizontalmente en ese orden en una imagen, entonces también aparecen horizontalmente en el mismo orden dentro de la otra imagen. Hay que tener en cuenta que debido a oclusiones y áreas superpuestas entre vistas, es posible que ambas cámaras no vean los mismos puntos. Sin embargo, el orden se mantiene. Si los puntos A, B y C están dispuestos de izquierda a derecha en la imagen izquierda y B no se ve en la derecha debido a oclusiones, entonces la cámara derecha verá los puntos A y C de izquierda a derecha.

Como se explica a continuación, se presta conveniente caracterizar la restricción epipolar en términos relativos a matrices bidimensionales 3x3, conocidas como esencial y fundamental.

3.2.4 Matriz Esencial

La matriz esencial $[E]$ contiene información sobre la traslación y rotación que relaciona las dos cámaras en el espacio físico (ver figura 3.6). Esta matriz es puramente geométrica y no recoge información relativa a los sensores. **Relaciona la posición, en coordenadas físicas**, del punto P visto por la cámara izquierda (denotado como P_l) con la posición del mismo punto visto desde la cámara derecha (definido como P_r). Como aclaración, no hay que confundir P_l con p_l. Aunque se refieren al mismo punto contenido en el plano de proyección de la cámara izquierda, el primero representa la intersección con el rayo que une O_L y P en coordenadas espaciales, mientras que el segundo representa las coordenadas del píxel resultante dentro de la imagen.

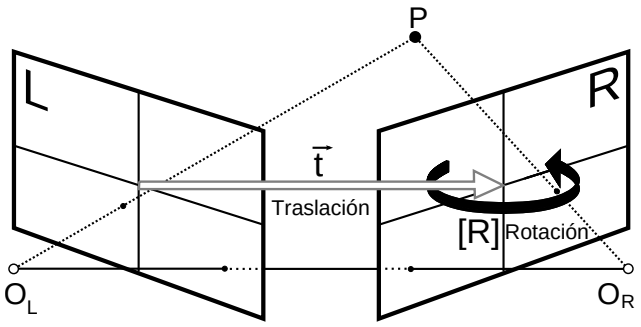


FIGURA 3.6 – **Matriz Esencial:** la geometría esencial de las imágenes estéreo es capturada por la matriz esencial [E], que contiene toda la información acerca de la traslación [T] y la rotación [R], que describe la posición de la segunda cámara con respecto a las coordenadas globales de la primera cámara.

En esta sección los parámetros intrínsecos de las cámaras se suponen conocidos. Dado un punto P, queremos obtener la relación entre las posiciones observadas p_l y p_r de las dos imágenes. Esta relación define la matriz esencial. Partimos de la relación entre P_l y P_r, posiciones físicas de P visto en el sistema de coordenadas de cada cámara y que pueden ser relacionados mediante la geometría epipolar, como ya hemos visto.

Tomando como sistema de coordenadas de referencia el centrado en O_L de la cámara izquierda, la posición del punto observado es P_l y el origen de coordenadas de la otra cámara está localizado en T (posición del afijo de $\overrightarrow{t}$ en el espacio). El punto P visto por la cámara derecha es P_r en ese sistema de coordenadas, donde $P_r = [R](P_l - T)$. De acuerdo con la restricción epipolar, los tres vectores $\overrightarrow{O_L p_l}$, $\overrightarrow{O_R p_r}$ y $\overrightarrow{O_L O_R}$ deben ser coplanares (figura 3.4). La ecuación de los puntos x de un plano cuyo vector normal es

$\overrightarrow{n}$ y que pasa por un punto a es la siguiente:

$$(x - a) \cdot \overrightarrow{n} = 0 \tag{3.14}$$

Puesto que el plano epipolar contiene los vectores P_l y T, podemos sustituir $\overrightarrow{n} = T \times P_l$ en la ecuación 3.14. De manera que la ecuación de todos los puntos P_l pertenecientes a un plano que contiene a T resulta:

$$(P_l - T)^T (T \times P_l) = 0 \tag{3.15}$$

Como el objetivo es relacionar p_l y p_r a partir de la relación de P_l y P_r, introducimos P_r en la ecuación mediante la igualdad mencionada $P_r = [R](P_l - T)$ que puede ser reescrita como $(P_l - T) = [R]^{-1} P_r$, y, utilizando la propiedad de ortogonalidad de $[R]$, sustituyendo $[R]^T = [R]^{-1}$ resulta:

$$([R]^T P_r)^T (T \times P_l) = 0 \tag{3.16}$$

Siempre es posible reescribir un producto vectorial como una multiplicación de matrices. Por lo tanto, definimos la matriz S tal que:

$$T \times P_l = SP_l \implies S = \begin{bmatrix} 0 & -T_z & T_y \\ T_z & 0 & -T_x \\ -T_y & T_x & 0 \end{bmatrix} \tag{3.17}$$

Esto conduce al primer resultado. Sustituyendo en el segundo producto:

$$(P_r)^T [R] S P_l = 0 \tag{3.18}$$

Este producto $[R]S$ define la matriz esencial [E], compactando la ecuación:

$$(P_r)^T [E] P_l = 0 \tag{3.19}$$

Si se sustituye utilizando las ecuaciones de proyección $p_l = f_l \frac{P_l}{Z_l}$ y $p_r = f_r \frac{P_r}{Z_r}$ y se divide todo por $\frac{Z_l Z_r}{f_l f_r}$, se obtiene el resultado final:

$$(p_r)^T [E] p_l = 0 \tag{3.20}$$

La matriz esencial [E] fue introducida por [69] y es una matriz 3x3 de rango 2, de manera que la ecuación 3.20 se puede interpretar como la **ecuación de una recta**. En concreto, $l_r = [E]p_l$ es la línea epipolar asociada al punto p_l en la segunda imagen, y de hecho la ecuación 3.20 se puede reescribir como $p_r^T \cdot l_r = 0$, indicando que p_r pertenece a l_r.

Los nueve coeficientes de $[E]$ se pueden parametrizar con tres grados de libertad de la matriz de rotación $[R]$ y dos grados de libertad que definen la dirección del vector de traslación $\overrightarrow{t}$. Además tiene dos restricciones adicionales: en primer lugar su determinante es cero (matriz 3x3 de rango 2) y en segundo lugar, los dos autovalores distintos de 0 (que caracterizan a las matrices esenciales según [70]) son iguales debido a que la matriz S es antisimétrica y $[R]$ la matriz de rotación. En total hay siete restricciones.

3.2.5 Matriz Fundamental

La matriz esencial contiene toda la información sobre la geometría que relaciona a las dos cámaras, sin embargo no proporciona información acerca de las mismas. Puesto que en la práctica trabajamos con coordenadas de píxel, para encontrar una relación entre un píxel de una imagen y la línea epipolar asociada en la otra imagen, es necesario introducir información intrínseca sobre las cámaras. La matriz fundamental $[F]$ añade información sobre los parámetros intrínsecos de ambas cámaras a la información de la matriz esencial. Esta matriz **relaciona la posición, en unidades de píxel**, de un punto del plano de imagen de la cámara izquierda (denotado como p_l) con la posición del mismo punto visto en la imagen derecha (definido como p_r).

Hasta ahora los parámetros de las cámaras se suponían conocidos y se ha trabajado en coordenadas normalizadas de imagen. Para esta sección debemos comenzar reescribiendo la ecuación 3.20 para considerar los efectos derivados de los parámetros intrínsecos de la cámara:

$$(\hat{p_r})^T[E]\hat{p_l} = 0 \tag{3.21}$$

donde la relación de p_l y p_r con $\hat{p_l}$ y $\hat{p_r}$ viene dada por $p_l = [K]_l\hat{p_l}$ y $p_r = [K]_r\hat{p_r}$, respectivamente. $[K]_l$ y $[K]_r$ son las matrices de parámetros intrínsecos de ambas cámaras. Sustituyendo $\hat{p_l} = [K]_l^{-1}p_l$ y $\hat{p_r} = [K]_r^{-1}p_r$:

$$p_r^T([K]_r^{-1})^T[E][K]_l^{-1}p_l = 0 \tag{3.22}$$

Definiendo la matriz fundamental como:

$$[F] = ([K]_r^{-1})^T [E][K]_l^{-1} \tag{3.23}$$

de manera que la expresión se simplifica, resultando:

$$p_r^T [F] p_l = 0 \tag{3.24}$$

Nótese la ecuación 3.23 que relaciona la matriz fundamental con la esencial. Si tenemos imágenes rectificadas, normalizando los puntos dividiendo por las distancias focales, la matriz intrínseca $[K]$ se convierte en la matriz identidad con $[F] = [E]$. La matriz fundamental, al igual que la esencial, tiene rango 2. Esta matriz tiene siete parámetros, dos para cada epipolo y tres para la homografía que relaciona los dos planos de imagen. El autovector de $[F]$ (resp. $[F]^T$) correspondiente a su autovalor 0 es la posición del epipolo e_l (e_r). Análogamente, $l_l = [F]p_r$ (resp. $l_r = [F]^T p_l$) representa la línea epipolar correspondiente al punto p_r (resp. p_l) en la primera (resp. segunda) imagen.

Las matrices $[E]$ y $[F]$ se pueden calcular directamente a partir de los parámetros intrínsecos y extrínsecos. Las ecuaciones 3.20 y 3.24 también imponen restricciones sobre los elementos de las matrices, independientemente de la posición 3D de los puntos observados. En particular, esto implica que $[E]$ y $[F]$ pueden ser calculados a partir de un **número suficiente de correspondencias** entre las imágenes sin utilizar un patrón de calibración.

3.2.6 Rectificación

Los cálculos asociados a los algoritmos estéreo se simplifican considerablemente cuando las imágenes de interés han sido rectificadas, es decir, sustituidas por dos imágenes equivalentes con un **plano de imagen común paralelo a la línea de base** que une los dos centros ópticos (ver figura 3.7). Desafortunadamente, una configuración perfectamente alíneada es muy poco común en un sistema estéreo real, ya que las dos cámaras casi nunca tienen planos de imagen exactamente coplanares, con las filas alineadas.

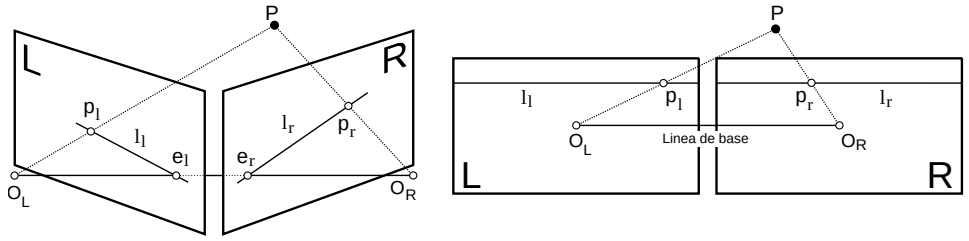


FIGURA 3.7 – **Rectificación**

El proceso de rectificación consiste en **reproyectar los planos de imagen** de las dos cámaras de manera que se encuentren en el mismo plano, con las filas perfectamente alineadas en una configuración fronto-paralela. Queremos que las filas entre las dos cámaras estén alíneadas tras la rectificación para que la correspondencia estéreo (encontrar el mismo punto en dos vistas diferentes) sea más fiable y manejable computacionalmente. Con una elección apropiada del sistema de coordenadas, las líneas epipolares rectificadas sirven como líneas de búsqueda en las nuevas imágenes, y son también paralelas a la línea de base. Este hecho facilita la búsqueda de correspondencias.

Hay dos grados de libertad involucrados en la elección del nuevo plano de imagen:

- La **distancia entre este plano y la línea de base**. Aunque es esencialmente irrelevante porque modificarla solo cambia la escala de las imágenes rectificadas, que se puede compensar mediante el escalado inverso de los ejes de coordenadas de imagen.
- La **dirección de la normal** del plano rectificado ha de ser la del plano perpendicular a la línea de base.

El resultado de alinear las filas horizontalmente dentro de un **plano de imagen común** que contiene ambas imágenes es que los ejes ópticos (o rayos principales) de las cámaras son paralelos, de manera que cortan en el infinito. Como consecuencia, los **epipolos** se encuentran **en el infinito**, es decir, la imagen del centro de proyección en una imagen es paralela al plano de la otra imagen, además los planos epipolares de la figura 3.5 serán paralelos entre sí. Puesto que existe un número infinito de planos fronto-paralelos posibles entre los que elegir, es necesario añadir restricciones como maximizar el solapamiento entre vistas o minimizar la distorsión, entre otros. El proceso da lugar a ocho términos, cuatro para cada una de las cámaras. Para cada cámara tendremos un vector de distorsión, una matriz de rotación (para aplicar a la imagen), la matriz de

parámetros de la cámara sin rectificar y la misma matriz rectificada. A partir de estos términos, se puede hacer un mapa sobre el cual interpolar píxeles de la imagen original para crear una nueva imagen rectificada.

Como hemos comentado, y como conclusión de esta sección, la importancia del proceso de rectificación reside en que **simplifica el problema de correspondencia limitándolo a la búsqueda en una línea epipolar**. Esta es una de las principales asunciones de los algoritmos de correspondencia estéreo.

3.3 Revisión de implementaciones

Hay muchas formas de calcular los términos de rectificación, en OpenCV [67] existen funciones implementadas que permiten desarrollar los dos siguientes algoritmos:

- **Algoritmo de Hartley** [71]. No requiere calibración estéreo ya que solo utiliza la matriz fundamental. Este algoritmo busca **homografías que mapean los epipolos con el infinito** minimizando disparidades calculadas entre dos imágenes estéreo. Para ello, primero se halla un determinado número de correspondencias entre las dos imágenes evitando calcular los parámetros intrínsecos de las cámaras, ya que esta información va implícita. Por tanto, solo es necesario calcular la **matriz fundamental**, que se puede estimar a partir de siete o más correspondencias. Como ventaja, permite la calibración estéreo en tiempo real, sin embargo tiene graves limitaciones debido a que no hay referencias sobre la escala de los objetos (tamaño y distancia) ni se puede tener conocimiento sobre la matriz de parámetros intrínsecos, dando lugar a ambigüedades en la reconstrucción debido a la apariencia de los objetos en función de la perspectiva.

- **Algoritmo de Bouguet** [72]. Es un trabajo que recopila y simplifica el método inicialmente presentado por Tsai [73] y Zhang [74] [75]. Jean-Yves Bouguet nunca publicó este algoritmo más allá de la implementación para Matlab, conocida como ***Camera Calibration Toolbox***. Conocidas las matrices de rotación y traslación $([R], \overrightarrow{t})$, este algoritmo trata de **minimizar los cambios** o transformaciones que la reproyección produce sobre cada una de las imágenes originales (minimizando las distorsiones y deformaciones del resultado) **maximizando el área común** de la escena vista por ambas.

El algoritmo de Hartley se puede utilizar para derivar **estructura a partir del movimiento** capturado por una cámara simple pero puede dar lugar a imágenes más distorsionadas que el algoritmo calibrado de Bouguet. En situaciones en las que se puede utilizar patrones de calibración (i.e. un tablero de ajedrez) lo apropiado es utilizar el algoritmo de Bouguet.

Además de *Camera Calibration Toolbox*, existen varias implementaciones de rectificación para Matlab:

- ***Epipolar Rectification Toolkit*** (RectifKitE) [76] utiliza la información de calibración de las cámaras para rectificar las imágenes. Se trata de un algoritmo de rectificación lineal para cualquier *rig* estéreo en general que toma las dos matrices de proyección perspectiva de las cámaras originales ($[M]$) y calcula un par de matrices proyección de rectificación. El objetivo de los autores es la creación de un algoritmo compacto (22 líneas de código) lo más fácilmente reproducible y usable posible. En la figura 3.8 se presentan los resultados para el par de imágenes estéreo de ejemplo *Sport* (creada por INRIA-Syntim), donde se muestra el par de imágenes rectificadas en la parte inferior además de las líneas epipolares, sobre las vistas derechas, correspondientes a los puntos marcados de las vistas izquierdas.

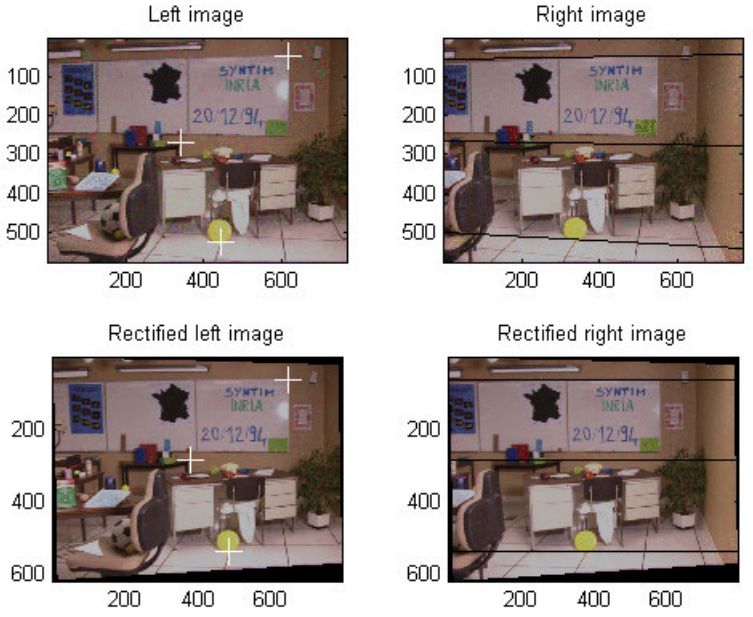


FIGURA 3.8 – **Resultado de RectifKitE.** Par estéreo *Sport* (arriba) y par rectificado (abajo). Las imágenes de la derecha representan las líneas epipolares correspondientes a los puntos marcados en las imágenes de la izquierda.

- ***Uncalibrated Epipolar Rectification Toolkit*** (RectifKitU) [77] trata la rectificación epipolar no calibrada intentando aproximar una transformación sobre las imágenes al caso ideal (calibrado o euclídeo), donde se aplica una transformación colineal inducida por el plano en el infinito. Extendiendo esto al caso no calibrado para minimizar el error de rectificación, se obtiene la **rectificación epipolar quasi-euclídea**. Se basa en un trabajo anterior [78] que propone un método para evitar el cálculo de la matriz fundamental. En el caso euclídeo, la rectificación es geométricamente una rotación de los planos de imagen, que se induce considerando el plano en el infinito (el lugar geométrico de la disparidad cero en el par rectificado). En el caso no calibrado, en el espacio proyectivo, cualquier plano puede ser el de referencia en el infinito. Esto conduce a considerar que la rectificación quasi-euclídea es referida al plano que aproxima el plano en el infinito. Por lo tanto, se asume que los parámetros intrínsecos son desconocidos y que se dispone de un conjunto de correspondencias entre puntos. Siguiendo la línea de [78], busca las transformaciones colineales que hacen que los puntos de las correspondencias satisfagan la geometría epipolar del par de imágenes rectificados. La figura 3.9 muestra los resultados para el par de ejemplo *cporta*.

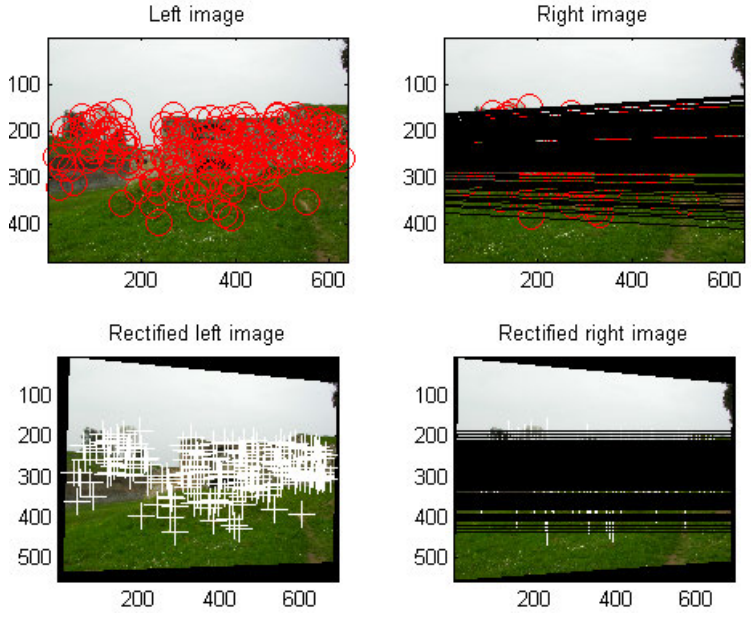


FIGURA 3.9 – **Resultado de RectifKitU.** Par estéreo *cporta* (arriba) y par rectificado (abajo). Las imágenes de la derecha representan las líneas epipolares correspondientes a los puntos marcados en las imágenes de la izquierda.

En el proyecto denominado ***Uncalibrated Stereo Vision*** [79] retoman las ideas expuestas en la descripción del algoritmo de Hartley, es decir el proceso de determinación de la matriz fundamental utilizando un conjunto de correspondencias conocidas o **método de calibración débil**. En dicho proyecto se utiliza el método de calibración débil para intentar determinar una matriz fundamental, rectificar la imagen y utilizar los algoritmos de correspondencia existentes en OpenCV [80] para computar mapas de profundidad. En esta aproximación tratan de implementar un sistema iterativo de correspondencia estéreo con calibración débil para determinar un mapa de profundidad a partir de imágenes estéreo no calibradas. La implementación hace uso de un detector de características (*feature tracker*) para determinar algunas correspondencias iniciales. El proceso sigue las siguientes fases:

1. Encontrar correspondencias iniciales utilizando el detector de puntos característicos. El detector utilizado es una implementación en C de KLT (Kanade-Lucas-Tomasi), basado en el primer trabajo de Lucas y Kanade [81] desarrollado íntegramente por Tomasi y Kanade [82], cuya descripción completa está contenida en [83] y cuya última modificación fue realizada por Birchfield [84]. Sin embargo se demuestra que funciona mal en la localización de correspondencias entre puntos característicos.

2. Determinar la matriz fundamental $[F]$.

3. Rectificar las imágenes usando $[F]$ [85].

4. Aplicar el algoritmo de correspondencia estéreo calibrado de Birchfield y Tomasi [86] para crear un mapa de profundidad.

5. Si el mapa generado es malo, encontrar nuevos puntos de correspondencia buscando en el espacio cercano a las líneas epipolares determinadas originalmente.

6. Repetir desde el paso 1 utilizando los nuevos puntos de correspondencia.

Existen muchas más implementaciones, todas basadas en los mismos conceptos que aparecen explicados en la sección 3.2 de fundamentos teóricos. Parece suficiente el análisis de las técnicas anteriores para comprender las fortalezas y debilidades que caracterizan la rectificación calibrada y no calibrada, y así poder tomar una decisión en función del problema concreto a resolver.

3.4 Discusión y conclusiones

En esta sección describimos cómo hemos afrontado finalmente el problema de la rectificación. Recordamos que la idea principal del proyecto *ImmersiveTV* es la difusión de contenidos de televisión 3D en formato side-by-side donde el receptor se encarga de estimar la información de profundidad a partir del contenido recibido. Como sabemos, en este proyecto trabajamos con un contenido de vídeo real procedente de la grabación en 3D de un partido de fútbol con varias cámaras distribuidas en posiciones diferentes. Lo que se deduce de todo esto es que, por lo general, no dispondremos de la información de calibración de todas las cámaras. Por lo tanto nos interesa la **rectificación no calibrada**. Un algoritmo como el de Hartley, RectifKitU o *Uncalibrated Stereo Vision* podría servirnos para rectificar las imágenes.

Por ahora, hablar de rectificación en tiempo real prácticamente carece de sentido ya que es un proceso muy costoso y poco preciso. Sin embargo, se ha demostrado que con un equipamiento correcto, una cuidadosa calibración de las cámaras y una adecuada producción y postproducción del contenido es técnicamente posible generar contenido rectificado en directo. La rectificación en el lado del receptor es una cuestión crítica a resolver, mediante el uso de algoritmos como los descritos en el presente capítulo.

La adición de un algoritmo de rectificación previo introduce cierto grado de complejidad al sistema de difusión 3DTV propuesto. En primer lugar, podría ser necesario verificar en tiempo real si el contenido que se está recibiendo está rectificado en cada momento. Una posible solución que no añade apenas retardo, sería introducir algún tipo de información de señalización a través de la cual el trasmisor sea capaz de indicar al receptor cuándo debe realizar el proceso de rectificación. Otra posible solución mucho más costosa sería la detección de contenido rectificado en el receptor, que debería trabajar en tiempo real.

Una vez detectada o indicada la necesidad de rectificar, entra en juego el algoritmo o técnica de rectificación. La introducción de un proceso de rectificación previo al módulo de correspondencia estéreo o una posible adaptación de alguno de los algoritmos que hemos explicado para permitir su aplicación en tiempo real podría considerarse para futuros trabajos en esta línea. Pensemos por ejemplo en un algoritmo de rectificación

calibrada, para lo cual es necesario conocer los parámetros de todas las cámaras involucradas en la captura de un evento o programa concreto. En este caso, el retardo viene dado por la transmisión o descarga de dicha información al principio del programa y a continuación, del proceso de cálculo de la transformación geométrica proyectiva que se realiza para cada uno de los cuadros estéreo. Por otra parte, también tiene que existir un método de asociación de los parámetros con cada cámara, es decir, en cada cambio de punto de vista o escena debería recibirse algún tipo de señalización que permita al receptor cambiar los parámetros de transformación que tiene que aplicar en cada momento, o producirse una transmisión de los nuevos parámetros. Todo esto complica los receptores, aumentando los costes, y posiblemente implica algunas modificaciones sobre los estándares de formatos de transmisión y codificación de vídeo.

En el caso de utilizar un algoritmo de rectificación no calibrada, no se transmite ningún parámetro de cámara puesto que la transformación proyectiva se estima a partir de las dos vistas que llegan al receptor en cada momento. Por otra parte, este proceso de rectificación supone más retardo que el caso calibrado, ya que implica más cálculos y además puede ser menos efectivo porque causa más deformaciones sobre las vistas originales. Sin embargo, puesto que se tiene un número limitado de cámaras o puntos de vista en el programa que se transmite, la estimación de las matrices de homografía se podría realizar una vez al principio de cada cambio de escena y se mantendría hasta que se produjese un cambio de punto de vista. Esto nos lleva de nuevo a la necesidad de señalización para los cambios de cámara. Adicionalmente, teniendo en cuenta el número limitado de puntos de vista posibles, podría aplicarse un mecanismo de aprendizaje o memoria en el receptor para que fuera posible asociar cada cámara con sus parámetros estimados. Por ejemplo, el transmisor podría asignar un número o identificador la primera vez que cada enviara señal proveniente de cada cámara conservando los mismos identificadores durante la emisión del programa, y el receptor tener conocimiento de ello. De nuevo los receptores serían más costosos, aunque en el caso más simple, todos los parámetros se estimarían cuadro a cuadro.

Por si todo esto no es suficiente, efectos como el zoom modifican los parámetros de las cámaras complicando bastante todo lo explicado en estos últimos párrafos.

Como hemos demostrado en la introducción de este capítulo (sección 3.1), para finalizar este capítulo podemos concluir que el contenido con el que trabajamos está alineado y no requiere rectificación. Por ello, no hemos incluido un módulo de rectificación en nuestra solución, ya que además introduciría cierto grado de complejidad al ser necesario detectar en tiempo real si el contenido recibido está o no rectificado. Además está fuera del objetivo principal de este trabajo, que se centra fundamentalmente en el estudio de los algoritmos de correspondencia estéreo. Sin embargo, al ser un requisito fundamental de los algoritmos de correspondencia que el contenido esté alineado, es apropiado dedicar un capítulo a explicar los conceptos de calibración y rectificación, que en un caso más general y para cualquier tipo de contenido, ha de tenerse en cuenta.

Capítulo 4

Correspondencia y Reconstrucción

4.1 Introducción

En este capítulo se hace un recorrido a lo largo de los algoritmos de correspondencia, explicando las características que los determinan tales como algunas restricciones a las que están sujetos, posibles clasificaciones de los tipos de algoritmos que nos podemos encontrar, así como la división de su estructura en fases o etapas comunes a la mayoría de ellos. También se describe con cierto detalle algunas de las técnicas más conocidas.

La tarea que sigue al problema de correspondencia es el llamado problema de reconstrucción o reproyección, que consiste en la obtención del valor de profundidad (distancia a la cámara) de cada punto de la escena y se describe en la sección 4.3 explicando su relación con la información de disparidad.

En cuanto a la parte práctica, hacemos un breve resumen de algoritmos de correspondencia con los que se ha experimentado y sobre todo, analizamos los trabajos de evaluación estudiados con el fin de obtener un criterio de elección consistente de los algoritmos finalmente escogidos para resolver nuestro problema. Por tanto, la última sección del capítulo se centra en la descripción razonada de los algoritmos de correspondencia utilizados en nuestra aplicación y cómo evaluamos los resultados obtenidos a partir de ellos. Más adelante, en otro capítulo, se mostrará con mayor detalle las pruebas realizadas y los resultados obtenidos.

4.2 Correspondencia estéreo

4.2.1 Definición de disparidad

El problema de correspondencia consiste en encontrar, a partir del píxel que es resultado de proyectar un punto dado de la escena sobre una cámara (en la imagen de referencia), el píxel correspondiente a su proyección sobre la otra cámara (en la imagen de comparación).

Partiendo de una configuración de imágenes rectificadas, que llamamos fronto-paralela, en la cual las filas de ambas imágenes están correctamente alineadas, podemos garantizar que dado un píxel p_l en la vista izquierda, su correspondiente p_r en la imagen derecha se encuentra en la misma línea horizontal cumpliendo la restricción epipolar (capítulo 3). Definimos **disparidad** en el contexto de la correspondencia estéreo al desplazamiento horizontal entre dos píxeles homólogos, uno visto por la cámara izquierda y el otro por la derecha, que son la proyección de un mismo punto 3D.

Suponiendo que un punto P del mundo físico tiene una proyección visible en cada imagen, p_l y p_r de coordenadas (x_l, y_l) y (x_r, y_r) respectivamente, obtenemos el valor de disparidad como la diferencia:

$$d(x_l) = x_l - x_r \tag{4.1}$$

donde hay que matizar que se denota $d(x_l)$ al valor de disparidad asociado al píxel situado en la coordenada x_l de la imagen izquierda, ya que la definición cambia si se toma como referencia la imagen derecha $d(x_r) = x_r - x_l$, obteniendo el valor de disparidad asociado al píxel situado en la coordenada x_r de la imagen derecha. Los valores $|d(x_l)|$ y $|d(x_r)|$ idealmente deben coincidir, aunque más adelante veremos que esto no siempre sucede debido a causas como las oclusiones o las características propias de los algoritmos de estimación, y que la comparación sirve para comprobar la validez del valor estimado.

La disparidad sólo se puede calcular en la región visual en la que ambas vistas se solapan. Conocidas las coordenadas físicas de las cámaras o los tamaños de los objetos en la escena, podemos inferir medidas de profundidad por medio de la triangulación de la disparidad medida entre los puntos correspondientes en las dos vistas. Sin algún tipo de información física, sólo se puede calcular la profundidad con un factor de escala.

Si no se conoce los parámetros intrínsecos de las cámaras, sólo se puede computar las posiciones de los puntos respecto a una transformación proyectiva (hay ambigüedades).

Gracias a la disparidad, tenemos percepción de profundidad. En la sección 4.3 hablaremos de la relación que existe entre los valores de disparidad y la profundidad percibida. Es necesario aclarar, que por la definición de la ecuación 4.1, según nuestro convenio de signos, un valor negativo en la disparidad se corresponde con el caso descrito en la sección 2.1 ilustrado en la figura 2.3(a) donde se produce un paralaje positivo causando que el objeto parezca estar detrás de la pantalla. Asímismo, el valor de disparidad cero, coincide con el caso de paralaje cero de la figura 2.3(b) donde los objetos se encuentran en el plano de la pantalla. Finalmente, un valor positivo de $d(x_l)$ refleja un paralaje negativo (figura 2.3(c)) en el que los objetos parecen estar situados delante de la pantalla.

4.2.2 Restricciones principales

En el cálculo de la correspondencia, existen restricciones (hipótesis) que se aplican a los algoritmos para hallar los puntos correspondientes. Estas restricciones están basadas en propiedades físicas y geométricas de los objetos y superficies presentes en la escena, y su relación. Las restricciones en ocasiones se implementan y utilizan explícitamente, pero por lo general están implícitas [87]. Las más comunes son las siguientes:

- **Restricción epipolar:** viene dada por la geometría epipolar del par estéreo explicada en la sección 3.2.3. Determina que la correspondencia de un punto en una imagen **se debe buscar a lo largo de su línea epipolar** en la otra imagen. Esta restricción simplifica enormemente el proceso de correspondencias reduciendo el coste computacional y elimina ambigüedades entre posibles candidatos.
- **Semejanza (*similarity*):** la restricción de semejanza es fundamental y establece que las intensidades o **características** de los píxeles correspondientes al mismo punto de un objeto en las imágenes izquierda y derecha deben ser **casi idénticas para todos los píxeles correspondientes**. Por tanto, sus valores de color e intensidad no deben variar de manera significativa entre distintos puntos de vista. El márgen de variación admisible depende en gran medida de la cámara con la que se toman las imágenes.

- **Unicidad (*uniqueness*):** la restricción de unicidad también es esencial y define que, en la mayoría de los casos, para cada píxel de una imagen de entrada **existe a lo sumo uno correspondiente** en la otra imagen. De hecho, se contempla que pueda no existir píxel correspondiente en la segunda imagen como puede ocurrir en el caso de producirse una oclusión o que el píxel correspondiente esté fuera de la imagen. Pero lo que no se puede admitir es que un píxel tenga más de una correspondencia.
- **Suavidad (*smoothness*) o continuidad (*continuity*):** la restricción de suavidad, también conocida como restricción de continuidad, aparece bastante en el campo de la investigación de los algoritmos de correspondencia estéreo modernos. Esta restricción postula que **la profundidad de las superficies** físicas en el mundo real es una propiedad que **varía suavemente**, excepto en los bordes de los objetos donde se producen discontinuidades.
- **Restricción de orden:** esta restricción determina que **los puntos** de la imagen de referencia **aparecen en el mismo orden** dentro de la imagen de comparación, y aunque es posible que alguno de estos puntos no sea visible debido a alguna oclusión, el resto de puntos aparecerán en el orden original. Por ejemplo, si la proyección del punto P está a la izquierda de la proyección del punto Q en la imagen izquierda, entonces la proyección del punto P estará a la izquierda de la proyección del punto Q en la imagen derecha.

Las que se explican a continuación pueden ser consideradas restricciones ya que los algoritmos las tienen en cuenta en su implementación, aunque sería más correcto clasificarlas como problemas a enfrentar.

- **Ausencia de textura (*textureness*):** las superficies con textura uniforme o ausencia de textura **generan ambigüedades** dificultando el proceso de selección del mejor candidato para establecer una correspondencia. Como consecuencia se puede obtener correspondencias espúreas, es decir, píxeles a los que se les asigna un valor de disparidad válido que no se corresponde con el correcto.
- **Oclusiones:** constituye un aspecto muy importante a tener en cuenta en el planteamiento de la visión estéreo ya que sucede en puntos donde no es posible encontrar correspondencias. Se produce una oclusión cuando un **punto** en el espacio tridimensional comprendido delante de las cámaras es **representado en una de**

las imágenes mientras que aparece bloqueado en la otra. Esto ocurre porque puede haber un objeto visible delante de dicho punto y según la perspectiva, en una de las vistas resulta ocluido. Implican una discontinuidad en el valor de profundidad de la escena y son fuente de numerosos errores en muchos algoritmos estéreo. Sin embargo, estas regiones ocultas aportan información valiosa sobre la estructura de la escena. No existe una solución general al problema de las oclusiones y puesto que una oclusión puede derivar en correspondencias espúreas, uno de los objetivos en la investigación de la correspondencia estéreo es la minimización del impacto de las oclusiones sobre el resultado final. Existe una clasificación de algoritmos en función de cómo abordan el problema de las oclusiones [88]:

- **Algoritmos que detectan oclusiones** buscando discontinuidades de disparidad. Pueden dar lugar a falsas detecciones.
- **Algoritmos que disminuyen la sensibilidad a las oclusiones** basándose en medidas de semejanza robustas (como *census*, tabla 4.1).
- **Algoritmos que modelan las oclusiones** teniendo en cuenta las restricciones que implican, como por ejemplo en [89].

Otra manera de enfrentar el problema de oclusiones es la utilización de una tercera vista, pero este caso se sale del ámbito de la correspondencia estéreo.

4.2.3 Principales etapas del proceso de correspondencia

Los algoritmos que resuelven el problema de correspondencia pueden estructurarse, por lo general, en **cuatro fases principales**: coste de correspondencia, agregación de coste, cálculo de disparidad y optimización, y refinamiento de disparidad. Aunque no todos ellos atraviesen las cuatro etapas, esta división permite compararlos eficientemente de forma genérica [16]. A continuación describimos los cuatro bloques.

1. **Coste de correspondencia:**

   Esta fase es fundamental, sobre todo para los algoritmos locales. Se realiza la comparación entre los píxeles de las imágenes izquierda y derecha. La medida de semejanza se suele obtener a partir de una **función de coste** que compara intensidades de los píxeles de ambas imágenes [90]. El coste de correspondencia se computa y almacena. Normalmente, cuanto menor es el valor de coste, mayor es la probabilidad de encontrar la correspondencia correcta. En la tabla 4.1 recogemos las funciones de coste más extendidas.

Medida de semejanza	**Definición**
Diferencia Absoluta [90]	$C_{AD}(x,y,d) = \lvert I_L(x,y) - I_R(x-d,y)\rvert$
Diferencia al Cuadrado [16, 17, 91]	$C_{SD}(x,y,d) = \lvert I_L(x,y) - I_R(x-d,y)\rvert^2$
Gradiente [90]	$C_{GRAD}(x,y,d) = \lvert\nabla_x I_L(x,y) - \nabla_x I_R(x-d,y)\rvert + \lvert\nabla_y I_L(x,y) - \nabla_y I_R(x-d,y)\rvert$
Birchfield-Tomasi (BT) [86, 92]	$C_{BT} = d(x_L,x_R) = \min\left\{\bar{d}(x_L,x_R,I_L,I_R), \bar{d}(x_R,x_L,I_R,I_L)\right\}$ $\bar{d}(x_L,x_R,I_L,I_R) = \min_{x_R-\frac{1}{2}\le i\le x_R+\frac{1}{2}} \left\lvert I_L(x_L) - \hat{I}_R(i)\right\rvert$ $\bar{d}(x_R,x_L,I_R,I_L) = \min_{x_L-\frac{1}{2}\le i\le x_L+\frac{1}{2}} \left\lvert \hat{I}_L(i) - I_R(x_R)\right\rvert$ $\hat{I}_k(i-\frac{1}{2}) = \frac{1}{2}\left(I_k(i) + I_k(i-1)\right)$
Suma de Diferencias Absolutas (SAD) [16]	$C_{SAD}(x,y,d) = \sum_{(i,j)\in N(x,y)} \lvert I_L(i,j) - I_R(i-d,j)\rvert$
SAD de Media Cero (ZSAD) [28]	$C_{ZSAD}(x,y,d) = \sum_{(i,j)\in N(x,y)} \left\lvert \left(I_L(i,j) - \bar{I}_L\right) - \left(I_R(i-d,j) - \bar{I}_R\right)\right\rvert$
Suma de Diferencias al Cuadrado (SSD) [16]	$C_{SSD}(x,y,d) = \sum_{(i,j)\in N(x,y)} \lvert I_L(i,j) - I_R(i-d,j)\rvert^2$
SSD de Media Cero (ZSSD) [28]	$C_{ZSSD}(x,y,d) = \sum_{(i,j)\in N(x,y)} \left\lvert \left(I_L(i,j) - \bar{I}_L\right) - \left(I_R(i-d,j) - \bar{I}_R\right)\right\rvert^2$
Correlación Cruzada Normalizada (NCC) [93]	$C_{NCC}(x,y,d) = \frac{\sum_{(i,j)\in N(x,y)} I_L(i,j)\cdot I_R(i-d,j)}{\sqrt{\sum_{(i,j)\in N(x,y)} I_L^2(i,j) \cdot \sum_{(i,j)\in N(x,y)} I_R^2(i-d,j)}}$
NCC de Media Cero (ZNCC) [28]	$C_{ZNCC} = \frac{\sum_{(i,j)\in N(x,y)} \left(I_L(i,j) - \bar{I_L}\right)\cdot\left(I_R(i-d,j) - \bar{I_R}\right)}{\sqrt{\sum_{(i,j)\in N(x,y)} \left(I_L(i,j) - \bar{I_L}\right)^2 \cdot \sum_{(i,j)\in N(x,y)} \left(I_R(i-d,j) - \bar{I_R}\right)^2}}$
Suma de Gradientes [90]	$C_{SGRAD}(x,y,d) = \sum_{(i,j)\in N(x,y)} \lvert\nabla_x I_L(i,j) - \nabla_x I_R(i-d,j)\rvert + \sum_{(i,j)\in N(x,y)} \lvert\nabla_y I_L(i,j) - \nabla_y I_R(i-d,j)\rvert$
Rank [94]	$C_{RANK}(x,y,z) = \sum_{(i,j)\in N(x,y)} \left(\dot{I_L}(i,j) - \dot{I_R}(i-d,j)\right);$ $\dot{I_k}(i,j) = \sum_{(m,n)\in M\times N} I_k(m,n) < I_k(i,j)$
Census (SHD) [94]	$C_{CENSUS}(x,y,z) = \sum_{(i,j)\in N(x,y)} HAMMING\left(\dot{I_L}(i,j), \dot{I_R}(i-d,j)\right);$ $\dot{I_k}(i,j) = BITSTRING_{m,n}\left(I_k(m,n) < I_k(i,j)\right)$

TABLA 4.1 – **Funciones de coste o métricas de semejanza típicas.**

La notación habitual para representar el coste es a través de una función $C(x, y, d)$ de las coordenadas de la imagen de referencia y de la disparidad. La función de coste devuelve el valor de desigualdad o diferencia para una coordenada (x, y, d) en el espacio de disparidad. El espacio de disparidad está definido por las coordenadas de los píxeles disponibles de la imagen y el rango de búsqueda de disparidad. Normalmente, el rango de búsqueda de disparidad se tiene que especificar manualmente y depende de las características del par de imágenes de entrada.

La Correlación Cruzada Normalizada (NCC) es el método estadístico estándar para determinar la semejanza. Su **normalización**, y en general la de cualquier función de coste lo hace relativamente **insensible a la distorsión radiométrica**. La Suma de Diferencias al Cuadrado (SSD) es más simple computacionalmente, al igual que la Suma de Diferencias Absolutas (SAD) que suele ser utilizada por eficiencia computacional. Debido a la forma de las imágenes (generalmente rectangular), SAD se presta a la utilización de ventanas cuadradas (agregación).

Otra de las funciones a destacar es la de **Birchfield y Tomasi (BT)**, que garantiza precisión a nivel de subpíxel, ya que hace uso de la interpolación lineal de la intensidad en el intervalo comprendido entre dos píxeles vecinos logrando que la función de coste no sea sensible al muestreo de la imagen.

2. **Agregación de coste:**

   Esta fase consiste en agrupar el coste de los píxeles vecinos pertenecientes a una región. La forma de **agrupar el coste** puede ser una suma, el valor medio dentro de la región o el promedio de los valores de coste de los píxeles vecinos de un píxel concreto. Las regiones de agrupación pueden ser ventanas cuadradas de tamaño fijo o variable, que habitualmente son desplazables, o núcleos de convolución como el *box filter* (filtro de media separable), filtro binomial o métodos de difusión, entre otros. La principal ventaja de la agregación es la reducción de la alta sensibilidad al ruido que sufre la comparación de píxeles individuales. Sin embargo, los métodos basados en área tienden a generar resultados con menor detalle. Por lo tanto, se debe alcanzar un **compromiso entre la sensibilidad al ruido y la pérdida de detalle** cuando se selecciona el tamaño de la ventana de agregación.

3. **Cálculo de disparidad y optimización:**

   En esta fase se aplica la lógica utilizada para la **selección del valor de disparidad** entre los candidatos. Depende del método escogido. En los métodos locales, partiendo de los costes agregados en el espacio de disparidad, podría buscarse el

valor de coste mínimo (estrategia *Winner-Take-All* (WTA)). En los métodos globales, se minimiza una función más compleja y la optimización depende de la técnica concreta utilizada (programación dinámica, optimización de líneas de búsqueda, enfriamiento simulado, corte de grafos,...).

4. **Refinamiento de disparidad:**
   Etapa de posprocesado para refinar el resultado con el objetivo de reducir el ruido y suavizar la imagen de profundidad. Este paso puede considerarse independiente del algoritmo de correspondencia y puede consistir en el ajuste de precisión a nivel de subpíxel o un filtrado paso bajo para eliminar falsos positivos, así como verificaciones de la validez de ciertas correspondencias que se pueden corregir (eliminación de agujeros).

4.2.4 Tipos de algoritmos de correspondencia

Es difícil definir una única clasificación de los algoritmos de correspondencia debido a la existencia de múltiples aspectos que los diferencian. En primer lugar se pueden clasificar desde el punto de vista del **tipo de resultado** que se obtiene:

- **Algoritmos de correspondencia discreta:** establecen la correspondencia para un conjunto discreto de píxeles, por lo tanto sólo se dispondrá del valor de profundidad de un subconjunto de píxeles de la imagen dando lugar a un mapa disperso. Los métodos más conocidos son los denominados **algoritmos en dos partes** [95]:
  - **SIFT (*Scale Invariant Feature Transform*) [96]:** en primer lugar, detecta puntos característicos (*features*) como bordes o esquinas y guarda información de su posición. Además, se almacena información sobre los píxeles de su entorno como la posición, orientación y valores de intensidad en una estructura de datos conocida como descriptor. De esta manera se puede reconocer el mismo punto en otras imágenes que contengan al mismo objeto. Las características detectadas son **invariantes al escalado y la rotación** aunque sensibles a los cambios de iluminación entre imágenes. La segunda parte consiste en la correspondencia entre características detectadas en ambas vistas.
  - **SURF(*Speeded Up Robust Features*) [97]:** se basa en los mismos principios explicados del método anterior pero utiliza otras técnicas de optimización

mediante la convolución de imágenes completas y las simplifica al mínimo necesario para realizar la detección de forma **más rápida y robusta**.

- **Algoritmos de correspondencia densa:** se establece la correspondencia para todos los píxeles de la imagen de referencia.

En nuestro caso lo que interesa es obtener un **mapa de disparidad denso** ya que es necesario conocer la profundidad de cada píxel de la imagen para poder así calcular las oclusiones del elemento sintético que se insertará posteriormente. El trabajo de Scharstein y Szeliski [16] es la referencia más importante de este campo ya que desarrolla la **primera taxonomía o clasificación de algoritmos de correspondencia densa** basados en dos imágenes. En ella se definen los siguientes tipos:

- **Métodos locales:** se busca la correspondencia comparando píxeles contenidos en una ventana o vecindad, dentro de una región de la imagen. Explotan **información puramente local**, como la semejanza de patrones de brillo cerca de los candidatos a la correspondencia. Los métodos locales pueden ser muy eficientes, pero presentan sensibilidad en las regiones localmente ambiguas (e.g. regiones ocluidas o regiones con textura uniforme).
- **Métodos de optimización global:** las técnicas de fusión estéreo presentadas en el punto anterior son puramente locales, en el sentido de que comparan patrones de brillo y bordes alrededor de píxeles individuales, pero ignoran las restricciones que pueden relacionar puntos próximos. Como contraste, las aproximaciones globales formulan el problema como una **minimización de una función de energía única** incorporando restricciones de orden y suavidad sobre píxeles adyacentes. Estas técnicas pueden ser menos sensibles a los problemas locales ya que las restricciones globales proporcionan ayuda adicional para las regiones difíciles de corresponder localmente. Sin embargo, estos métodos suponen mayor coste computacional.
- **Programación dinámica:** es una técnica que se aplica para minimizar una función global de energía en tiempo polinomial. Consiste en buscar el **camino de menor coste a lo largo de una línea de búsqueda** (*scanline*). Aplica fundamentalmente la restricción de orden.
- **Algoritmos cooperativos:** finalmente, los algoritmos cooperativos, inspirados por los modelos computacionales de la visión estéreo humana, fueron prácticamente los primeros métodos propuestos para la computación de disparidad [98–101]. Tales algoritmos realizan cálculos locales iterativamente, pero usan operaciones no

lineales que dan lugar a un comportamiento general similar a los algoritmos de optimización global. De hecho, para algunos de estos algoritmos, es posible especificar una función global para ser minimizada [102]. Una variante del algoritmo original fue desarrollada en [103].

A lo largo de los años se ha ido actualizando y matizando según evoluciona la investigación. Por simplicidad, en [88] se diferencia a los algoritmos **según el tipo de restricciones que aplican**, distinguiendo únicamente dos: las restricciones locales que se aplican a una pequeña cantidad de píxeles que rodean al píxel de interés y las restricciones globales para referirse a aquellas restricciones sobre líneas de búsqueda o la imagen completa. En la tabla 4.2 se recogen los principales métodos excluyendo aquellos que utilizan más de dos vistas.

Aproximación	**Referencias**	**Descripción**
Métodos locales		
Block Matching	[94, 104–106]	Busca el máximo grado de correspondencia o el error mínimo sobre una región pequeña, típicamente usando variantes de correlación cruzada o métricas de rango (*rank*) robusto.
Optimización basada en gradiente	[81, 107]	Minimiza un funcional, típicamente la suma de diferencias al cuadrado, sobre una región pequeña.
Correspondencia de características	[108–113]	Corresponde características fiables en lugar de las propias intensidades.
Métodos globales		
Programación dinámica	[86, 114–117]	Determina la superficie de disparidad para una línea de búsqueda como el mejor camino entre dos secuencias de primitivas ordenadas. Típicamente, el orden viene dado por la restricción de orden epipolar.
Curvas intrínsecas	[118, 119]	Mapea las líneas de búsqueda epipolar al espacio de curvas intrínsecas para convertir el problema de búsqueda en un problema de búsqueda de vecinos cercanos. Las ambigüedades se resuelven usando programación dinámica.
Corte de grafos (*graph cuts*)	[19, 120–124]	Determina la superficie de disparidad como el mínimo corte del flujo máximo en un grafo.
Difusión no lineal	[102, 125, 126]	Aplica un proceso de difusión local para agregar el coste.
Belief Propagation	[127]	Resuelve las disparidades por medio del paso de mensajes en una red de confianza.
Métodos sin correspondencias	[128–130]	Deforman un modelo de la escena basándose en una función objetivo.

TABLA 4.2 – **Clasificación de algoritmos de correspondencia según [88]**

Por otra parte, se realizan clasificaciones de métodos que implementan una sola de las cuatro etapas del proceso de correspondencia explicadas sin tener en cuenta el resto [21].

Hasta ahora, por lo general hemos hablado de métodos locales, más eficientes computacionalmente en términos de tiempo de proceso y memoria utilizada pero que dan lugar a un mapa de disparidad de menor calidad que no tiene en cuenta propiedades de la imagen completa; y de métodos globales, que tienen en cuenta ciertas restricciones de toda la imagen que ayudan a obtener un mapa de disparidad de mayor calidad a cambio de un mayor coste computacional. Sin embargo, siempre se ha buscado un **compromiso entre eficiencia y calidad** apareciendo técnicas híbridas o **semiglobales** que tratan de reunir las ventajas de ambos tipos de métodos [131–133]. Además, se pueden utilizar otros tipos de técnicas de visión artificial como la **detección de bordes** o geometría de la escena y la **segmentación**, combinadas con los algoritmos de correspondencia para obtener resultados más robustos [134, 135].

4.2.5 Correspondencia densa

En esta sección describimos los cuatro tipos de algoritmos de correspondencia densa más significativos para ilustrar los posibles planteamientos de la solución que nos podemos encontrar. Generalmente, sirven como punto de partida para otras aproximaciones aunque también existen otras con un planteamiento radicalmente distinto.

4.2.5.1. Técnicas locales

En el caso de los algoritmos basados en métodos locales, la disparidad en un punto depende únicamente de los valores de intensidad en torno a una ventana finita, en la cual se realiza suposiciones de suavizado mediante agregación de costes. Estos algoritmos se pueden descomponer en los pasos 1, 2 y 3. Por ejemplo, el algoritmo basado en la suma de diferencias cuadradas (SSD) se puede describir como:

1. El coste de correspondencia es la diferencia al cuadrado de los valores de intensidad entre píxeles.
2. La agregación se realiza mediante la suma de costes en ventanas cuadradas de disparidad constante.
3. La disparidad se calcula eligiendo el menor coste de agregación en cada pixel.

***Block Matching* o correspondencia de bloques**.

Se trata de un método de correspondencia densa local donde se utiliza como soporte de agregación de costes un área correspondiente a **una ventana cuadrada** [67, 136–138]. En primer lugar, se divide la imagen de referencia en bloques de igual tamaño. Para cada uno de estos bloques, se computa una función de coste utilizando como comparación las intensidades de los píxeles contenidos en una ventana del mismo tamaño y desplazada de posición dentro de la imagen de comparación (figura 4.1).

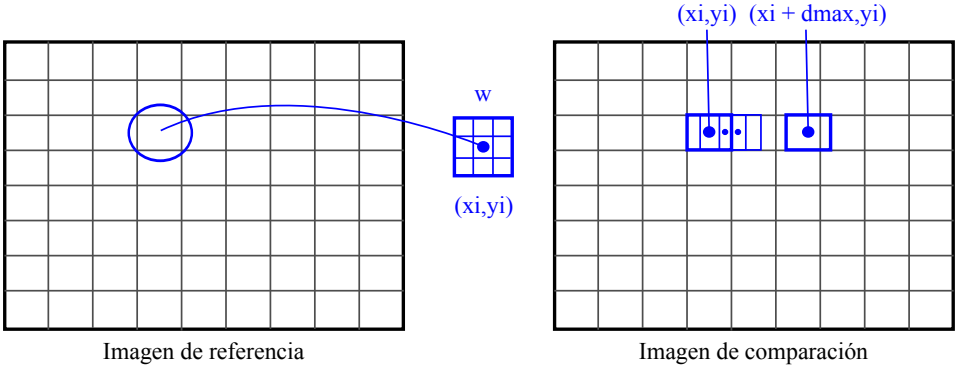


FIGURA 4.1 – ***Block Matching* o correspondencia de bloques:** se compara una ventana de la imagen de referencia con versiones desplazadas de la misma en la imagen de comparación.

En cuanto a la función de coste (y agregación) empleada, puede utilizarse cualquiera de las recogidas en la segunda parte de la tabla 4.1, que reúne aquellas que hallan la suma dentro de una vecindad. La optimización utilizada para asignar un valor de disparidad a cada bloque se basa en la estrategia ***Winner Take All***, que consiste en tomar para todo el bloque un único valor de disparidad correspondiente al menor coste de entre todos los candidatos.

Este método es bastante rápido y sencillo de implementar. Sus **parámetros** fundamentales son el rango de búsqueda de disparidad, definido por sus extremos d_{min} y d_{max} y el ancho de la ventana cuadrada win_{size}.

El principal problema que presenta es que no tiene en cuenta la **restricción de suavidad**, en el sentido de que el valor de disparidad asignado para un bloque no se ve afectado por la disparidad obtenida para los bloques vecinos, aunque sí en el sentido de asignar un único valor de disparidad para todo un bloque de píxeles cercanos.

Es importante escoger un **tamaño adecuado de ventana**, ya que si es demasiado pequeña no tomará suficientes muestras de variación de intensidad dando problemas en

zonas sin textura. Por otro lado, si la ventana es demasiado grande, se obtiene un mapa de disparidad suavizado y con pérdida de detalle. Para compensar estos efectos se puede utilizar ventanas con tamaño variable, para lo cual es necesario conocer alguna información de bordes, o asignar pesos diferentes a los píxeles dentro de una misma ventana.

4.2.5.2. Técnicas globales y planteamiento del problema MRF

Los métodos globales realizan el proceso de optimización minimizando una función de energía determinada por ciertas restricciones aplicadas sobre toda la imagen. Se puede plantear como un problema estadístico basado en ***Markov Random Fields*** **o Campos aleatorios de Markov** [20, 139].

Dado un conjunto de píxeles **P** y un conjunto de etiquetas **L**, que serán las magnitudes que queremos medir para cada píxel (intensidades, disparidades o vectores de movimiento), el objetivo es asignar una etiqueta f_p a cada píxel p. Consideramos que las etiquetas varían suavemente en casi todas las regiones, pero podría haber variación brusca en zonas como bordes de objetos. Para ello, se evaluará una **función de energía** con la siguiente estructura:

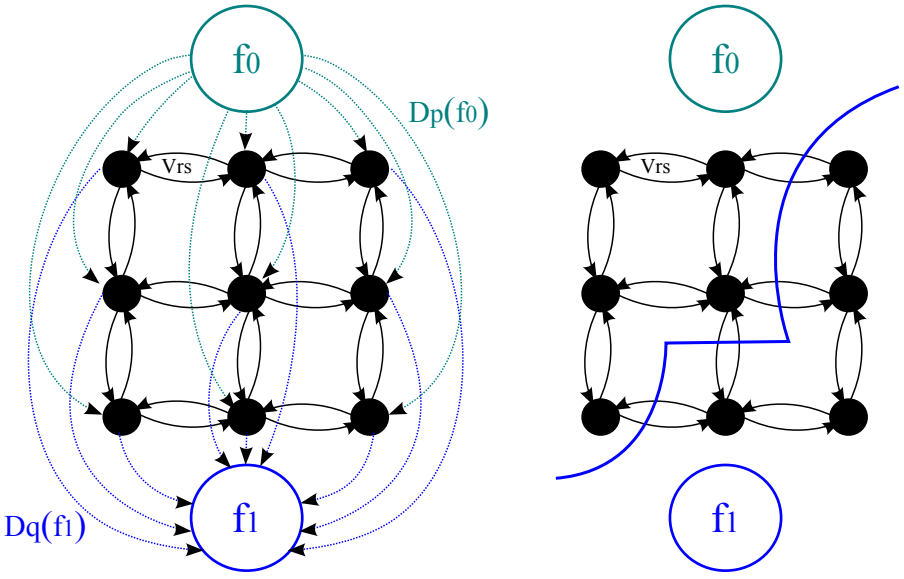
$$E\left(f_p\right) = E_{Data}\left(f_p\right) + \lambda \cdot E_{Smooth}\left(f_p\right) = \sum_{p \in P} D_p\left(f_p\right) + \lambda \sum_{(p,q) \in N} V\left(f_p, f_q\right) \tag{4.2}$$

donde el primer término $D_p(f_p)$ se denomina ***Data Cost*** definido como el coste de asignar al píxel p la etiqueta f_p. El segundo término $V(f_p, fq)$ se denomina ***Discontinuity Cost*** y se refiere al coste de asignar a dos píxeles vecinos p y q las etiquetas f_p y f_q, respectivamente, que habitualmente equivale al valor de la diferencia de ambas etiquetas. Denotamos como **N** a una vecindad o un entorno de vecinos conectados a 4, es decir, se consideran como vecinos aquellos píxeles contiguos laterales, superior e inferior.

Se conoce como **MAP** (*mínimum a-posteriori*) al tipo de problema de estimación que tiene el objetivo de asignar las etiquetas que tengan menor coste de energía para un MRF definido [140, 141]. La complejidad de este problema es máxima (*NP-hard*). A continuación se presentan algunas técnicas para abordar este problema.

***Graph Cuts* o corte de grafos**.

Este método se basa en la construcción de un **grafo** plano cuyos **nodos son los píxeles** de la imagen y las **aristas** que interconectan dichos nodos tienen asociados un coste o penalización relacionado con el *discontinuity cost* (V_{rs} en la figura 4.2(a)). Es decir, se penalizan los nodos vecinos que tengan un salto de disparidad. Además de los nodos principales, existen unos **nodos virtuales o terminales** correspondientes a las posibles etiquetas. En la figura 4.2(a) se representa un grafo que, por simplicidad, admite únicamente dos etiquetas (f_0 y f_1), pero en el caso de la correspondencia estéreo habrá tantas etiquetas como niveles de disparidad (Número de etiquetas= $d_{max} - d_{min} + 1$). La arista que une cada nodo píxel con el nodo terminal tiene asociado un coste o penalización relacionado con el *data cost* ($D_p(f_0)$ y $D_p(f_1)$). Es decir, se refleja la penalización por asignar a cada píxel las posibles etiquetas.



(a) **Grafo dirigido con dos etiquetas** (b) **Corte mínimo del grafo**

FIGURA 4.2 – ***Graph cuts*:** se agrupan los nodos del grafo en función de la etiqueta asignada a través del corte que minimiza el coste.

El problema consiste en asignar el **etiquetado óptimo de manera que se minimice la función global de energía** [142]. La forma de resolver el etiquetado óptimo se puede plantear como la realización de cortes en las aristas del grafo de manera que queden agrupados todos los nodos con la misma etiqueta (i.e. valor de disparidad), como se esquematiza en la figura 4.2(b). Este planteamiento se denomina ***min-cut*** [19].

Otro planteamiento de resolución es el denominado ***max-flow*** [122–124], que consiste en encontrar el camino por el cual se puede llevar la máxima cantidad de agua de

f_0 a f_1, suponiendo que las aristas son tuberías con capacidad proporcional a su coste. Es importante destacar que *max-flow* es equivalente a *min-cut*, de manera que los cortes en el grafo se resuelven habitualmente mediante algoritmos de *max-flow*.

Lo explicado anteriormente es una simplificación, porque en nuestro caso trabajamos con múltiples etiquetas, de manera que el problema resultante es de complejidad NP. En este caso, se utilizan unos algoritmos denominados ***move-making***, que se basan en mantener o cambiar la etiqueta que un nodo tenga asignado en un momento determinado, de manera que se puede considerar como un problema binario (cambiar o no cambiar). Los más conocidos son ***Swap*** y ***Expansion*** [120, 143], algoritmos iterativos que realizan una serie de movimientos permitidos para reducir la energía global.

Las técnicas basadas en graph cuts teóricamente obtienen resultados con bastante calidad, sin embargo consumen muchos recursos y sigue siendo un problema muy complejo que requiere tiempo de computación.

***Belief propagation* o propagación de confianza**.

Esta aproximación permite reducir el problema a un simple cálculo del **valor mínimo de una suma** [18, 127]. Se basa en el **traspaso de mensajes** entre nodos vecinos a lo largo del grafo definido por una malla que está formada por los píxeles de la imagen conectados a 4, como en la figura 4.3.

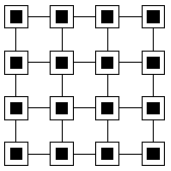


FIGURA 4.3

En [144] se describe un algoritmo iterativo bastante sencillo que permite entender los aspectos principales de la optimización basada en *belief propagation*. En cada instante t, todos los nodos generan un mensaje que al final de la iteración será transmitido a sus vecinos. Supongamos que el nodo p genera un mensaje, como el de la ecuación 4.3, dirigido a su nodo vecino q:

$$m_{pq}^{t}\left(f_q\right) = \min_{f_p}\left(V\left(f_p, f_q\right) + D_p\left(f_p\right) + \sum_{s \in N(p)\setminus q} m_{sp}^{(t-1)}\left(f_p\right)\right) \tag{4.3}$$

donde $N(p) \setminus q$ representa al conjunto de nodos vecinos de p que no son vecinos de q. $V(f_p, f_q)$ es lo que hemos denominado *discontinuity cost* y se calcula como la diferencia entre los valores de las etiquetas f_p y f_q asignadas a los nodos p y q, respectivamente, para penalizar las discontinuidades o saltos de disparidad entre dos píxeles vecinos.

Finalmente, $m_{sp}^{(t-1)}$ es el mensaje que llegó a p procedente de un vecino s (que no es vecino de q) en el instante de tiempo (o iteración) anterior.

En la última iteración, que denotamos como **T**, se calcula un **vector de confianza o *belief vector*** para cada nodo. La longitud de este vector depende del rango de disparidad considerado, es decir, se almacenarán $d_{max} - d_{min} + 1$ mensajes finales asociados a cada nodo. El mensaje final generado por el nodo q y asociado a la etiqueta (o valor de disparidad) f_q viene dado por la ecuación siguiente:

$$b_q(f_q) = D_q(f_q) + \sum_{p \in N(q)} m_{pq}^{T}(f_q) \tag{4.4}$$

donde $D_q(f_q)$ se corresponde con el denominado *data cost*, esto es, el coste de asignar al nodo q la etiqueta f_q y puede ser calculado mediante alguna de las funciones de coste de la tabla 4.1 que comparen píxeles individuales. El término m_{pq}^{T} se refiere al mensaje que llegó a q desde p en la última iteración **T**, siendo p cada uno de sus vecinos.

El valor resultante asociado a la etiqueta f_q se almacena en la posición de índice f_q dentro del vector de confianza. De esta manera, resulta sencillo escoger, para cada nodo, la etiqueta f_q que minimiza $b_q(f_q)$.

La principal ventaja de este método es la simplificación de la resolución del problema formulado como MRF, sin embargo requiere gran cantidad de memoria para computar todos los mensajes, siendo complejidad de orden $O(nk^2T)$, donde n es el número de píxeles de la imagen, k el número de etiquetas (o valores de disparidad) posibles dado por el parámetro n_{disp} y **T** el número de iteraciones que viene determinado por el parámetro n_{iter}.

Existe además una variante **multirresolución o multiescala** [145], con el parámetro adicional n_{levels} que determina cuantos niveles de resolución va a resolver, y permite obtener un resultado bastante aproximado reduciendo el tiempo de computación a costa de perder precisión o nivel de detalle.

***Dynamic Programming Stereo* o programación dinámica**.

La programación dinámica es un método aplicado en la resolución de numerosos problemas de cálculo. Se basa en reducir la complejidad de cálculo de la optimización mediante la descomposición en subproblemas menores. En visión estéreo, se trata de un método global que se resuelve en tiempo polinomial. Las principales asunciones de esta técnica son la restricción epipolar y de orden, ya que se basa en la búsqueda de un **camino de mínimo coste a lo largo de una línea de búsqueda** (o *scanline*) [115].

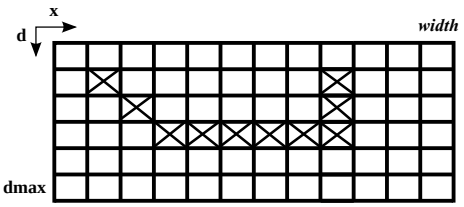


FIGURA 4.4 – **DSI: *Disparity Space Image***

Para poder encontrar el camino de menor coste es necesario construir la **imagen del espacio de disparidad o DSI** (*disparity space image*) correspondiente a cada línea de búsqueda. La DSI almacena los costes de asignar a los píxeles de cada línea de búsqueda, cada uno de los posibles valores de disparidad. Es decir, como se esquematiza en la figura 4.4, en el eje horizontal se representa la coordenada horizontal de cada píxel de la línea y en el eje vertical los posibles valores de disparidad, por lo tanto en cada celda de la DSI se almacena el coste correspondiente. Una vez obtenida, tratamos de recorrerla de izquierda a derecha buscando el camino con menor coste posible.

Un ejemplo importante es el algoritmo de Bobick-Intille [89], donde se propone modelar las oclusiones restringiendo los movimientos posibles a través de la DSI para construir el camino de menor coste:

- **Horizontal:** un desplazamiento horizontal a través de la DSI significa un valor constante de disparidad que se corresponde con un plano frontal. Es el resultado de una **buena correspondencia**.
- **Diagonal:** un desplazamiento en diagonal hacia abajo representa una **oclusión** en la *scanline* izquierda. Es decir, un píxel en la imagen izquierda tiene varios posibles candidatos en la derecha.
- **Vertical:** un desplazamiento hacia arriba en la DSI representa una **oclusión** en la *scanline* derecha. Es decir, un píxel en la imagen derecha tiene varios posibles candidatos en la izquierda.

La capacidad de la programación dinámica para minimizar la función de coste en tiempo polinomial hace que sea un buen candidato para las aplicaciones en tiempo real. Sin embargo, como no aplica la restricción de suavidad entre líneas de búsqueda, se obtiene un **mapa de disparidad rayado**, formado por líneas horizontales, aunque existen planteamientos donde se tiene en cuenta condiciones de continuidad entre scan-lines [114, 117]. También su precisión es peor que la de otros métodos como *semiglobal matching* [131–133], *Belief Propagation* o *Graph Cuts*, aunque actualmente se busca la optimización del rendimiento aprovechando la potencia de GPU [146].

4.3 Reconstrucción o reproyección

Como sabemos de la sección 2.2, una vez discriminada la imagen correspondiente a cada ojo, la disparidad binocular produce sensación de profundidad [13]. La figura 4.5 muestra la posición de la **profundidad percibida** de los puntos debido a distintas disparidades, **según nuestro convenio de signos**. Tanto el signo como la magnitud de la disparidad determinan la posición del punto 3D percibido.

Como también se discute en trabajos anteriores [14, 15], la profundidad percibida (Z) de un punto 3D se relaciona con la disparidad mediante la siguiente ecuación:

$$Z = \frac{b \cdot D}{b + d} \tag{4.5}$$

donde b representa la distancia entre los dos ojos, y D la distancia de observación, que está relacionada con la convergencia.

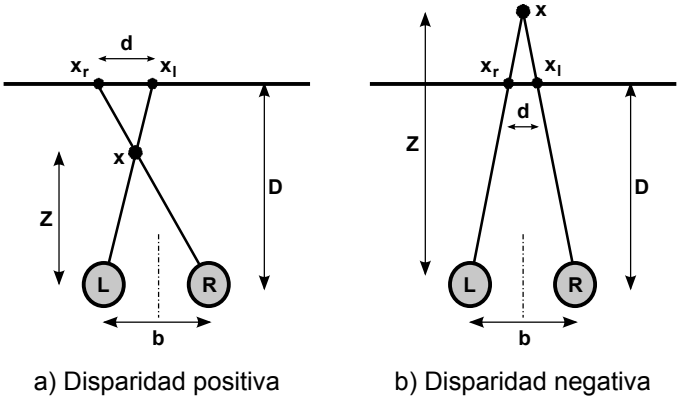


FIGURA 4.5 – **Profundidad percibida en función del signo de disparidad:** Los objetos con disparidad positiva (paralaje negativo) parecen estar delante de la pantalla, mientras que los objetos con disparidad negativa (paralaje positivo aparecen detrás de la pantalla.

Una cámara estéreo imita el sistema visual humano tomando imágenes de una escena desde posiciones ligeramente desplazadas. Cuando la configuración relativa de dos cámaras difiere únicamente por un desplazamiento en dirección horizontal, la configuración se denomina paralela y la disparidad es un valor unidimensional a lo largo de la dirección horizontal. **La disparidad es función de la distancia de un punto 3D a las cámaras** al igual que los parámetros de la geometría epipolar, como la línea de base y la distancia focal. La figura 4.6 muestra la relación de la profundidad con la disparidad. Por una simple semejanza de triángulos, la conocida fórmula de la disparidad estéreo se obtiene como sigue:

$$Z = \frac{b \cdot f}{d} = \frac{b \cdot f}{x_l - x_r} \tag{4.6}$$

donde b es la distancia entre dos cámaras, denominada línea de base o *baseline* y f es la distancia focal.

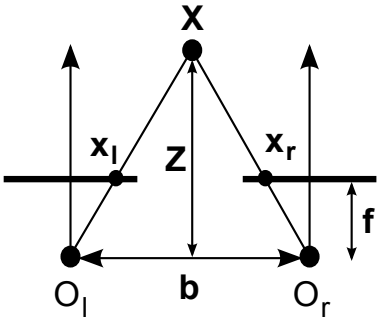


FIGURA 4.6 – **Reconstrucción del valor de profundidad.**

Puesto que la línea de base y la distancia focal son parámetros establecidos durante la etapa de captura, no pueden ser modificados posteriormente. Por lo tanto, la forma más práctica de adjustar la profundidad percibida podría ser modificar la disparidad mediante desplazamiento y escalado.

4.4 Revisión de implementaciones de algoritmos de correspondencia

Con el fin de hallar una solución para resolver nuestro problema a corto plazo debido a las limitaciones temporales del proyecto *ImmersiveTV*, se ha planteado una estrategia basada en aplicar algoritmos cuya implementación sea accesible para calcular el mapa de disparidad de nuestro contenido y decidir, según los resultados, qué técnica utilizar. Como resultado, se han revisado algunas **implementaciones disponibles** que vamos a describir brevemente. Todos los algoritmos e implementaciones mencionadas a continuación suponen que las imágenes de entrada están rectificadas.

Comenzamos con el conjunto de algoritmos de correspondencia densa accesibles a través del **sitio web de visión estéreo de Middlebury** [23] descrito en [16]. Este grupo de implementaciones, que llamaremos ***stereoMatcher*** incluye algoritmos locales basados en ventana, algoritmos de difusión y métodos de optimización global mediante programación dinámica, enfriamiento simulado, *belief propagation* y *graph cuts*. Está organizado de forma modular, de manera que se pueden realizar combinaciones de las cuatro fases de la correspondencia estéreo permitiendo experimentar diferentes aproximaciones al problema. Dispone de los siguientes módulos:

1. **Funciones de coste:** AD, SD, SAD, SSD, NCC, BT, SGRAD, rank.
2. **Agregación de coste:** ventanas cuadradas, ventanas de tamaño variable, filtro binomial, *box filter*, modelos gaussianos y métodos de difusión(regular, bayesiana).
3. **Optimización:** WTA, DP, SO (optimización de líneas de búsqueda), SA (enfriamiento simulado), GC, BP.
4. **Refinamiento:** interpolación a nivel de subpíxel, posfiltrados.

La siguiente implementación es la **librería de minimización MRF** [20], que es un trabajo posterior realizado por Middlebury. Su código es accesible a través de [147]. La implementación de esta librería incluye varias técnicas de optimización para minimizar un problema de *Markov Random Fields* aplicado a la correspondencia estéreo:

1. ICM (*Iterated Conditional Modes*).
2. *Graph cuts*: algoritmos *max-flow*, α*-expansion* y *swap* [19, 120, 142].
3. *Max Product Belief Propagation* (BP): propagación de mensajes a través de los nodos [18].
4. *Sequential tree-reweighted message passing* (TRW-S): propagación de mensajes a través de una estructura en árbol [140, 148].

Las dos implementaciones descritas anteriormente (*stereoMatcher* y MRF) **no cumplen los requisitos** de nuestra aplicación, ya que en el primer caso son bastante antiguas e incluso obsoletas, y en el segundo caso, al tratarse de **técnicas globales** que suponen un elevado coste computacional, **pueden resultar poco eficientes** y como consecuencia, es posible que no sean capaces de procesar vídeo en tiempo real. Además, están diseñados para imágenes de baja resolución y condiciones de iluminación definidas.

Todo esto nos lleva a buscar implementaciones más eficientes. La librería ***libELAS*** [149], cuya implementación es accesible a través de [150] y nos referiremos a ella como **ELAS**, está diseñada para trabajar sobre imágenes de alta resolución. Optimiza el espacio de búsqueda mediante la triangulación a partir de un conjunto discreto de puntos de control que han sido correspondidos de forma robusta. Para calcular la disparidad del resto de píxeles utiliza un **modelo probabilístico**, reduciendo ambigüedades. Se esperan buenos resultados en las zonas donde hay textura y una tasa en torno a 1 fps.

Una sencilla implementación de *belief propagation* descrita en [144] y accesible en [151], desarrolla el algoritmo ***loopy belief propagation*** explicado en la sección 4.2.5.2. La computación de los mensajes está optimizada ya que se actualizan siguiendo un patrón de tipo *checkerboard*. También se recoge la versión multiescala de *belief propagation*. Es determinante la elección adecuada del número de iteraciones (n_{iter}), que determina el alcance de los mensajes entre nodos. También es capaz de calcular los resultados a una tasa de 1 fps, dando lugar a un mapa ruidoso en el que la suavidad depende del número de iteraciones y del número de niveles (n_{levels}). En este último caso la suavidad del mapa se debe a la pérdida de detalle en la pirámide multirresolución.

La librería **OpenCV** [67] dispone de varias implementaciones basadas en GPU [146], lo cual resulta apropiado para los contenidos de vídeo en tiempo real, además de implementaciones más modernas de algunos algoritmos ya comentados.

- **Implementaciones basadas en GPU:** están desarrolladas en CUDA [152–154].
  - ***Block Matching* (BM_GPU):** se basa en [136] y utiliza la estrategia WTA para minimizar la función de coste calculada como suma de diferencias al cuadrado (**SSD**) agregada sobre ventanas de tamaño fijo.
  - ***Belief Propagation* (BP_GPU):** implementación de *Loopy Belief propagation* multirresolución [144] optimizada para GPU.

- ***Constant Space Belief Propagation* (CSBP):** descrito en [155], se basa en el anterior algoritmo de *belief propagation* iterativo, que va reduciendo jerárquicamente el rango de búsqueda de disparidad de manera que para resolver el problema se necesita un espacio de memoria constante, a diferencia del anterior método, que requiere una cantidad de memoria que se incrementa en cada iteración. Fijando los niveles posibles de disparidad en la resolución original, resuelve el problema en tiempo linealmente proporcional al número de píxeles de la imagen. En otras palabras, evita información redundante en el intercambio de los mensajes.

- **Implementaciones basadas en CPU:** desarrolladas en C++ con las funciones propias de la librería de OpenCV.

  - ***Block Matching* (BM_CPU)** de nuevo, basada en [136], utiliza la estrategia WTA para minimizar la función de coste calculada como suma de diferencias absolutas (**SAD**) utilizando ventanas de tamaño fijo.

  - ***Semiglobal Block Matching* (SGBM):** implementa una ligera modificación del algoritmo descrito en [132], utilizando agregación de coste basada en ventanas en vez de comparar píxeles individuales y la penalización por discontinuidad (*discontinuity cost*) se expresa como una función global. No implementa la función de coste basada en información mutua, sino que utiliza la métrica de Birchfield-Tomasi de precisión sub-píxel. Incluye un prefiltrado y post-procesado definido por la restricción de unicidad, realizando interpolación cuadrática y filtrado de puntos aislados.

  - ***Variational Matching* (StereoVAR):** implementación experimental del algoritmo descrito en [156] que combina herramientas como la regularización y el procesado multiescala para estimar la profundidad, preservando las discontinuidades.

  - ***Graph Cuts* (GC)** [157]: minimiza la función de energía de complejidad NP utilizando un algoritmo de graph cuts que computa un mínimo local. Preserva las discontinuidades y comprueba la unicidad. Se considera una implementación obsoleta.

La mayoría de los algoritmos anteriores tienen un parámetro que define el número de disparidades (n_{disp}), sin embargo, salvo excepciones, no permiten definir el valor mínimo y máximo del rango de disparidad (d_{min} y d_{max}). Esto se debe a que están diseñados para aplicarse sobre imágenes creadas específicamente para los experimentos de laboratorio, en las que se garantiza el cumplimiento de los requisitos principales (configuración paralela, imágenes rectificadas), con un rango de disparidad bien controlado. Sabemos que el signo de la disparidad depende de la posición del objeto respecto al plano de imagen y según nuestro convenio, será negativo cuando un objeto se sitúa por delante. El principal problema de las escenas reales es la falta de control sobre estas restricciones, produciéndose **situaciones de paralaje negativo y positivo dentro de la misma imagen**. Por lo tanto, llegamos a la conclusión de que es necesario considerar casos en los que la **disparidad sea negativa**.

4.5 Revisión de evaluaciones de algoritmos de correspondencia densa

Para encontrar un criterio de selección del algoritmo de correspondencia estéreo que proporcione los mejores resultados en nuestro caso particular, es necesario analizar los algoritmos existentes. Debido a la relevancia y el amplio número de aplicaciones en este campo, a lo largo de los años se han realizado muchos trabajos de comparación y evaluación. Se puede distinguir entre trabajos dedicados a medir el **rendimiento de un algoritmo** de correspondencia, **comparación** entre diferentes **técnicas**, evaluaciones de una o varias **fases de correspondencia** (coste, agregación, optimización y refinamiento, sección 4.2.3) y definición o aplicación de **métricas de confianza**, entre otros. Algunos de ellos han dado lugar a entornos de evaluación como ***Middlebury Stereo Vision Page*** [23], descrito en [16, 17, 39, 158], ***The KITTI Vision Benchmark Suite*** [24, 25, 159], o sitios web de proyectos que proporcionan referencias y contenidos diseñados específicamente para la evaluación de algoritmos, como ***.enpeda.. Image Sequence Analysis Test Site (EISATS)*** [26]. Sin embargo, cada uno de estos trabajos fue concebido con un propósito específico, y por lo tanto fueron desarrollados para diferentes escenarios de aplicación usando distintos tipos de contenidos de entrada y atendiendo a criterios de evaluación particulares. Como consecuencia, están generalmente enfocados a los algoritmos o la fase concreta del proceso de correspondencia aplicados sobre sus datos específicos.

Inicialmente, el **problema de correspondencia estéreo era tratado de forma paralela al de predicción de movimiento**, ya que se puede considerar que cada vista de una imagen estéreo es un desplazamiento horizontal de la otra vista de igual manera que un cuadro de vídeo es el resultado de aplicar un desplazamiento al cuadro anterior. Una de las principales referencias es el trabajo de **Szeliski [29]**, donde se utiliza el **error de predicción como medida de calidad** de los resultados. Para ello, partiendo de múltiples vistas (o cuadros) de una misma escena (o secuencia) sintética, se haya el mapa de disparidad (o error de predicción) a partir de dos de ellas. El siguiente paso es interpolar una tercera vista que no ha sido utilizada para calcular el mapa y comparar el resultado con la tercera vista original. Hay que destacar que la comparación se realiza **sin *ground-truth***, midiendo el error **RMS** entre las vistas original y reconstruida, además de la **desviación estándar**. Adicionalmente, se contabiliza el **porcentaje de *outliers*** o píxeles de la imagen de error que superan 3 veces la desviación estándar, que son causados por oclusiones, distorsión radiométrica y errores residuales de movimiento. En un trabajo posterior [160], Szeliski describe la elaboración manual de *ground-truth* para mejorar la comparación de resultados y diferencia entre regiones sin textura y ocluidas.

A continuación, Scharstein y Szeliski establecieron una clasificación o **taxonomía de los algoritmos de correspondencia [16]** definiendo las cuatro fases principales que componen la mayoría de algoritmos y puede ser considerado la referencia principal para la evaluación de **imágenes estáticas**, debido a que la **base de datos elaborada para realizar los experimentos** han sido muy utilizadas después. Se trata de **escenas de laboratorio** formadas por un conjunto de vistas en las que podemos ver **objetos reales** distribuidos de la mejor manera posible para los experimentos (predominando planos frontales), o **imágenes sintéticas** generadas para un propósito específico. En trabajos posteriores [17, 91, 158], **nuevas imágenes realistas** con más zonas sin textura, con objetos geométricamente más complejos y teniendo en cuenta variaciones de iluminación, fueron añadidas a la base de datos **completando el sitio web de Middlebury** [23], donde se recoge el contenido de los experimentos. Los experimentos realizados en [16] comparan los resultados de aplicar diversos métodos estéreo sobre dos vistas alineadas:

- **Experimentos sobre el coste:** aplicando diversas funciones de coste en algoritmos locales (con diferentes métodos de agregación) y sobre algoritmos globales.

- **Experimentos sobre la agregación:** comparando ventanas cuadradas, desplazables, filtros iterativos y difusión.
- **Experimentos sobre optimización:** analizando el efecto de asignar distintos pesos al término *discontinuity cost*.
- **Experimentos sobre refinamiento subpíxel:** analizando los efectos del muestreo de la disparidad.
- **Rendimiento:** analizando el tiempo de ejecución de los algoritmos.

La comparación de resultados de los experimentos se realiza de la siguiente manera:

- **Conocido el *ground-truth*** se calculan estadísticas de error sobre la **imagen completa** y también diferenciando entre **zonas con y sin textura**, regiones **ocluidas** y píxeles en torno a **discontinuidades**. Las estadísticas son:
  - **Error cuadrático medio** entre el mapa calculado y el *ground-truth*.
  - **Porcentaje de píxeles mal correspondidos:** superan un umbral de error.
- **Sin *ground-truth*** toman la idea de Szeliski [29] y utilizan la interpolación o predicción de vistas.
  - ***Forward warp*: interpolar una vista que no ha sido utilizada** para obtener el mapa de disparidad a partir de la vista de referencia (utilizada para extraer el mapa) y el mapa calculado.
  - ***Inverse warp*: reconstruir la imagen de referencia** (utilizada para obtener el mapa) a partir del mapa de disparidad y una tercera vista no utilizada en el proceso, con el fin de calcular el error de predicción.

Otra forma de analizar los resultados es verificar la validez de los valores de disparidad asignados a cada píxel a través de diversas **medidas de validez**, que no requieren *ground-truth*. El trabajo de **Banks [28]** compara los resultados de aplicar varias medidas de semejanza (SAD, ZSAD, SSD, ZDSSD, NCC, ZNCC, Census, Rank) sobre **imágenes exteriores** para analizar la **sensibilidad frente al ruido** e **inmunidad a la distorsión radiométrica**. Para ello, introduce un conjunto de medidas de validez comparando su capacidad de eliminar correspondencias falsas o establecer una magnitud de confianza:

- **Inspección visual**: interpretación de resultados observando características de la imagen original.
- **Consistencia izquierda-derecha** (LRC): requiere dos mapas de disparidad, uno para la vista izquierda y otro para la derecha, y se realiza una **comprobación cruzada** (*cross-check*) para verificar la validez de los mapas estimados.
- Identificación de **disparidades localmente anómalas** comparando con los valores de los píxeles vecinos.
- Identificación de **zonas sin textura**.
- Identificación de **correspondencias aisladas**.
- Comparación de los valores de **coste de correspondencia** (MSM) obtenidos de la medida de semejanza.
- Identificación de **correspondencias ambiguas** debido a la existencia de candidatos con el mismo coste.

A lo largo de trabajos posteriores [161–163] se ha ido definiendo nuevas medidas de confianza de los resultados obtenidos sobre **imágenes interiores y exteriores**, y se ha realizado la siguiente clasificación de métricas basadas en:

- La medida del **coste de correspondencia**: MSM.
- **Propiedades locales de la curva de coste** en torno al mínimo: CUR.
- **Mínimos locales de la curva de coste**: PKR, PKRN, MMN, NLM.
- **Curva de coste completa**: PRB, MLM, AML, NEM, NOI, WMN, WMNN.
- **Consistencia entre los mapas de disparidad izquierdo y derecho**: LRC, LRD.
- **Rasgos distintivos**: DTS, DSM, SAMM.

Posteriormente se han publicado bases de datos de **vídeo** para evaluaciones de *stereo matching* en el contexto específico de **conducción por autopista** [25, 159, 164, 165]. Puesto que son **imágenes reales en un entorno exterior**, se producen ricas variaciones de exposición, luz y brillo, tanto como zonas con poca, ambigua o ninguna textura. Las secuencias de vídeo de conducción se aproximan bastante a las características de nuestro contenido deportivo, con la pequeña diferencia de que la grabación es continua y no existen cambios (saltos) de escena o punto de vista. El **entorno de pruebas de KITTI** [24, 25, 159] permite evaluar los **resultados** obtenidos al aplicar algoritmos de correspondencia **sobre los vídeos** de su base de datos **realista** mediante el cálculo de diversas estadísticas, comparándolos con el ***ground-truth***:

- **Out-Noc:** porcentaje de píxeles erróneos (cuyo error supera un umbral) en áreas no ocluidas (visibles en ambas vistas).
- **Out-All:** porcentaje de píxeles erróneos en la imagen completa.
- **Avg-Noc:** disparidad media en áreas no ocluidas.
- **Avg-All:** disparidad media en la imagen completa.
- **Density:** porcentaje de píxeles para los cuales se dispone de *ground-truth*

La base de datos del proyecto **.enpeda.. [26]** está compuesta por **secuencias largas de vídeo real de alta resolución** con escenas de conducción por autopista bajo diferentes condiciones como la lluvia, la noche o movimiento muy rápido. Existen numerosas publicaciones que utilizan esta base de datos para evaluar el comportamiento de algoritmos de correspondencia aplicados sobre este tipo de contenido [30, 164–166]. El trabajo de Morales [165], que posteriormente completó en [30], vuelve a recurrir al uso del **mapa de disparidad como error de predicción** para interpolar una tercera vista, aunque hay que comentar que esta técnica no se ha vuelto a utilizar con secuencias estéreo del mundo real ya que habitualmente contamos con sólo dos vistas. Comparando la vista reconstruida con la original, se realizan las siguientes **medidas de calidad**:

- **RMS en las regiones no ocluidas**.
- **NCC en las regiones no ocluidas**.

De todo este conjunto de trabajos, Klette [166] introduce **nuevas medidas de error** para evaluar la estabilidad y robustez de los algoritmos, sin embargo sólo son aplicables en el caso de disponer de *ground-truth*. Además, proponen de nuevo el uso del error de predicción sin *ground-truth*.

Un aspecto importante a tener en cuenta en las aplicaciones de vídeo es el **tiempo de proceso**, suponiendo un reto minimizarlo para maximizar la **eficiencia**. Se han desarrollado multitud de evaluaciones de **algoritmos de correspondencia en tiempo real** [22, 146, 167–169] analizando los parámetros y fases del proceso que afectan más a la eficiencia con el fin de optimizar las implementaciones. Todos estos trabajos se centran en **implementaciones basadas en GPU** que son comparadas mediante el número de **cuadros por segundo** que son capaces de procesar o los recursos computacionales (memoria) que consumen. Como principal ejemplo, Kalarot [22] evalúa el rendimiento de varias implementaciones de algoritmos en CUDA obteniendo las siguientes estadísticas:

- **Tasa de cuadros por segundo:** depende del número de cálculos necesarios y eficiencia de mapeo de la memoria a la GPU.
- **Rendimiento o *throughput*:** número de disparidades calculadas por unidad de tiempo. Depende de la capacidad de paralelización, memoria disponible y niveles de disparidad.
- **Escalabilidad y límites de configuración:** determina la resolución máxima de las imágenes y el máximo número de niveles de disparidad.
- **Precisión o *accuracy*:** Es inversamente proporcional a la velocidad. Se mide calculando el RMS con respecto al *ground-truth* y el porcentaje de correspondencias incorrectas.

Como hemos visto, la mayoría de evaluaciones disponen de ***ground-truth* (GT)** con el que comparar sus resultados. Sin embargo, nosotros no nos encontramos en esa situación, teniendo que prestar mayor atención a las técnicas más relevantes utilizadas para evaluar sin GT. A parte de la **inspección visual**, la técnica más empleada consiste en el uso del **error de predicción**, y puede estar basada en una trasformación directa (*forward*) o inversa (*inverse warping*) para interpolar una tercera vista o reconstruir la de referencia. Por lo general, este planteamiento del error de predicción se utiliza en el caso de contenido multivista. En cuanto a la comparación del mapa de disparidad calculado con el GT o de la imagen reconstruida con respecto a la original, se utilizan medidas de error típicas como el **error cuadrático medio** o la **raíz del error cuadrático medio** y se extrae el **porcentaje de píxeles mal correspondidos**, apoyándose en una tolerancia de error. Además, se pueden realizar estas medidas restringiéndose a las regiones con alta probabilidad de ser problemáticas, como las **zonas sin textura** o las **regiones ocluidas** y cercanas a **bordes** de objetos. Finalmente, lo más habitual para comparar la **eficiencia** es medir la tasa de cuadros por segundo que los algoritmos son capaces de procesar.

4.6 Solución propuesta

Esta sección describe la solución adoptada. En primer lugar se comentan los algoritmos utilizados y a continuación se define el criterio de comparación y evaluación.

4.6.1 Preselección de algoritmos para nuestra evaluación

En este apartado justificamos la selección de algoritmos que hemos incluido en nuestra aplicación. Para realizar un análisis que abarque el mayor número posible de tipos de algoritmo y que sea lo suficientemente sencillo como para tomar una decisión correcta aplicada a nuestro problema, hemos escogido algunas representaciones de métodos locales y globales, además de uno semiglobal. Además, tomamos versiones tanto de CPU como de GPU para estudiar el rendimiento.

En primer lugar, como ejemplo más representativo de **métodos locales**, tenemos **BM_CPU** y **BM_GPU**. El primero es el más sencillo de implementar (SAD), resultando bastante rápido (casi tiempo real) debido a la agregación mediante ventanas cuadradas. Sin embargo, la segunda mejora la implementación anterior ya que es más robusta frente a errores al utilizar SSD, además la eficiencia es enormemente más alta debido a su implementación para GPU trabajando en tiempo real y, por último, añade una etapa de posprocesado para eliminar outliers dando lugar a mapas de disparidad más suaves y densos.

El ejemplo por excelencia de **optimización global** es **GC** que, a pesar de requerir un elevado coste computacional para alcanzar una solución convergente y no ser preferible utilizarlo para contenidos en tiempo real, el tipo de resultados que propociona es de gran calidad sobre todo en escenas donde predominan los planos frontales. Es interesante para nuestro análisis porque sirve como referencia para aproximaciones más rápidas. Tal es el caso de *belief propagation* cuyo potencial reside en su manera implícita de modelar la restricción de suavidad. Finalmente hemos tomado **BP_GPU** y para poder realizar comparaciones entre GPU y CPU, añadimos a nuestro entorno la implementación en C++ para CPU facilitada por el autor, y la llamaremos **BPVISION**. Además hemos incluído **CSBP** para comprobar las mejoras frente a los anteriores ya que, según sus autores, es más rápido y obtiene un resultado menos ruidoso.

Por otra parte, la **aproximación** de correspondencia de bloques **semiglobal SGBM** representa un compromiso interesante entre calidad y eficiencia por lo que podemos obtener un resultado cuya calidad sea superior a la de una aproximación local, pero con un coste computacional inferior con respecto a los algoritmos globales, pudiendo alcanzar una tasa razonable para trabajar casi a tiempo real.

Finalmente, como en los vídeos deportivos predominan las características dinámicas, parece interesante aprovechar algoritmos en los que los parámetros puedan adaptarse automáticamente al contenido y autoajustarse en función de los resultados sobre los cuadros anteriores. Por el momento, los algoritmos variacionales como **StereoVAR** no son tan comunes pero tienen un gran potencial. Otra implementación interesante por su diseño para trabajar sobre imágenes reales de alta resolución es **ELAS**.

Como resumen, la tabla 4.3 recoge los algoritmos considerados en la aplicación, clasificados por tipo, indicando si se ejecutan sobre CPU o GPU, además de las fases que los componen.

Algoritmo	**Proc.**	**Coste**	**Agregación**	**Optimiza.**	**Refinam.**
Métodos locales					
BM_CPU [67, 136]	CPU	SAD	Ventana Cuadrada	WTA	Unicidad, textura
BM_GPU [67, 136]	GPU	SSD	Ventana Cuadrada	WTA	Textura
Métodos globales					
GC [157]	CPU	SD	-	Graph Cut	Unicidad
BPVISION [144]	CPU	SAD + Penal. lineal	-	Belief Vector	-
BP_GPU [67, 144]	GPU	SAD + Penal. lineal	-	Belief Vector	-
CSBP [155]	GPU	Data cost Disc. cost	-	Belief Vector jerárquico	-
Métodos mixtos					
SGBM [132]	CPU	BT	Ventana Cuadrada	DP	Unicidad, Interp. Cuadrática, filtrado puntos
ELAS [149]	CPU	-	Triangulación a partir de puntos de control	MAP probabilístico	Textura, unicidad
Métodos variacionales					
StereoVAR [156]	CPU	Data term, Smoothness term	Difusión	Regularización	-

Tabla 4.3 – **Algoritmos preseleccionados para nuestra evaluación.**

4.6.2 Metodología de evaluación de los resultados

Además de utilizar la **inspección visual** para comprobar el resultado del proceso de correspondencia, es necesario calcular de forma objetiva el error obtenido. El principal inconveniente que se presenta es la ausencia de *ground-truth* que sirva como referencia para comprobar la calidad y validez de la correspondencia estimada. Por lo tanto, es necesario recurrir a medidas de error indirectas.

Partimos de la idea de **utilizar vistas reconstruidas a partir del mapa de disparidad** computado [28–30, 160], ya que de cierta manera, el mapa de disparidad representa un error de predicción entre vistas. La diferencia con respecto a estos trabajos es que en nuestro caso sólo disponemos de dos vistas de la misma escena, por lo que no es viable reconstruir otros puntos de vista. Por lo tanto, nos centramos en la transformación inversa (*inverse warping*) o **reconstrucción de la imagen de referencia**. A partir de la vista de comparación (derecha) (I_R) y del mapa de disparidad calculado, se puede obtener una predicción de la imagen de referencia (izquierda) (I_L) (figura 4.7). Sea $\tilde{I}_L$ la imagen de referencia reconstruida, de dimensiones M×N, la comparamos con la de referencia I_L a través **dos medidas de error** diferentes.

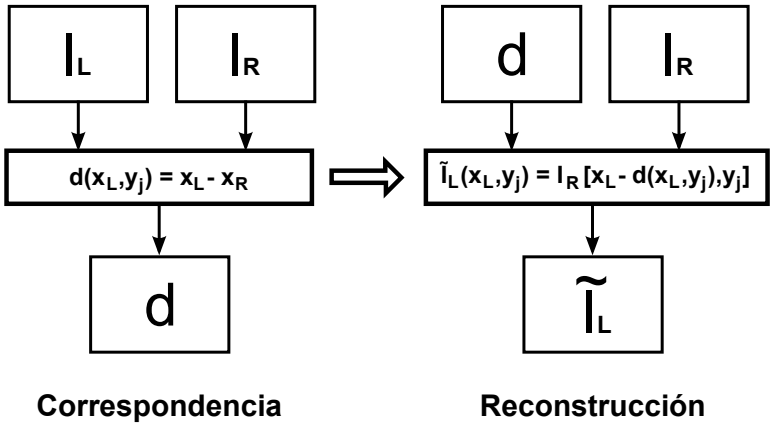


FIGURA 4.7 – **Reconstrucción o interpolación de la imagen de referencia** a partir del mapa de disparidad calculado y la imagen de comparación (derecha).

La primera medida es la **raíz cuadrada del error cuadrático medio (RMS)** y viene dada por las expresiones siguientes:

$$\begin{aligned} RMS &= \sqrt{MSE}, \\ MSE &= \frac{1}{M \cdot N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left| \tilde{I}_L(x_i, y_j) - I_L(x_i, y_j) \right|^2 \end{aligned} \tag{4.7}$$

donde la segunda ecuación representa el **error cuadrático medio (MSE)**. Puesto que el RMS depende directamente del MSE, tiene que ser bajo para considerar que el resultado es bueno, por lo tanto se trata de **minimizar este error**.

La segunda medida de error es la **relación pico señal a ruido (PSNR)**, dada por:

$$PSNR = 10 \log_{10} \left(\frac{I_{max}^2}{MSE} \right) \tag{4.8}$$

donde se observa que está inversamente relacionada con el MSE, por lo tanto, el resultado será **mejor cuanto mayor sea la PSNR**.

Para reforzar la comparación de los resultados, se calcula la **imagen de error normalizada** a partir de la diferencia absoluta entre la imagen de predicción $\tilde{I}_L$ y la de referencia I_L, de manera las regiones donde mayores errores se producen en la estimación de disparidad sean visualmente apreciables.

$$I_{dif} = \frac{\left| \tilde{I}_L - I_L \right|}{\text{máx} \left| \tilde{I}_L - I_L \right| - \text{mín} \left| \tilde{I}_L - I_L \right|} \times 255 \tag{4.9}$$

Con todos estos datos, se puede calcular dos estadísticas más: la **tasa de píxeles inválidos** (P_i) y la **tasa de píxeles erróneos** (P_e). Un píxel se define como inválido cuando el algoritmo de correspondencia estéreo devuelve un valor no definido. Esto ocurre, sobre todo, en zonas ocluidas. Por otra parte, la tasa de píxeles erróneos de la imagen de error se define como el porcentaje de píxeles que cumplen la condición $\left| \tilde{I}(x_i, y_j) - I(x_i, y_j) \right| > th$, donce th es un umbral que representa el máximo error permitido. La suma de ambos porcentajes, da lugar a la **tasa de píxeles incorrectos**.

$$P_{err} = P_i + P_e = \frac{invalidos}{M \cdot N} \times 100 + \frac{erroneos}{M \cdot N} \times 100 \tag{4.10}$$

Para comprobar la eficiencia de los algoritmos, se mide la **tasa de cuadros por segundo (fps)** a la que trabaja cada método. Para lograr una tasa aproximada al tiempo real, es necesario alcanzar unos 25 fps.

Capítulo 5

Implementación y funcionalidades del sistema

5.1 Introducción

Para llevar a cabo la evaluación de los algoritmos de correspondencia preseleccionados en la sección 4.6.1, se ha desarrollado una aplicación en ***Visual C++*** utilizando el entorno ***Microsoft™ Visual Studio 2010*** [170], haciendo uso de la librería de visión artificial **OpenCV (versión 2.4.0)** [171] y de **Qt (versión 4.8.0)** [172, 173] para el desarrollo de la interfaz gráfica de usuario (GUI).

La herramienta desarrollada permite tomar como dato de entrada una **imagen**, una **secuencia** de imágenes contenidas en un directorio y **vídeos estereoscópicos**, que pueden estar en formato *side-by-side* o tratarse de dos vistas separadas. Se ha creado un entorno que permite seleccionar los valores de los **parámetros de entrada comunes** a la mayoría de los algoritmos de correspondencia habituales, como puede ser la disparidad mínima y el número de niveles de disparidad, que determinan el rango de disparidad, o el tamaño de ventana de agregación de los algoritmos locales, el número de iteraciones de los algoritmos iterativos y los niveles jerárquicos de los algoritmos multirresolución. Un entorno común permite **evaluar** el rendimiento y calidad de todos los algoritmos bajo las mismas condiciones. Además, la herramienta ofrece la posibilidad de **guardar los resultados** en un formato que facilita su comparación, por medio de un archivo de texto que acompaña a las imágenes de salida. En la figura 5.1 se representa un esquema de las entradas y salidas de la herramienta.

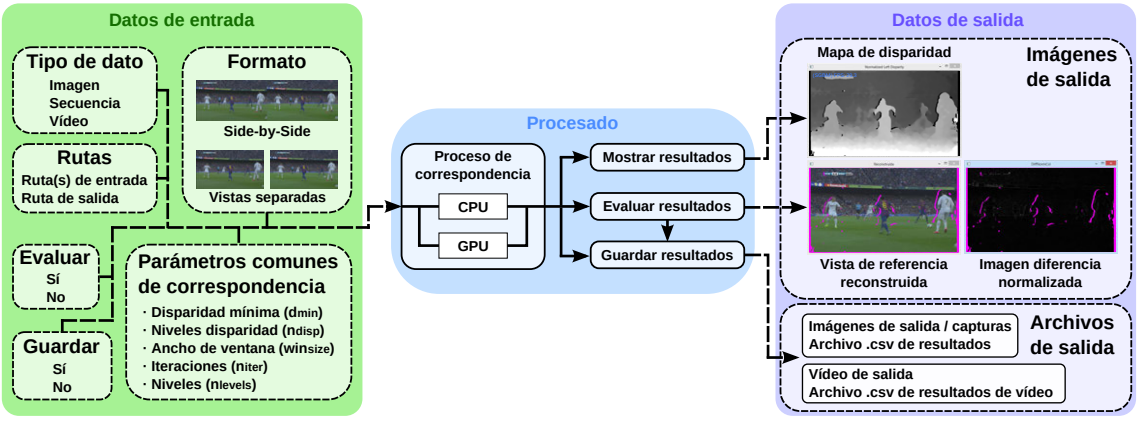


FIGURA 5.1 – **Esquema de entradas y salidas de la herramienta.**

En cuanto a la implementación, como detallamos más adelante, ha dado lugar a ciertas modificaciones sobre la librería basada en GPU de OpenCV. Con el objetivo de optimizar la ejecución, esta librería utiliza la arquitectura **CUDA** (utilizamos la versión 4.2) [152–154] propuesta por *NVIDIA Corporation* para paralelizar los hilos de ejecución cuando se procesan las imágenes, incrementando considerablemente el rendimiento de los algoritmos. Por lo tanto, la librería `cv::gpu` ha sido modificada dando lugar al conjunto de funciones bajo el espacio de nombres `cv_bis::gpu` para preservar las clases y funciones originales y poder experimentar con valores de disparidad negativos. Todo este proceso ha sido posible gracias a que el lenguaje de programación CUDA está basado en C++. Además de la librería GPU de OpenCV, también se han tenido que realizar adaptaciones y modificaciones sobre otros algoritmos de correspondencia para poder integrarlos dentro de un entorno común y, en algún caso, para que soporten trabajar con valores negativos de disparidad.

En este capítulo explicamos detalladamente la herramienta desarrollada haciendo un recorrido a lo largo de las funcionalidades de las que dispone el sistema de estimación de mapas de profundidad. Comenzamos enumerando los parámetros y variables de la aplicación y a continuación explicamos las modificaciones realizadas sobre los algoritmos de correspondencia. Para finalizar el capítulo, describimos el aspecto visual que presenta la herramienta.

5.2 Parámetros y variables de la aplicación

Comenzamos describiendo brevemente los principales parámetros y variables que contiene nuestro programa. Nos referimos como **parámetro** a cualquier variable que el usuario puede proporcionar o modificar durante la ejecución, mientras que llamamos **variable** a cualquier valor que el sistema utiliza internamente.

Los **parámetros de la aplicación** se definen como una estructura (`struct Params`) que los reúne de forma compacta. A continuación realizamos una breve descripción de los parámetros, agrupados según la relación que tienen con la aplicación:

- **Tipo de entrada**

  `type`**:** tipo enumerado que puede tomar los valores `IMAGE`, `SEQUENCE` o `VIDEO`.

  `bool sidebyside`**:** *flag* que indica si la entrada está formada por dos vistas separadas o por una en formato *side-by-side*.

- **Archivos de entrada**

  `string src`**:** ruta de la imagen o vídeo, o directorio de la secuencia en SbS.

  `string left_param`**:** ruta de la imagen o vídeo, o directorio de la secuencia de la vista izquierda.

  `string right_param`**:** ruta de la imagen o vídeo, o directorio de la secuencia de la vista derecha.

- **Algoritmos de correspondencia**

  `method`**:** tipo enumerado que recoge los métodos de correspondencia (`BM_CPU`, `BM`, `GC`, `BPVISION`, `BP`, `CSBP`, `SGBM`, `ELAS` y `VAR`).

  `int dmin`**:** valor mínimo de disparidad (d_{min}). Puede ser negativo.

  `int ndisp`**:** niveles de disparidad (n_{disp}), determina el rango de disparidad.

  `int winSize`**:** ancho de ventana (win_{size}) de los algoritmos locales.

  `int iterations`**:** número de iteraciones (n_{iter}) de los algoritmos iterativos.

  `int levels`**:** número de niveles (n_{levels}) de los algoritmos multirresolución.

- **Evaluar y guardar**

  `bool eval`: *flag* que indica si se evalúan o no los resultados.

  `bool guardar`: *flag* que indica si se guardan o no los resultados.

  `string output_dir`: directorio donde se guardan los resultados.

La aplicación está definida como una clase llamada *StereoGUI* que hereda de la clase *QMainWindow* de Qt, la cual proporciona todas las herramientas para trabajar sobre una interfaz gráfica basada en ventanas (`class StereoGUI : public QMainWindow`). A continuación describimos las **variables** definidas en esta clase y que permitirán la correcta ejecución del programa:

- **Parámetros**

  `Params p`**:** estructura que contiene los parámetros descritos anteriormente.

- **Datos de entrada**

  `string path`**:** directorio de trabajo.

  `string nombreFichero`**:** nombre del archivo(s) de entrada.

  `string ext`**:** extensión del archivo(s) de entrada.

- **Imágenes de entrada**

  `cv::VideoCapture video`**:** captura de vídeo si el archivo es un vídeo en SbS.

  `cv::VideoCapture left_video`**:** captura de vídeo para la vista izquierda.

  `cv::VideoCapture right_video`**:** captura de vídeo para la vista derecha.

  `cv::Mat source`**:** matriz de OpenCV que guarda la imagen de entrada SbS.

  `cv::Mat left_src`**:** matriz que almacena la imagen de la vista izquierda.

  `cv::Mat right_src`**:** matriz que almacena la imagen de la vista derecha.

  `cv::Mat left`**:** matriz de la vista izquierda transformada para pasarla como parámetro de entrada a los algoritmos de correspondencia.

  `cv::Mat right`**:** matriz de la vista derecha transformada para pasarla como parámetro de entrada a los algoritmos de correspondencia.

- **Acciones**

  `bool running`**:** *flag* que indica si se está ejecutando la correspondencia.

- **Algoritmos de correspondencia**

  `cv::StereoBM bm_cpu`**:** objeto que representa BM_CPU.

  `cv_bis::gpu::StereoBM_GPU bm`**:** objeto que representa BM_GPU.

  `CvStereoGCState* gc`**:** estructura que representa GC.

  `Bpvision bpVision`**:** objeto que representa BPVISION.

  `cv_bis::gpu::StereoBeliefPropagation bp`**:** objeto BP_GPU.

  `cv_bis::gpu::StereoConstantSpaceBP csbp`**:** objeto que representa CSBP.

  `cv::StereoSGBM sgbm`**:** objeto que representa SGBM.

  `Elas elas`**:** objeto que representa ELAS.

  `StereoVar2 var`**:** objeto que representa StereoVAR.

- **Imágenes de salida**

  `cv::Mat disp_left`**:** matriz de OpenCV que almacena el mapa de disparidad de la vista izquierda.

  `cv::Mat disp_right`**:** matriz de OpenCV que almacena el mapa de disparidad de la vista derecha.

  `cv::Mat disp_left_norm`**:** matriz de OpenCV que almacena el mapa de disparidad normalizado de la vista izquierda, para poder representarlo visualmente como una imagen.

  `cv::Mat disp_right_norm`**:** matriz de OpenCV que almacena el mapa de disparidad normalizado de la vista derecha, para poder representarlo visualmente como una imagen.

- **Evaluación**

  `StereoEval stEval`**:** objeto de la clase *StereoEval* que contiene todos los elementos necesarios para evaluar los resultados. Esta clase se explica en el apartado 5.5.5.

- **Resultados**

  `fstream results`**:** archivo de resultados de capturas individuales.

  `fstream res_video`**:** archivo de resultados de vídeo.

- **Otras variables**

  `int frame_number`**:** indica el cuadro actual dentro de la secuencia o vídeo.

  `int64 work_begin`**:** variable que almacena el instante de comienzo del proceso de correspondencia. Ayuda a calcular la tasa de cuadros por segundo.

Esta sección puede considerarse una referencia de consulta, ya que en los sucesivos apartados explicamos el funcionamiento del programa y aparecen algunas de estas variables representadas en los diagramas.

5.3 Modificaciones sobre los algoritmos de correspondencia

En esta sección vamos a describir las **modificaciones y adaptaciones** realizadas sobre los algoritmos de correspondencia. En la mayoría de los casos, ha sido necesario realizar modificaciones **para soportar valores de disparidad negativos**, y en algunos otros se ha tenido que adaptar la estructura **para integrarlos** adecuadamente **en el mismo entorno**. Los algoritmos **BM_CPU, GC, SGBM y StereoVAR no han sido modificados** ya que, originalmente, son capaces de procesar valores de disparidad negativos y, debido a que pertenecen a la librería OpenCV, son fácilmente integrables.

Comenzamos por los algoritmos pertenecientes a la **librería basada en GPU de OpenCV** que se encuentran bajo el mismo espacio de nombres y, como consecuencia, pueden ser utilizados fácilmente dentro de un mismo entorno. Los algoritmos **BM_GPU, BP_GPU y CSBP** pertenecen al espacio de nombres `cv::gpu` y ninguno de ellos admite valores de disparidad negativa. Para realizar modificaciones sobre estas implementaciones y dejar intacto el código original, se ha creado un nuevo espacio de nombres llamado `cv_bis::gpu`. A continuación enumeramos los principales cambios realizados sobre cada uno de los algoritmos:

- **Modificaciones realizadas sobre BM_GPU**
  - Añadimos un nuevo atributo `int dmin`, que representará el valor mínimo de disparidad, a la clase `StereoBM_GPU` donde se define el algoritmo. Como consecuencia, se debe añadir al constructor para establecer su valor:

```
StereoBM\_GPU (int preset, int dmin = DEFAULT_DMIN, int ndisparities = DEFAULT_NDISP, int winSize = DEFAULT_WINSZ);
```

  - Se debe añadir el parámetro `int dmin` a todas las funciones que se llaman a través operador de la clase `cv_bis::gpu::StereoBM_GPU`, que actúa como envoltorio para acceder a las funciones implementadas en CUDA.
  - Originalmente, la imagen de disparidad se inicializa a 0. Esto no es muy útil, por lo que se sustituye la **inicialización** para que el valor por defecto sea un **valor de disparidad inválido** correspondiente a `dmin-1`.

```
cudaSafeCall(cudaMemset2D(disp.data, disp.step, dmin-1, disp.cols, disp.rows)); // Inicializo la matriz a dmin-1
```

- Los principales cambios se han realizado sobre el núcleo (*kernel*) de CUDA, que implementa el algoritmo de correspondencia. Es necesario **modificar los límites de los bucles** `for` para que la búsqueda comience a partir de `dmin`. Por otra parte, para que la matriz de disparidad resultante contenga valores negativos, se cambia **el tipo de dato** de `unsigned char` a `signed char` admitiendo valores entre -128 y 127. El código resultante se presenta en la sección del anexo I.1.1, donde las líneas que han sido modificadas tienen un comentario indicando la versión original.

- **Modificaciones realizadas sobre BP_GPU**
  - Análogamente a BM_GPU, se añade el atributo `int dmin` a la clase donde se define el algoritmo, `StereoBeliefPropagation`, así como al resto de funciones que se anidan para acceder a las funciones de CUDA. El constructor de la clase resulta de la siguiente forma:

```
SStereoBeliefPropagation(int dmin, int ndisp, int iters, int levels, float max_data_term, float data_weight,
                         float max_disc_term, float disc_single_jump,int msg_type = CV_32F);
```

  - Como en el caso anterior, se **modifican los límites de los bucles** `for` para comenzar la búsqueda a partir de `dmin`.
  - Como sabemos, las técnicas basadas en *Belief Propagation* **utilizan el denominado *belief vector*** para llevar a cabo la optimización y selección del valor de disparidad. A nivel de implementación, esto se realiza mediante ***arrays* unidimensionales** cuya longitud es igual al número de niveles de disparidad (n_{disp}). En dichos *arrays* se almacena el coste asociado al **valor de disparidad que coincide con la posición del array**, es decir, en la posición 0 del *belief vector* se guarda el coste asociado a un valor de disparidad 0, en la posición 1, el coste de disparidad 1, y así sucesivamente. Como consecuencia de ello, puesto que los índices de un *array* son **mayores o iguales que 0**, no es posible trabajar con valores de disparidad negativos. Por lo tanto, para resolver este problema, **aplicamos un offset d-$\mathbf{d_{min}}$** a los índices del *array* al almacenar los costes en el vector. Así, el coste asociado a d_{min}, se almacena en la posición d_{min}-d_{min}=0, el coste asociado a $d_{min}+1$ se guarda en el índice (d_{min}+1)-d_{min}=1, sucesivamente, hasta llegar al nivel de disparidad máximo $d_{min}+n_{disp}-1$ en la posición (d_{min}+n_{disp}-1)-d_{min}=n_{disp}-1. En el anexo I.1.2 se muestra el código del núcleo para calcular el *data cost* y para calcular la disparidad minimizando el coste.

- **Modificaciones realizadas sobre CSBP**

  · Esta implementación es muy similar a BP_GPU, por lo que experimenta las mismas modificaciones. Se añade el atributo `int dmin` a la clase donde se define el algoritmo, `StereoConstantSpaceBP`, así como al resto de funciones que se anidan para acceder a las funciones de CUDA. El constructor de la clase resulta de la siguiente forma:

```
StereoConstantSpaceBP(int dmin, int ndisp, int iters, int levels, int nr_plane,float max_data_term, float data_weight,
                      float max_disc_term, float disc_single_jump,int min_disp_th = 0,int msg_type = CV_32F);
```

  · Se **modifican los límites de los bucles** `for` para comenzar la búsqueda a partir de `dmin`.

  · Se aplica un **offset d-$\mathbf{d_{min}}$** a los índices del *array* al almacenar los costes en el vector. En la sección del anexo I.1.3 se adjunta el código del núcleo para inicializar los valores de *data cost*.

Las implementaciones de algoritmos que no pertenecen al entorno de OpenCV, como **BPVISION y ELAS**, no están dentro del mismo espacio de nombres y tienen que ser adaptadas para integrarse adecuadamente con el resto. ELAS fue desarrollado aprovechando parte del código implementado en BPVISION originalmente, lo cual facilita la integración de ambos algoritmos. Parte de este código común se encuentra en el archivo `image.h`, que contiene herramientas para leer, crear, guardar y modificar archivos de imagen. Una de las modificaciones realizadas sobre este archivo ha sido añadir una función para **inicializar todos los píxeles de una imagen con un valor por defecto**, y dos funciones para **convertir una imagen en una matriz de OpenCV y viceversa**, tal y como se muestra en el anexo I.2.1.

A continuación resumimos las modificaciones sobre BPVISION y ELAS:

- **Modificaciones realizadas sobre BPVISION**

  · Creamos la clase `Bpvision` para envolver todas las funciones originales, haciéndolas privadas, y definimos la estructura de parámetros añadiendo `int dmin`. El archivo `bpvision.h` mostrado en el anexo I.2.2 contiene la definición de la clase. Además, se adjunta la definición de una de las funciones que permiten determinar el rango de disparidad.

  · Al igual que en los casos anteriores, se modifican los **límites de los bucles** `for` para que la búsqueda comience a partir de `dmin`. En el anexo se muestra la función que minimiza el *belief vector*.

  · De la misma manera que BP_GPU y CSBP, se aplica el **offset d-$\mathbf{d_{min}}$** a los índices de los *arrays* al almacenar los costes. En el anexo se adjunta el código de la función que estima el *data cost* para cada valor de disparidad y de la función que calcula los mensajes para cada nodo.

- **Modificaciones realizadas sobre ELAS**

  · Se modifica la clase `Elas` para envolver todas las funciones originales, haciéndolas privadas, y añadimos `int dmin` a la estructura de parámetros. El resultado de la clase Elas está en el anexo I.2.3.

  · Originalmente, la imagen de disparidad se inicializa a -10, ya que se asume que cualquier valor de disparidad menor que 0 es inválido. Por lo tanto, para soportar valores negativos, se sustituye la **inicialización** para que el valor por defecto sea un **valor de disparidad inválido** correspondiente a `dmin-1`. En el anexo I.2.3 se presenta el fragmento de código que inicializa la imagen de disparidad.

  · Se modifican las condiciones `if` para que se verifiquen para valores mayores de `dmin-1` en vez de valores mayores o iguales a 0. En el anexo se presenta como ejemplo la función que elimina las correspondencias inconsistentes de los puntos de control.

A modo de resumen, la tabla 5.1 recoge los algoritmos considerados en la aplicación, describiendo las modificaciones realizadas.

Algoritmo	**GPU**	**Implemen.**	**Modificaciones y comentarios**
Métodos locales			
BM_CPU [67, 136]	No	C++ OpenCV	No ha sido modificada. Parámetros: d_{min}, n_{disp} y win_{size}.
BM_GPU [67, 136]	Sí	OpenCV CUDA/C++	Modificación de la clase `cv::gpu::StereoBM_GPU` y de sus funciones para tomar como parámetro d_{min} y poder ajustar el rango de disparidad. Inicialización de la imagen de disparidad a `dmin-1` y cambio de tipo de dato de `unsigned char` a `signed char`. Límites de bucles `for` para comenzar por `dmin`. Parámetros: d_{min}, n_{disp} y win_{size}.
Métodos globales			
GC [157]	No	C++ OpenCV	No ha sido modificada. Parámetros: d_{min}, n_{disp} y n_{iter}.
BPVISION [144]	No	C++	Creación de la clase `Bpvision` para envolver las funciones originales y añadir como parámetro d_{min} para poder ajustar el rango de disparidad. Límites de bucles `for` para comenzar por `dmin`. Offset d-$d_{\min}$ al almacenar costes en los *arrays*. Parámetros: d_{min}, n_{disp}, n_{iter} y n_{levels}.
BP_GPU [67, 144]	Sí	OpenCV CUDA/C++	Modificación de la clase `cv::gpu::StereoBeliefPropagation` y de sus funciones para tomar como parámetro d_{min} y poder ajustar el rango de disparidad. Límites de bucles `for` para comenzar por `dmin`. Offset d-$d_{\min}$ al almacenar costes en los *arrays*. Parámetros: d_{min}, n_{disp}, n_{iter} y n_{levels}.
CSBP [155]	Sí	OpenCV CUDA/C++	Modificación de la clase `cv::gpu::StereoConstantSpaceBP` y de sus funciones para tomar como parámetro d_{min} y poder ajustar el rango de disparidad. Límites de bucles `for` para comenzar por `dmin`. Offset d-$d_{\min}$ al almacenar costes en los *arrays*. Parámetros: d_{min}, n_{disp}, n_{iter} y n_{levels}.
Métodos mixtos			
SGBM [132]	No	C++ OpenCV	No ha sido modificada. Parámetros: d_{min}, n_{disp} y win_{size}.
ELAS [149]	No	C++	Modificación de la clase `Elas` añadiendo el parámetro d_{min} y varias funciones para poder ajustar el rango de disparidad. Inicialización de la imagen de disparidad a `dmin-1`. Condiciones `if` para valores mayores de `dmin-1`. Parámetros: d_{min} y n_{disp}.
Métodos variacionales			
StereoVAR [156]	No	C++ OpenCV	No ha sido modificada. Parámetros: d_{min} y n_{disp}.

TABLA 5.1 – **Modificaciones sobre los algoritmos de correspondencia.**

5.4 La Interfaz Gráfica de Usuario (GUI)

Al arrancar la aplicación, se muestra la interfaz gráfica de usuario, que está implementada con la **librería Qt** [172, 173] y tiene el aspecto de la figura 5.2. Qt es una librería desarrollada por Nokia para crear inferfaces gráficas interactivas multiplataforma. La GUI consiste en una ventana con título "StereoMatching" que permite seleccionar los principales parámetros descritos en la sección 5.2 y se distribuyen en la ventana siguiendo la misma agrupación: tipos de archivo, rutas de archivos de entrada, parámetros de los algoritmos de correspondencia, evaluación y guardado de resultados.

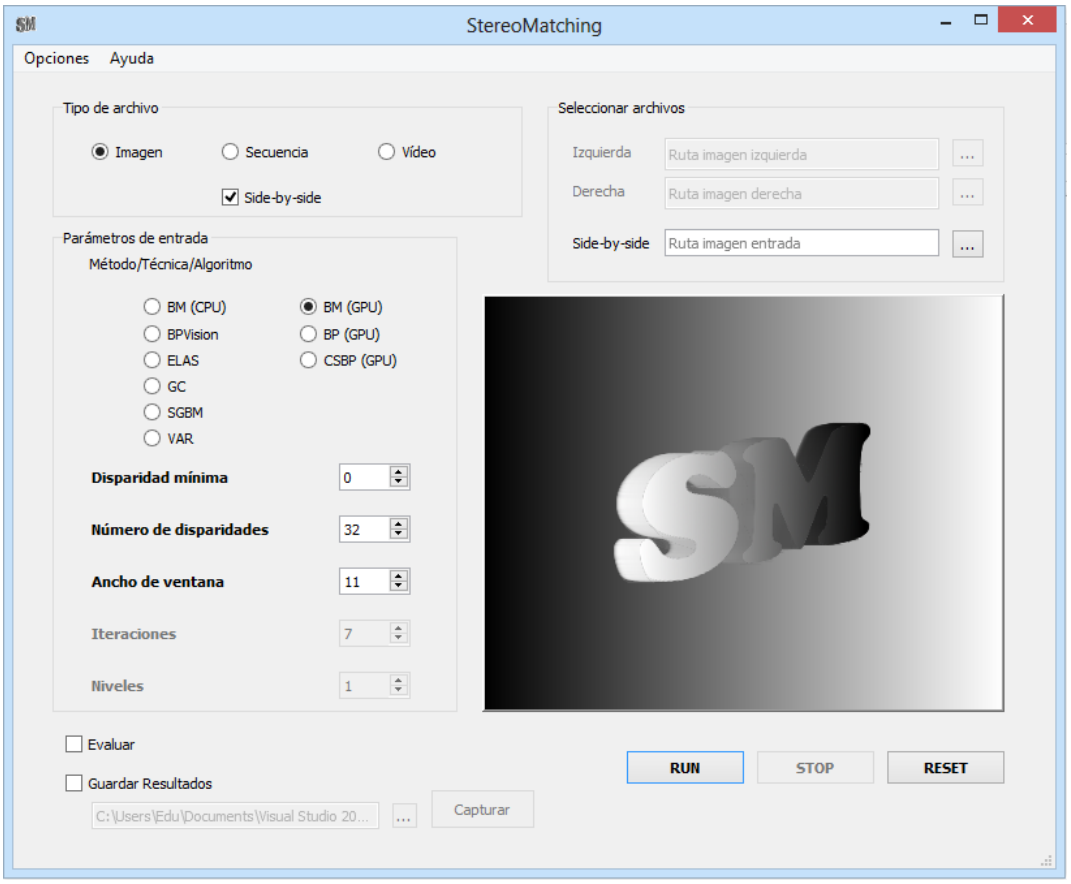


FIGURA 5.2 – **Interfaz de usuario.**

En la esquina superior izquierda de la ventana de la interfaz se muestra el icono diseñado para la herramienta.

En los próximos apartados hacemos un recorrido por cada elemento que forma parte la interfaz describiendo brevemente sus funcionalidades.

5.4.1 Área de selección del tipo de contenido de entrada

En el área superior izquierda de la ventana principal se encuentran las opciones para seleccionar el tipo de archivo de entrada. El tipo de archivo está definido por dos variables de la aplicación: `type` y `sidebyside`. La primera variable determina si el archivo es una imagen, un directorio con imágenes o un vídeo. Un conjunto de botones del tipo `QRadioButton` permite seleccionar una de las tres opciones de forma exclusiva, es decir, al marcar un tipo de archivo, el resto queda deseleccionado. La segunda variable indica el formato de empaquetado del archivo o archivos de entrada, que puede ser *side-by-side* o dos vistas separadas. Un cuadro de selección `QCheckBox` activa el *flag*.

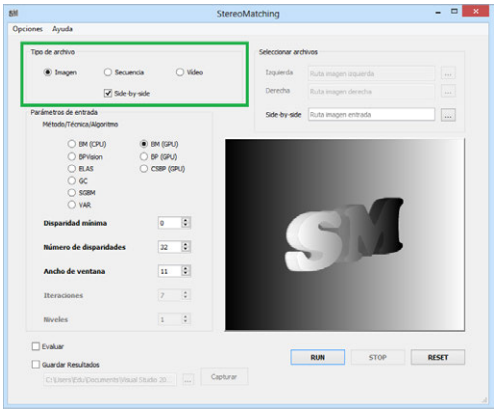


FIGURA 5.3 – **Tipo de entrada.**

5.4.2 Área de selección de la ruta de los archivos

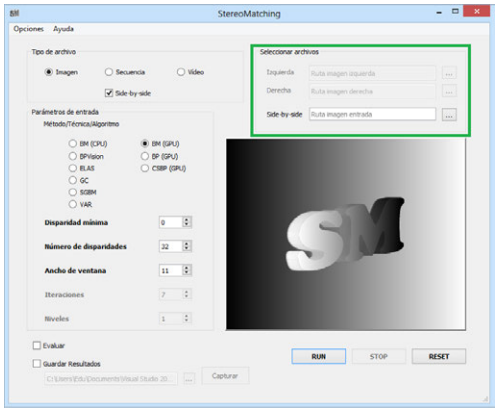


FIGURA 5.4 – **Rutas de entrada.**

El área superior derecha de la ventana principal está reservada para la selección de las rutas de los archivos o directorios de entrada. Se trata de tres cajas de texto de tipo `QLineEdit`, una para la vista izquierda, otra para la vista derecha y otra para la entrada en *side-by-side*. A la derecha de cada línea de texto, un botón `QToolButton` abre un diálogo para seleccionar la ruta del archivo de imagen o vídeo o del directorio de la secuencia. El cuadro de selección del formato de archivo, explicado antes, está asociado a las cajas de texto de manera que si está seleccionado el formato *side-by-side*, se deshabilita la posibilidad de seleccionar rutas separadas para la izquierda y la derecha, mientras que si no está seleccionado, se habilitan las rutas izquierda y derecha y se deshabilita la ruta para *side-by-side*.

5.4.3 Área de selección de parámetros

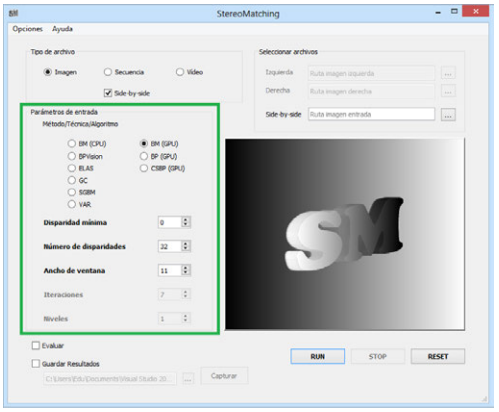
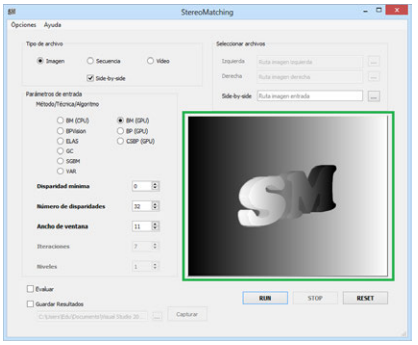


FIGURA 5.5 – **Area de parámetros de correspondencia.**

En la mitad izquierda de la ventana, la parte central presenta los parámetros de correspondencia principales. Estos se corresponden con las variables `p.method`, `p.dmin`, `p.ndisp`, `p.winSize`, `p.iterations` y `p.levels`, que hemos considerado comunes a la mayoría de algoritmos. En la figura 5.2 vemos que aparece seleccionado el método `BM` y, al igual que ocurría con los tipos de archivo de entrada, la selección es exclusiva (sólo se admite un elemento seleccionado). Según el método marcado, se habilitan únicamente los parámetros que utiliza. Por ejemplo, como `BM` no tiene iteraciones o niveles, estos parámetros están deshabilitados. El resto de son numéricos, por lo tanto, lo más apropiado es utilizar objetos de la clase `QSpinBox` que establecen valores numéricos modificables mediante texto o pulsando sobre las flechas superior e inferior.

5.4.4 Área de imagen

Es la zona donde se muestra la vista izquierda durante la ejecución del proceso de correspondencia. La imagen original es escalada para ajustarse al tamaño y la forma del recuadro pudiendo aparecer ligeramente deformada. Cuando no se visualiza contenido sobre el área, se muestra el logo de la herramienta.



(a) Área de imagen.

(b) Logo.

FIGURA 5.6 – **Área de imagen y logo de la herramienta.**

5.4.5 Botones de acción

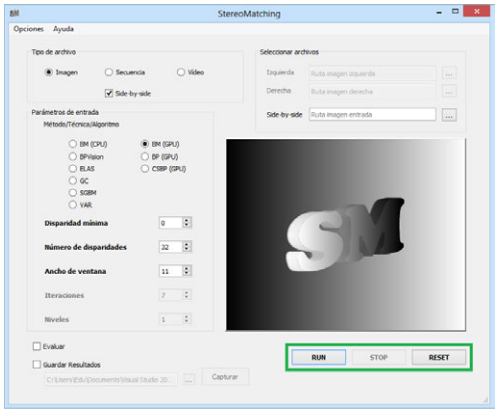


FIGURA 5.7 – **Botones de acción.**

Debajo del área de imagen se sitúan tres botones de tipo `QPushButton`. Cada uno de ellos está asociado a una acción diferente. Una pulsación sobre el botón "RUN" ejecuta la acción `StereoGUI::run()` encargada de iniciar la ejecución. Esta función comprueba el tipo de archivo de entrada seleccionado y decide si se tiene que llamar al subproceso `runImage()`, `runSequence()` o `runVideo()`. El botón "STOP" detiene la ejecución lanzando una interrupción (`StereoGUI::stop()`), conservando los parámetros seleccionados y restaura el área de la imagen dejando de mostrar la vista de entrada. El botón "RESET" restablece los parámetros por defecto (`StereoGUI::reset()`), volviendo de nuevo a la figura 5.2.

5.4.6 Evaluación y captura de resultados

En la esquina inferior izquierda tenemos las opciones relacionadas con la evaluación y guardado de resultados, que representan los *flags* `eval` y `guardar`, respectivamente. De nuevo un cuadro de selección `QCheckBox` permite determinar si se van a evaluar los resultados. Por lo tanto, si se selecciona, se calculará la reconstrucción de la imagen de referencia y la imagen diferencia normalizada junto con los estadísticos, en cada iteración. Otro cuadro de selección `QCheckBox` activa el *flag* de guardado de resultados, es decir, cuando está marcado, se guardarán los resultados. Además, se activa una línea de texto `QLineEdit` asociada donde se debe escribir la ruta de los archivos de salida (`output_dir`) o buscarla utilizando el botón `QToolButton` que abre un diálogo para seleccionar la ruta. Hay que comentar que en el caso de las imágenes y secuencias, al seleccionar el cuadro de guardar los resultados, también se marca el de evaluar. Esto no sucede cuando se trabaja con vídeos porque no tiene mucho sentido

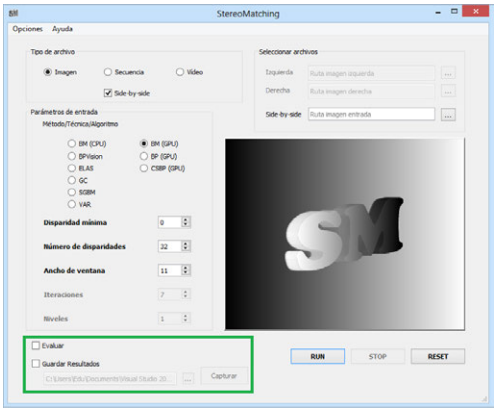


FIGURA 5.8 – **Evaluación y captura de resultados.**

realizar la evaluación cuadro a cuadro y porque interesa obtener los resultados en tiempo real. A la derecha, el botón de tipo `QPushButton` etiquetado como "Capturar" desencadena la acción de realizar una captura individual de resultados (`StereoGUI::capture()`). Es equivalente a pulsar la tecla 'f'. En este caso, si se trabaja con vídeo, se realizará la evaluación de resultados únicamente sobre el *frame* actual para añadirlos al archivo de resultados y almacenar las imágenes reconstruida y de error normalizada.

5.4.7 La barra de menú

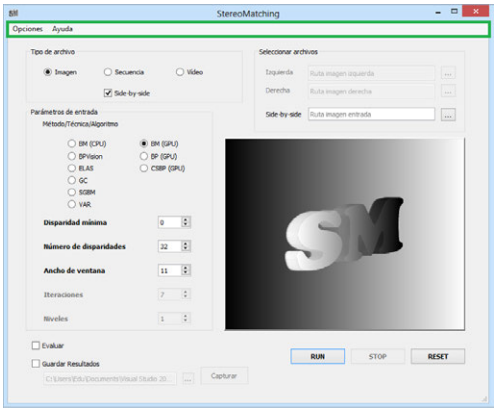
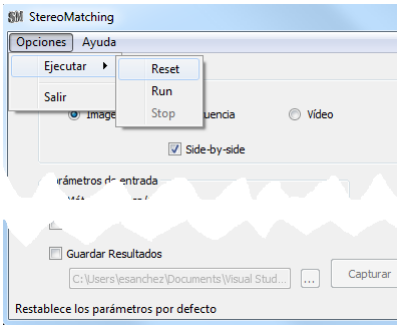
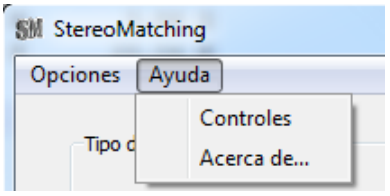


FIGURA 5.9 – **Barra de menú.**

Por último, hablaremos de la barra de menú, situada en la parte superior de la ventana. La clase `QMenuBar` proporciona las herramientas necesarias para elaborar el menú y añadir todas las opciones de forma flexible. La barra de menú consta de dos menús (`QMenu`), uno titulado "Opciones" y otro de "Ayuda". El primero contiene las tres acciones principales de nuestra herramienta ("Reset", "Run" y "Stop") y son equivalentes a pulsar sobre los botones de acción situados bajo el área de imagen. La otra acción es la etiquetada como "Salir", que finaliza la ejecución de la interfaz de usuario cerrando la aplicación. Además, se ha añadido una barra de estado (`QStatusBar`) en la parte inferior de la ventana de manera que, al situar el puntero sobre alguna de las opciones del menú, se muestra una breve descripción de la acción, como se puede observar en la figura 5.10(a). El segundo menú, el de ayuda tiene dos opciones: "Controles" y "Acerca de..." (figura 5.10(b)), que explicamos a continuación.



(a) Opciones



(b) Ayuda

FIGURA 5.10 – **Barra de menú**

Dentro del menú de ayuda, al pulsar la opción de "Controles" se abre un cuadro de diálogo con la información acerca de los comandos del teclado. En la figura 5.11 se presenta una breve descripción de la función de todas las teclas consideradas. Durante la ejecución, es equivalente utilizar los controles del teclado y los parámetros de la interfaz gráfica. Es decir, cualquier modificación sobre los parámetros causada por la pulsación de una tecla se actualiza en la interfaz gráfica. Análogamente, los parámetros se pueden modificar directamente sobre la interfaz gráfica y la pulsación de una tecla los actualizará partiendo del último valor que tengan.

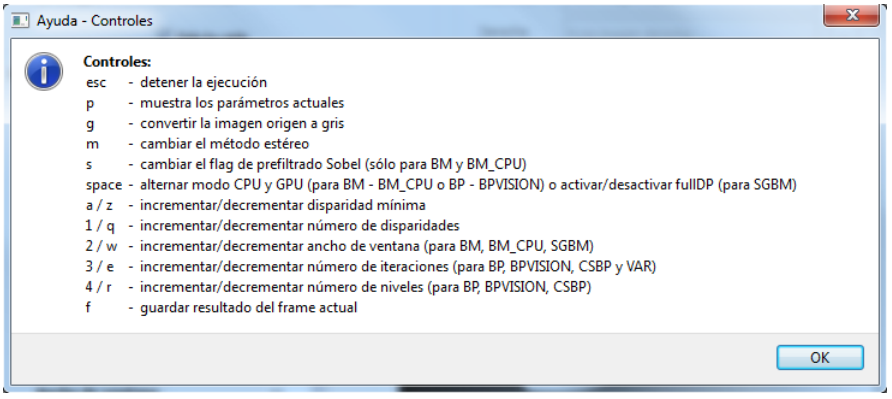


FIGURA 5.11 – **Barra de menú: ayuda de comandos.**

La opción "Acerca de.." abre un cuadro de diálogo con la descripción de la herramienta. Como vemos en la figura 5.12, se trata del título de este proyecto y el nombre del autor.

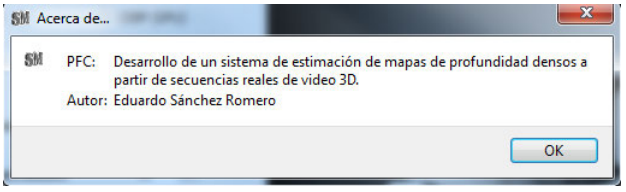
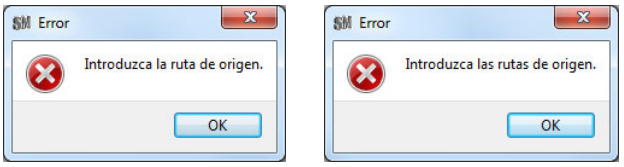


FIGURA 5.12 – **Barra de menú: ayuda acerca de...**

5.4.8 Mensajes de error y avisos

Para finalizar el capítulo, vamos a presentar algunos mensajes de error comunes que pueden aparecer durante la ejecución de la herramienta. Hay que comentar que son avisos que **no implican la finalización** de la ejecución de la interfaz gráfica y que **pueden ayudar** al usuario en caso de haber cometido un error al introducir los parámetros.

El primer error que se puede cometer es el llamado **error de ruta o rutas de origen**. Se produce cuando los campos de texto correspondientes a las rutas de archivos o directorios de entrada están en blanco en el momento de leer los parámetros. En caso de producirse, se muestra alguno de los dos mensajes de la figura 5.13. Según sea el tipo de archivo de entrada, se muestra el error de la figura 5.13(a) si es *side-by-side*, y se muestra el de la figura 5.13(b) si se trata de vistas separadas.



(a) Modo *side-by-side* (b) Modo dos vistas

FIGURA 5.13 – **Error de ruta de origen**

Una vez leídos los parámetros, se dispone de las rutas de entrada en las variables `src` o `left_param` y `right_param`. Si la ruta no se corresponde con ningún archivo de imagen o vídeo existente, se produce un **error de archivo**. En la imagen 5.14, el archivo "rutaFalsa.png" no existe.

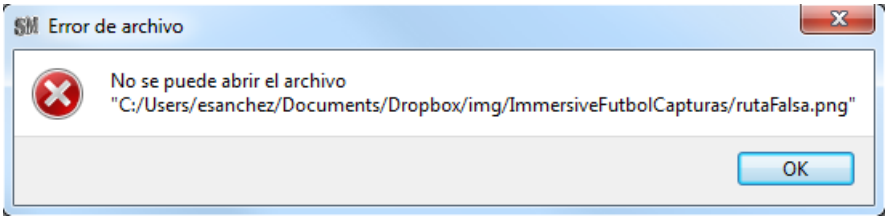


FIGURA 5.14 – **Error de archivo.**

Trabajando con secuencias, las variables `src` o `left_param` y `right_param` contienen la ruta de los directorios de entrada. Si no se corresponde con ningún directorio existente, se produce un **error de directorio**. En el ejemplo de la figura 5.15, se ha introducido como parámetro la ruta de un directorio llamado "resultados" que no existe.

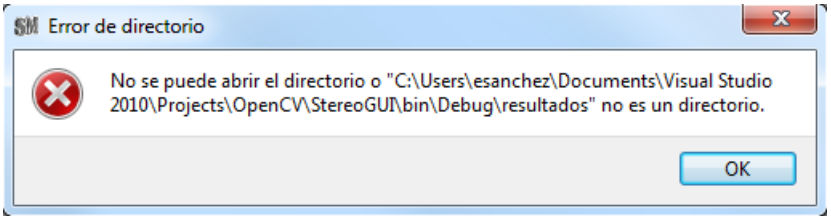
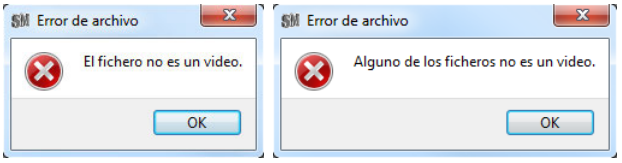


FIGURA 5.15 – **Error de directorio.**

Por otra parte, cuando se espera que la ruta del archivo o archivos de entrada corresponda con la de un vídeo y no es así, se produce un **error de archivo de vídeo**.



(a) (b)

FIGURA 5.16 – **Error de archivo de vídeo.**

Cuando se tiene las dos vistas almacenadas en las matrices `left_src` y `right_src` extraídas a partir de una imagen o de un cuadro de vídeo, se realizan ciertas comprobaciones. Una de ellas compara las dimensiones de las imágenes, ya que el proceso de correspondencia requiere dos vistas de la misma escena que tendrán las mismas propiedades. Si no coindicen las dimensiones, se produce un **error de entrada** como el de la imagen siguiente.

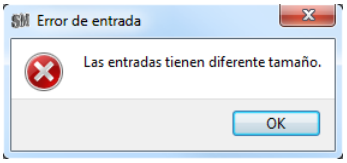


FIGURA 5.17 – **Error de entrada.**

Para acabar, el **error de fichero de resultados** se produce cuando no se puede leer o escribir correctamente al guardar los resultados. Generalmente ocurre cuando, al realizar una captura individual, el archivo csv de resultados está abierto con otro programa como *Excel*. Es un error bastante común ya que es habitual consultar los resultados en tiempo de ejecución. Cuando se produce, se guarda correctamente la imagen reconstruida y la diferencia normalizada, pero las estadísticas no se añaden al archivo de texto haciendo inútil la captura.

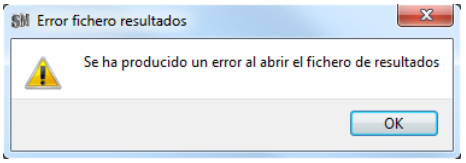


FIGURA 5.18 – **Error fichero resultados.**

5.5 Flujo del programa

Cuando arrancamos el programa aparece la interfaz gráfica de usuario (GUI), que muestra todos los parámetros que podemos seleccionar y las acciones a realizar (sección 5.4). Como se puede ver en el diagrama de flujo de la figura 5.19, las tres acciones que se pueden realizar son `RESET`, `STOP` y `RUN`. La primera restablece los parámetros por defecto, la segunda detiene la ejecución del proceso de correspondencia modificando la variable `running` (`running = false`), mientras que la última inicia la ejecución del proceso de correspondencia.

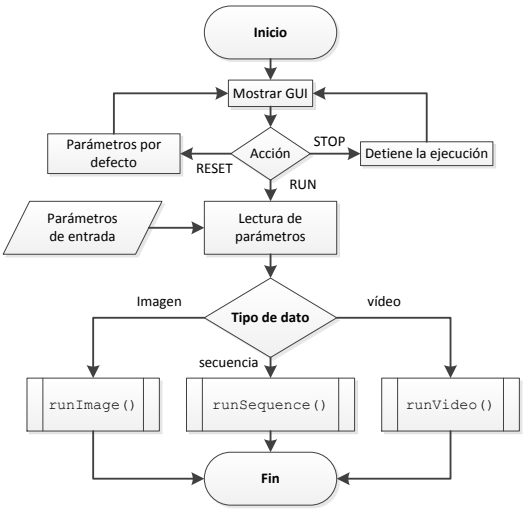


FIGURA 5.19 – **Flujo del programa.**

Tras accionar `RUN`, inmediatamente se leen los parámetros de entrada elegidos en la interfaz gráfica para su posterior aplicación. Según el tipo de datos de entrada (imagen, secuencia o vídeo), se llamará a un subproceso distinto (`runImage()`, `runSequence()` o `runVideo()`) para realizar la correspondencia. A continuación explicamos en detalle cada uno de los subprocesos.

5.5.1 Imagen (`runImage()`)

Cuando el archivo de entrada es una imagen, se ejecuta el subproceso `runImage()`, que está representado en el diagrama 5.20. En primer lugar se comprueba el formato de la imagen de entrada. Si se trata de una **imagen *side-by-side***, se obtiene el nombre del único archivo del que se dispone (`nombreFichero.ext`) a partir de la ruta de origen que se ha obtenido como parámetro (`src`). La razón por la que extraemos el nombre es porque se puede utilizar para nombrar los archivos de resultados. Para realizar la correspondencia es necesario disponer de dos vistas, una para cada ojo, por lo tanto se divide la imagen SbS en dos subimágenes de ancho mitad (`left_src` y `right_src`). Si, por el contrario, tenemos **dos imágenes** de entrada **separadas**, se extrae el nombre del archivo correspondiente a la vista izquierda a partir de la ruta (`left_param`) y no hace falta realizar ninguna operación adicional.

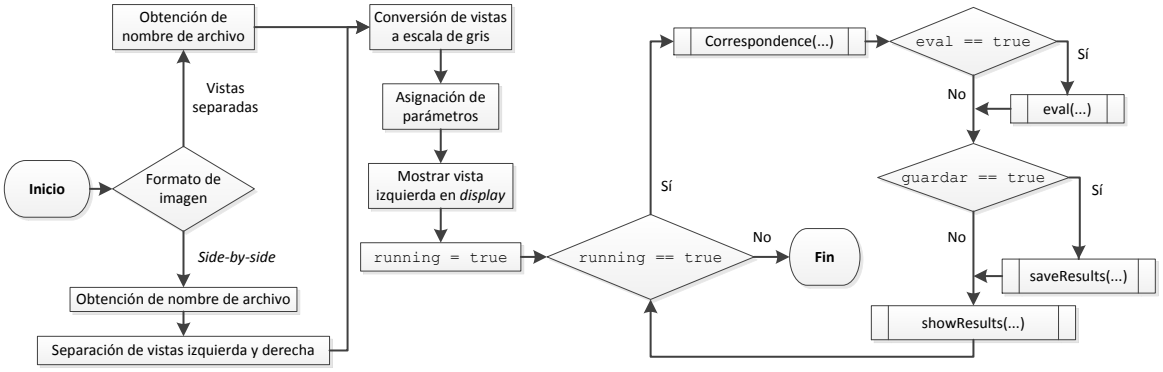


FIGURA 5.20 – **Flujo del subproceso `runImage()`**

Una vez tenemos ambas imágenes, comienza la preparación para aplicar los algoritmos de correspondencia. Primero **se convierten a escala de gris** (`left` y `right`). A pesar de que algunos algoritmos admiten imágenes en color, aunque puede darse el caso de que internamente las transformen a escala de gris, hacemos esta transformación para compararlos de forma más fiable porque así están en igualdad de condiciones. A continuación **se asignan los parámetros** de entrada a las variables y atributos de los distintos objetos que representan a los algoritmos de correspondencia y se muestra la imagen izquierda en la interfaz gráfica.

El siguiente paso es comenzar la ejecución del **proceso de correspondencia** activando el *flag* `running` y entrando en un bucle de tipo `while`. Mientras el *flag* permanezca activado, el subproceso `correspondence(...)` (sección 5.5.4) estima el mapa de disparidad. Una vez calculado el mapa de disparidad se comprueba si hay que **evaluar los resultados** obtenidos (subproceso `eval(...)`, sección 5.5.5) y si hay que **guardarlos** (subproceso `saveResults(...)`, sección 5.5.6). Hay que comentar que si se selecciona la opción de guardar al inicio de la ejecución, sólo se almacenarán los resultados de la primera iteración. En cualquier caso, se muestran los resultados tal y como se explica en la sección 5.5.7. Por último, **se escucha del teclado** si ha sido presionada alguna tecla para actualizar los parámetros y/o el estado de la ejecución o, como explicaremos en la sección 5.5.6, **realizar una captura**. Esto nos permite **variar los parámetros** dinámicamente para encontrar el mejor resultado y guardarlo. El estado `running` se puede desactivar accionando `STOP` (generando una interrupción) o la tecla escape del teclado, deteniendo la ejecución.

5.5.2 Secuencia de imágenes (`runSequence()`)

Este subproceso es muy parecido a `runImage()`, salvo que está pensado para calcular automáticamente el mapa de disparidad de todas las **imágenes estéreo contenidas en un directorio** o directorios especificados (diagrama 5.21). Aunque se denomina secuencia, las imágenes no tienen por qué estar relacionadas entre sí, pero también puede ser útil para estimar la disparidad de varios cuadros consecutivos capturados de un vídeo.

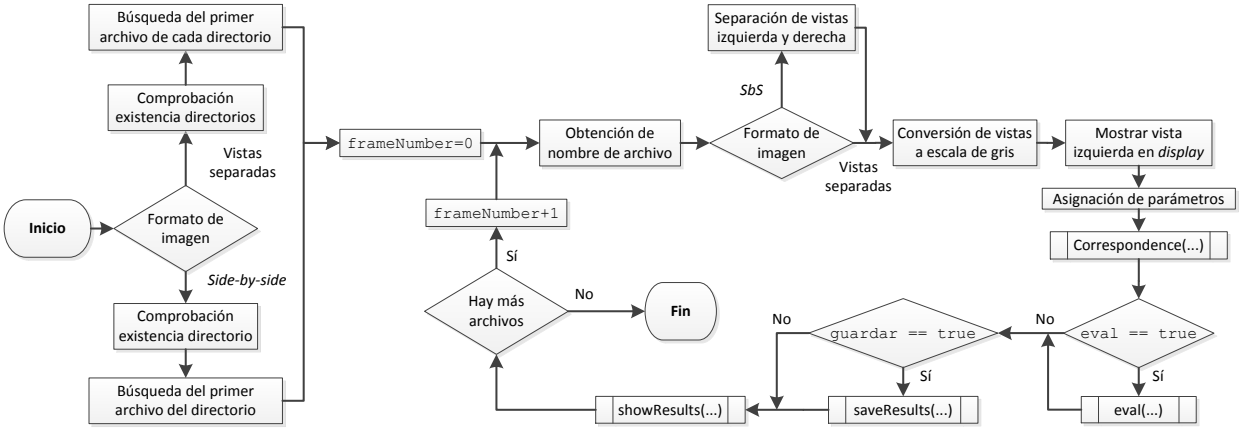


FIGURA 5.21 – **Flujo del subproceso `runSequence()`**

Como en el caso anterior, primero se comprueba el tipo de archivo de entrada. Ahora, el parámetro `src` representa la ruta del directorio que contiene el conjunto de imágenes estéreo en formato ***side-by-side*** y se comienza comprobando su existencia. A continuación se busca el primer archivo contenido en el directorio. Para dos **vistas separadas**, deben encontrarse en directorios distintos (`left_param` y `right_param`) y tener el mismo nombre, se comprueba la existencia de ambos y se localiza el primer archivo dentro de cada uno de ellos.

Localizados los archivos de imagen, se inicia el contador `frameNumber` y, al igual que en `runImage()`, se extrae su nombre y se preparan para el proceso de correspondencia (separación en dos vistas, si es necesario, y conversión a escala de gris). Tras la asignación de los parámetros a las variables y objetos que representan a los algoritmos, se comprueba si hay que evaluar y guardar los resultados. Estos, después se muestran en pantalla. Esta parte coincide con lo explicado en `runImage()`.

En cada iteración se comprueba si **hay más archivos** dentro de los directorios. Si es así, se incrementa en uno el contador de imágenes y se repite el proceso descrito

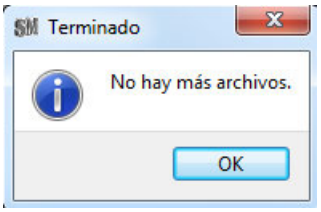


FIGURA 5.22 – **Secuencia finalizada correctamente.**

en el párrafo anterior. Si **no hay más archivos** y no ha ocurrido ninguna incidencia, se muestra el mensaje informativo de la figura 5.22, terminando la ejecución.

5.5.3 Vídeo (`runVideo()`)

Cuando se trabaja con vídeos, el proceso es muy similar a los dos anteriores salvo algunos matices. Se comienza inicializando un **contador** de cuadros que será útil, tanto para registrar la cantidad de imágenes procesadas en el cálculo de estadísticas, como para generar el nombre de las capturas de resultados. El otro contador que se inicializa es la **suma de *frames per second***, que se emplea para obtener el valor medio de cuadros por segundo que puede procesar nuestra herramienta.

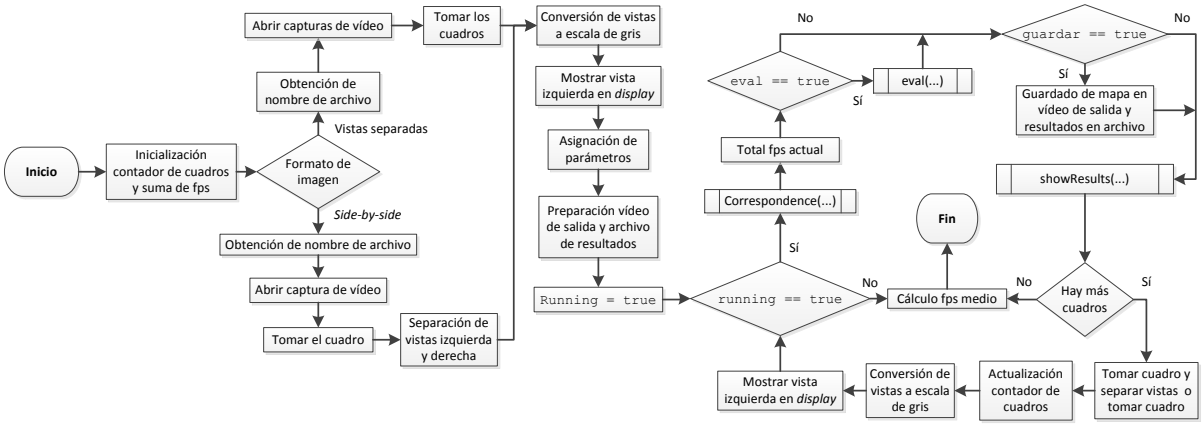


FIGURA 5.23 – **Flujo del subproceso `runVideo()`**

Al igual que en los anteriores subprocesos, se comprueba el formato de entrada. Si es un vídeo *side-by-side* (`video`), se obtiene su nombre, se abre una **captura de vídeo** y se toma el primer cuadro, que hay que separar en dos imágenes de ancho mitad. Por otra parte, si se trata de dos vídeos separados (`left_video` y `right_video`), se obtiene el nombre de la vista izquierda, se abren dos capturas de vídeo y se toma el primer cuadro de cada uno de ellos.

La preparación de las imágenes no cambia: conversión a escala de gris, se muestra la vista izquierda en el GUI y se asignan los parámetros a las variables y objetos que representan a los algoritmos. Una de las novedades que incorpora el procesamiento de los vídeos es la **preparación del vídeo de salida y del archivo de resultados**. Este módulo se explica con mayor detalle en el apartado 5.5.6 y su existencia se debe a que trabajar con vídeo exige un tratamiento diferente de los resultados.

Tras activar el *flag* `running` se entra en el bucle de ejecución del **proceso de correspondencia**. Después de calcular el mapa de disparidad, se actualiza el valor del contador suma de fps y se comprueba si hay que **evaluar** y **guardar** los resultados. En este caso, no se utiliza `saveResults(...)`, sino que se guarda directamente el mapa de disparidad actual en el vídeo de salida y se añaden algunos datos (que serán explicados en el apartado 5.5.6) en el archivo de resultados de vídeo. Posteriormente se muestran los resultados por pantalla. Si el vídeo contiene más cuadros, se repite la **fase de preparación**: tomar el cuadro, separar las dos vistas si fuera necesario y conversión a escala de gris. También hay que actualizar el contador de cuadros. Como en `runImage()`, se escucha si se ha pulsado alguna tecla para actualizar los parámetros y/o estado de ejecución. El bucle se repite mientras no cambie el estado `running` (accionando `STOP` o la tecla escape del teclado) y hasta que no se alcance el final del vídeo. En cualquiera de las dos situaciones, se **finaliza la ejecución** tras calcular el valor medio de los cuadros por segundo.

5.5.4 El módulo de correspondencia (`correspondence(...)`)

El proceso de correspondencia está implementado mediante una **función** que pertenece a la clase *StereoGUI* y que toma como **parámetros de entrada** las imágenes izquierda y derecha preparadas y los parámetros de la aplicación, y como **salida** la referencia del mapa de disparidad:

```
void StereoGUI :: correspondence(Mat& left, Mat& right,
                                 const Params& p, Mat& disp);
```

Esta función comienza **comprobando el método** elegido (`method`), ya que cada uno de ellos utiliza **distintos tipos de datos** de entrada, por lo que es necesario realizar algunas adaptaciones sobre las matrices de entrada. Por ejemplo, los algoritmos basados en GPU exigen matrices de entrada y salida contenidas en la memoria del dispositivo

gráfico (`cv::gpu::GpuMat`), en lugar de matrices en la memoria CPU (`cv::Mat`). Otros requieren que los valores de las imágenes estén en formato de 16 o 32 bits en coma flotante (`short` o `float`) en vez de utilizar 8 bits sin signo (`unsigned char`).

A continuación se hace una llamada a la función correspondiente al método, que devolverá una matriz con el **valor real de disparidad** de cada píxel de la imagen. De nuevo, según el método, la matriz de disparidad tendrá un formato distinto (entero de 8 bits con signo, entero de 16 bits con signo o 32 bits en coma flotante), por lo que, de nuevo se debe realizar una transformación para uniformizar los resultados. Se convierte la matriz de disparidad real a formato de 32 bits en coma flotante porque es el tipo de dato menos restrictivo y admite valores negativos. El resultado se guarda en `disp_left` (o `disp_right` para la vista derecha).

Para poder visualizar el mapa de disparidad como una imagen, sus valores deben estar comprendidos entre 0 y 255, siendo necesario realizar una **normalización** siguiendo la siguiente expresión:

$$disp\_left\_norm(i,j) = \frac{disp\_left(i,j) - \text{mín}\,\{disp\_left\}}{\text{máx}\,\{disp\_left\} - \text{mín}\,\{disp\_left\}} \times 255 \tag{5.1}$$

donde se observa que la matriz `disp_left_norm` almacena el **mapa de disparidad normalizado** apto para la visualización.

Por último, hay que comentar que se almacena el instante de comienzo del proceso de correspondencia en la variable `work_begin` para poder calcular la **tasa de cuadros por segundo** al finalizar. Como veremos, esta medida se muestra siempre sobre el mapa de disparidad al visualizar los resultados.

5.5.5 Evaluación de resultados (`eval(...)`)

La evaluación está definida en la **clase *StereoEval*** (`class StereoEval`). Tras el proceso de correspondencia, si está activado el *flag* `eval`, se aplica la siguiente función, que toma como parámetros las dos vistas originales (`left_src` y `right_src`), el mapa de disparidad real calculado y el parámetro de disparidad mínima `dmin`:

```
void StereoEval :: eval(Mat& left_original, Mat& right_original,
                        Mat& disp_calc, int dmin);
```

Las **variables** definidas en esta clase son las siguientes:

- **Variables estadísticas**

  `double rms`**:** error cuadrático medio.

  `double psnr`**:** relación pico señal a ruido.

  `double fps`**:** tasa de cuadros por segundo.

  `double invalidpix`**:** porcentaje de píxeles inválidos.

  `double badpix`**:** porcentaje de píxeles erróneos.

- **Imágenes de salida**

  `Mat reconst`**:** matriz que almacena la imagen de referencia reconstruida.

  `Mat diffNormCol`**:** matriz que almacena la imagen diferencia normalizada.

La función `StereoEval::eval` comienza calculando la **imagen de referencia reconstruida** mediante:

```
void StereoEval :: getLeftFromRight(const Mat& ref, const Mat& disp,
                                    Mat& target);
```

Esta función, capaz de reconstruir la imagen izquierda (`target`) a partir de la vista derecha (`ref`) y el mapa de disparidad real (`disp`), ha sido **implementada en CUDA**. La reconstrucción es una **transformación inversa**, es decir, para cada píxel de la imagen de salida que queremos generar, buscamos la posición horizontal del píxel de la imagen de entrada cuyo valor hay que copiar (equación 5.2), como se ilustra en la figura 4.7.

$$x_R = x_L - d(x_L, y_j); \tag{5.2}$$

donde x_R es la coordenada horizontal del píxel de la imagen `right_src` (`ref` en la función), x_L es la coordenada horizontal del píxel que estamos calculando de la imagen `reconst` (`target` en la función) y $d(x_L, y_j)$ es el valor real de disparidad correspondiente a la posición del píxel que estamos calculando y que obtenemos de `disp_left` (`disp` en la función).

Puesto que el valor de disparidad toma valores reales (`float`), la posición horizontal obtenida del píxel **no siempre** será un **valor entero**, por lo que es necesario calcular el valor del píxel mediante una **interpolación lineal** a partir de los vecinos más próximos. Hemos llamado `pixInf` al píxel cuya coordenada horizontal es el primer

entero inferior a x_R y `pixSup` al píxel cuya coordenada horizontal es el entero superior (figura 5.24).

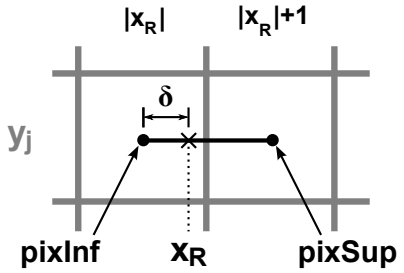


FIGURA 5.24 – **Interpolación lineal.**

En la imagen, se denota al entero inferior como $|x_R|$, ya que se obtiene extrayendo la parte entera de x_R, y la posición del entero superior se calcula aumentando la coordenada del inferior en una unidad ($|x_R| + 1$). La distancia entre x_R y $|x_R|$ es δ. El valor del píxel se obtiene mediante la interpolación definida por la siguiente expresión:

$$\begin{aligned} \hat{I}_L(x_L, y_j) &= (1-\delta) \cdot I_R(|x_R|, y_j) + \delta \cdot I_R(|x_R|+1, y_j) = \\ &= I_R(|x_R|, y_j) + [I_R(|x_R|+1, y_j) - I_R(|x_R|, y_j)] \cdot \delta \end{aligned} \tag{5.3}$$

Una vez realizada la reconstrucción se realiza la evaluación propiamente dicha. Para ello se llama al subproceso que se encarga de **calcular los estadísticos** (`rms` y `psnr`) y a continuación se genera la imagen diferencia normalizada. Se define como sigue, donde `ref` es la imagen de referencia (en nuestro caso `left_src`) y `reconst` es la imagen reconstruida que se quiere comparar:

```
void StereoEval::evalSinGT(Mat& ref, Mat& reconst);
```

Por último, **se cuentan los píxeles inválidos y** se extraen los **erróneos** con ayuda de la imagen de error normalizada para calcular las respectivas tasas P_i y P_e. Se marcan los píxeles inválidos sobre la imagen reconstruida y de error normalizada, de manera que se puede apreciar a simple vista la cantidad de píxeles sin valor de disparidad asignado. De la imagen de error normalizada se puede apreciar que los valores de mayor intensidad son los píxeles erróneos. En la sección 5.5.7 se mostrará el aspecto que presentan estas imágenes.

5.5.6 Guardar resultados

En el proceso de guardado de resultados debemos distinguir entre trabajar con imágenes y secuencias, o con vídeo. Como hemos visto, `runImage()` y `runSequence()` comprueban si hay que guardar los resultados una vez estimado el mapa de disparidad. En caso afirmativo, se llama a la siguiente función:

```
bool StereoGUI::saveResults(const string dir, const string extension);
```

Los parámetros de entrada son la ruta del directorio en el que se quiere guardar los resultados (`dir`) y la extensión que determina el formato de las imágenes resultantes (`extension`), que generalmente será png. El directorio de salida viene dado por el parámetro de entrada de la aplicación `output_dir`. En primer lugar, se genera el **nombre completo** con el que guardar el mapa de disparidad y concatena con la ruta de salida. El nombre depende del método de correspondencia aplicado ya que cada uno tiene distintos parámetros. El formato general de nombres es el siguiente:

`nombreFichero_p.type(p.method)`dmin(`p.dmin`)ndisp(`p.ndisp`)w(`p.winSize`)lev(`p.levels`)it(`p.iterations`)

Partiendo de este formato, el nombre **varía en función del método de correspondencia** (`p.method`) eliminando todos aquellos parámetros que no utiliza. Algunos algoritmos como `BM` y `BM_CPU` permiten activar un prefiltrado de las imágenes durante el proceso de correspondencia, esto se indica añadiendo a continuación del nombre "pref_sob(*flag*)" o "preFilterType(*tipo*)", respectivamente. También `SGBM` admite una optimización mediante programación dinámica y aparece en el nombre como "DP(`s/n`)". Además de todo esto, si el algoritmo utilizado calcula dos mapas de disparidad, se añade "_L" o "_R" y, por último la extensión.

Por ejemplo, sobre una imagen de entrada llamada "captura03.bmp"hemos aplicado el algoritmo `BM` con disparidad mínima `p.dmin=-10`, número de niveles de disparidad `p.ndisp=16`, tamaño de ventana `p.winSize=7` y prefiltrado desactivado. El nombre que se genera cuando llamamos a `saveResults(...)` indicando la extensión "png" es:

`captura03_Image(BM)dmin(-10)ndisp(16)w(7)pref_sob(0).png`

Bajo este nombre se guarda el **mapa de disparidad normalizado** ya que no necesitamos saber el valor real de profundidad porque lo más importante para nuestro

problema es conocer la relación entre los distintos planos de disparidad. Si además se ha realizado la **evaluación de resultados**, opción que se selecciona automáticamente al guardar los resultados en el caso de las imágenes y secuencias, se añade al nombre anterior el sufijo "Rec" para la imagen reconstruida (`reconst`) y "Diffn"para la diferencia normalizada (`diffNormCol`). Siguiendo con el ejemplo anterior, se habrá generado los siguientes nombres:

`captura03_Image(BM)dmin(-10)ndisp(16)w(7)pref_sob(0)Rec.png`

`captura03_Image(BM)dmin(-10)ndisp(16)w(7)pref_sob(0)Diffn.png`

En la misma ruta que las imágenes, también se guarda **un archivo de resultados** (`results`) con el nombre "Resultados.csv". Se trata de un archivo de texto en formato csv. El csv se caracteriza por su sencillez para recoger gran cantidad de información separada por comas. Cada vez que se guardan los resultados, se añade una fila siguiendo el siguiente formato:

"`nombreFichero`","`p.type`","`p.method`","`p.dmin`","`p.ndisp`","`p.winSize`","`p.iterations`","`p.levels`",

"`fps`","`rms`","`psnr`","P_i" "P_e"

La figura 5.25 muestra una captura del archivo "Resultados.csv" de nuestro ejemplo. Como se observa, hay **campos que no tienen valor**, esto se debe a que el método `BM` no utiliza los parámetros `p.iterations` y `p.levels`.

```
"Nombre","Tipo","Método","Disparidad mínima","Número de disparidades","Ventana","Iteraciones","Niveles","FPS","RMS","PSNR","Px inválidos (%)","Px erróneos (%)"
"captura03.bmp","Image","BM","-10","16","7","","","83.33","59.01","12.71","13.73","18.19"
```

FIGURA 5.25 – **Captura del archivo de resultados en formato csv.**

El archivo de texto es poco visual. Afortunadamente, se puede visualizar con *Microsoft*^{TM} *Excel* de manera más amigable en forma de tabla.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nombre	Tipo	Método	Disparidad mínima	Número de disparidades	Ventana	Iteraciones	Niveles	FPS	RMS	PSNR	Px inválidos (%)	Px erróneos (%)
2	captura03.bmp	Image	BM	-10	16	7			83.33	59.01	12.71	13.73	18.19

FIGURA 5.26 – **Captura del archivo de resultados visualizado en Excel.**

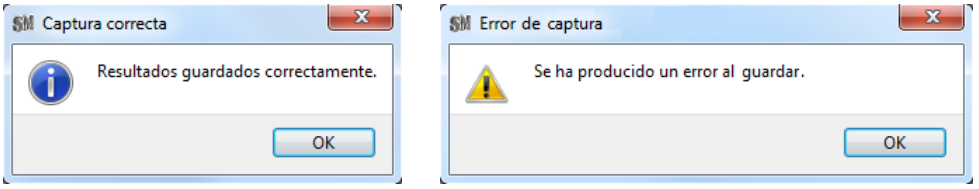
Nuestra herramienta permite además **realizar capturas de resultados** pulsando la tecla 'f' o accionando el botón "Capturar" de la interfaz gráfica. Como hemos comentado en el apartado `runImage()` (5.5.1), la variación dinámica de los parámetros

permite encontrar el mejor resultado y la realización de capturas sucesivas facilita su comparación. Cada vez que se realiza una nueva captura, se añaden los resultados al archivo.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Nombre	Tipo	Método	Disparidad mínima	Número de disparidades	Ventana	Iteraciones	Niveles	FPS	RMS	PSNR	Px inválidos (%)	Px erróneos (%)
2	captura03.bmp	Image	BM	-10	16	7			83.33	59.01	12.71	13.73	18.19
3	captura03.bmp	Image	BM_CPU	-10	16	7			49.01	85.98	9.443	30.23	12.77
4	captura03.bmp	Image	BM_CPU	-12	16	7			50.71	76.2	10.49	23.55	14.58

FIGURA 5.27 – **Realización de capturas sucesivas.**

Si todos los resultados se han **guardado correctamente**, aparecerá un cuadro de diálogo de confirmación. En caso contrario, aparecerá un aviso de error.



(a) Guardado correctamente (b) Error de guardado

FIGURA 5.28 – **Mensaje que confirma que los resultados se han guardado correctamente (izquierda) y advertencia de error (derecha).**

Ahora explicamos cómo se guardan los resultados para **vídeo**. En el apartado `runVideo()` (5.5.3) hemos introducido la preparación del vídeo de salida (`outputVideo`) y del archivo de resultados (`res_video`), porque además del csv ya explicado, cada vídeo **lleva asociado un archivo propio**. El nombre de este nuevo csv es el nombre del vídeo acabado en "out". Por defecto, cuando se trabaja sobre vídeos, no se realiza la evaluación en cada cuadro y el formato de resultados es el de la figura 5.29. Podemos observar que se calcula la tasa de cuadros por segundo para cada *frame* y cuando finaliza la ejecución del proceso de correspondencia se obtiene el **valor medio**.

1	Nombre	Tipo	Frame nº	Método	Disparidad mínima	Número de disparidades	Ventana	Iteraciones	Niveles	FPS
2	partido.avi	Video	0	BM	-14	16	9			54.83
3	partido.avi	Video	1	BM	-14	16	9			64.65
4	partido.avi	Video	2	BM	-14	16	9			53.25
5	partido.avi	Video	3	BM	-14	16	9			67.15
6	[illegible]	[illegible]	4	[illegible]	[illegible]	[illegible]	[illegible]			73
937	[illegible]	[illegible]	[illegible]	BM	-14	16	9			68.97
938	partido.avi	Video	936	BM	-14	16	9			69.85
939	partido.avi	Video	937	BM	-14	16	9			71.04
940	partido.avi	Video	938	BM	-14	16	9			64.64
941	partido.avi	Video	939	BM	-14	16	9			71.08
942	Número de frames	940								
943	Suma FPS	65312.6								
944	FPS medio	69.482								

FIGURA 5.29 – **Archivo de resultados para vídeo**

Adicionalmente, se pueden realizar **capturas individuales** de cuadros del vídeo presionando 'f' o accionando "Capturar". En tal caso, se realiza la evaluación de los resultados y, al igual que con las imágenes y secuencias, se guarda el mapa de disparidad normalizado, la imagen reconstruida y la diferencia normalizada. También se añade la información en el archivo "Resultados.csv". La única diferencia es que se añade el número de *frame* en el nombre de los archivos, como podemos observar en la primera columna de la figura 5.30.

Nombre	Tipo	Método	Disparidad mínima	Número de disparidades	Ventana	Iteraciones	Niveles	FPS	RMS	PSNR	Px inválidos (%)	Px erróneos (%)
captura03.bmp	Image	BM	-10	16	7			83.33	59.01	12.71	13.73	18.19
captura03.bmp	Image	BM_CPU	-10	16	7			49.01	85.98	9.443	30.23	12.77
captura03.bmp	Image	BM_CPU	-12	16	7			50.71	76.2	10.49	23.55	14.58
captura01.bmp	Sequence	BM	-13	16	9			0.3318	68.53	11.41	17.64	34.76
captura02.bmp	Sequence	BM	-13	16	9			73.09	56.23	13.13	12.29	18.67
captura03.bmp	Sequence	BM	-13	16	9			76.63	58.02	12.86	13.3	16.71
captura04.bmp	Sequence	BM	-13	16	9			74.38	95.98	8.487	34.46	7.146
partido.avi(43)	Video	BM	-14	16	9			71.98	114.7	6.937	46.27	5.779
partido.avi(215)	Video	BM	-14	16	9			71.37	75.24	10.6	20.24	21.43
partido.avi(923)	Video	BM	-14	16	9			68.22	49.77	14.19	7.684	35.29

FIGURA 5.30 – **Archivo de resultados.**

Es importante destacar que se guarda un **vídeo de salida** cuyo nombre es el del vídeo original acabado en "out" y en formato ".avi". El vídeo contiene el mapa de disparidad normalizado correspondiente a cada uno de los cuadros del vídeo original. El principal inconveniente es el gran tamaño del archivo generado.

5.5.7 Presentación de los resultados (`showResults(...)`)

El proceso encargado de mostrar los resultados está definido por una **función** que pertenece a la clase *StereoGUI* que toma como parámetros de entrada los parámetros `p` de la aplicación:

```
void StereoGUI::showResults(const Params& p);
```

Se trata de una sencilla función que abre una ventana con el título *"Normalized Left Disparity"* y sitúa en ella el **mapa de disparidad normalizado** correspondiente a la vista izquierda calculado en el proceso de correspondencia. Si el algoritmo utilizado calcula también el mapa de disparidad derecho (`p.method == Params::ELAS || p.method == Params::GC`), se mostrará en una ventana con el título *"Normalized Right Disparity"*.

En la esquina superior izquierda de la imagen aparece el valor de la **tasa de cuadros por segundo (`fps`)**, aunque como hemos comentado, esto solo se hace al

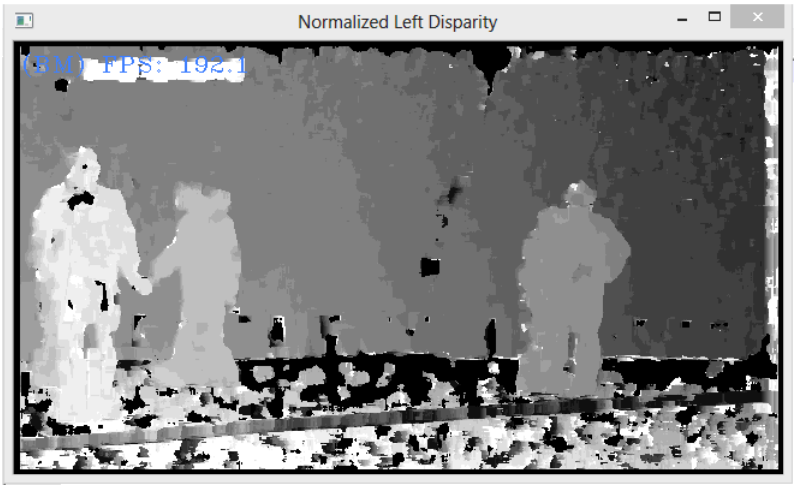
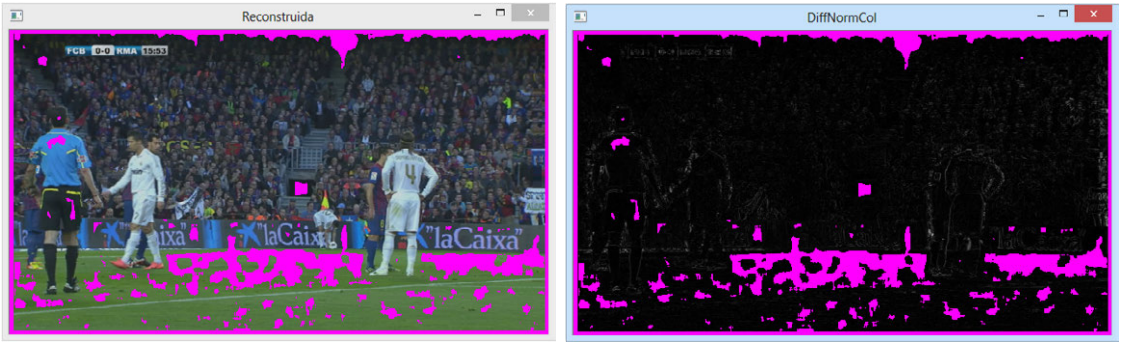


FIGURA 5.31 – **Mapa de disparidad normalizado:** ejemplo de resultado obtenido al aplicar el método BM_GPU (`p.method==BM`) sobre la captura del vídeo mostrada en la figura 1.8.

visualizar ya que, al guardar, la tasa calculada se escribe en el archivo de resultados. Esto nos proporciona de manera rápida una idea aproximada de la eficiencia de cada algoritmo.

Por otra parte, si se ha realizado la **evaluación de los resultados**, se abre una ventana para mostrar la imagen de referencia reconstruida (`reconst`) con el título *"Reconstruida"* y otra ventana para la imagen diferencia normalizada (`diffNormCol`) con el título *"DiffNormCol"*.



(a) Imagen reconstruida (b) Imagen diferencia normalizada

FIGURA 5.32 – **Resultados de la evaluación.** A la izquierda la imagen de referencia reconstruida y a la derecha la imagen diferencia normalizada.

Como se observa, los **píxeles inválidos** de la imagen reconstruida y de la imagen diferencia normalizada se han representado con otro color. Esto aporta visualmente una idea de la calidad del resultado obtenido con cada algoritmo.

Capítulo 6

Pruebas y evaluación de resultados

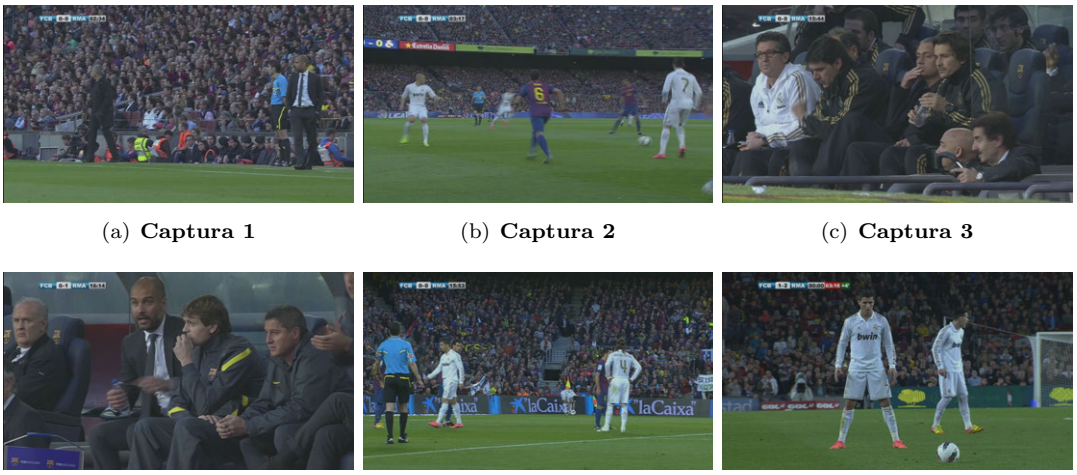
6.1 Introducción

Ya hemos explicado los fundamentos de la correspondencia estéreo y las funcionalidades de nuestra herramienta para evaluar qué algoritmos se comportan mejor frente a un problema determinado. Ahora vamos a describir y analizar los resultados obtenidos de la comparación realizada utilizando la herramienta para comprobar cuál de los algoritmos analizados ofrece mejores resultados en el caso que nos ocupa.

Comenzamos describiendo las capturas utilizadas para las pruebas, justificando la elección de acuerdo a ciertos criterios. La calidad de los resultados se puede analizar atendiendo tanto a criterios subjetivos, mediante inspección visual, como a criterios objetivos que permiten compararlos utilizando información que, a priori, parece más fiable. El análisis objetivo de calidad se basa en las tasas de píxeles inválidos y erróneos, indicadores de la capacidad del mapa de disparidad para recuperar la imagen de referencia mediante la reconstrucción a partir de la segunda vista y el mapa, así como en los indicadores de error RMS y PSNR. Por otra parte, el análisis de eficiencia de los algoritmos se realiza midiendo la tasa de cuadros por segundo que son capaces de procesar.

6.2 Capturas de prueba

Para representar en este documento los resultados de las pruebas realizadas, hemos escogido un conjunto de capturas del vídeo del partido, descrito en la sección 1.3.2, que aparecen recogidas en la figura 6.1.



(a) **Captura 1** (b) **Captura 2** (c) **Captura 3**

(d) **Captura 4** (e) **Captura 5** (f) **Captura 6**

FIGURA 6.1 – Capturas del vídeo del partido que utilizamos en nuestras pruebas.

Como ya hemos explicado, todas las capturas están en formato *HD Ready Side-by-Side* (2560x720 píxeles) conteniendo una vista para cada ojo de manera que la imagen correspondiente a cada uno de ellos es un cuadro HD 720p (1280x720 píxeles). Se ha tratado de escoger un subconjunto representativo que reúna el mayor número posible de situaciones conflictivas que se pueden producir en un encuentro deportivo.

En particular, las capturas 1, 2, 5 y 6 presentan grandes regiones uniformes o **sin textura** en la zona del terreno de juego (césped). Por otra parte, algunas presentan un **escenario con movimiento** tanto de los jugadores como del balón (capturas 2 y 5) en las que se produce emborronamiento. Sin embargo, las capturas 1, 3, 4 y 6 muestran situaciones más estáticas con mejor definición de los objetos de la escena. Una captura tomada a una distancia lejana de la escena, como la 1, 2 y 5, da lugar a un mayor **número de planos de profundidad** (niveles de disparidad) aumentando la posibilidad de producirse **oclusiones**. Por el contrario, las capturas 3, 4 y 6, están tomadas más cerca de la escena. Además, las capturas 1, 2, 5 y 6 tienen **zonas altamente texturizadas en superficies inclinadas** (gradas con público) que favorecen el proceso de correspondencia, donde los saltos entre niveles de disparidad del mapa dependen de la resolución (en bits) del algoritmo. Las capturas 3 y 4 son diferentes de las anteriores ya que reflejan una **escena** más **cercana** donde muchos objetos o personas dan lugar a una alta tasa de **oclusiones**.

6.3 Experimentos y discusión

A continuación detallamos los experimentos realizados para estudiar los algoritmos. Comenzamos por el análisis de calidad, tanto subjetiva como objetiva, y finalizamos con la comparación de eficiencia de los algoritmos.

6.3.1 Calidad

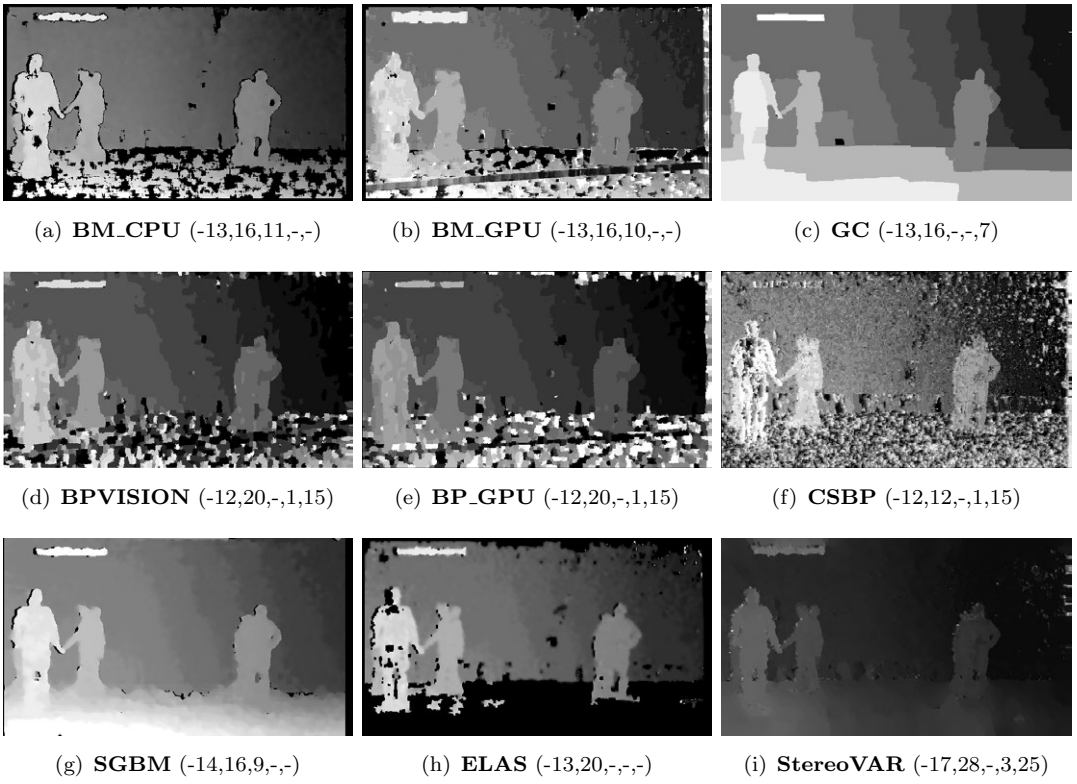
La calidad del mapa de disparidad viene dada por su capacidad para representar correctamente, para cada píxel de la imagen, los diferentes niveles de profundidad que existen en la escena tridimensional. Esta capacidad se puede analizar visualmente, observando el aspecto que presenta el mapa de disparidad en cada región de la imagen (suavidad y saltos de nivel de disparidad, agujeros, oclusiones) o cómo se comporta ante variaciones de los parámetros de correspondencia. Esto constituye el **análisis subjetivo**. Pero, también se puede realizar un **análisis objetivo** a partir de los resultados obtenidos de la comparación entre la imagen de referencia reconstruida y la de referencia original, y a partir de la información procedente de la imagen diferencia normalizada.

6.3.1.1. Calidad subjetiva

El experimento realizado para evaluar la calidad subjetiva de las diferentes técnicas de correspondencia estéreo escogidas para nuestra herramienta, ha sido la **búsqueda del mapa de disparidad aparentemente mejor** mediante la variación y ajuste de los diferentes parámetros. De esta manera, se puede apreciar el efecto de cada tipo de parámetro sobre el resultado final. Esto aporta un conocimiento adicional útil para el uso de la herramienta, ya que sabiendo cómo influye cada parámetro en el mapa de disparidad, se puede encontrar el mejor resultado más rápidamente.

La figura 6.2 presenta los mapas de disparidad seleccionados como resultado de aplicar cada método de correspondencia sobre la captura 5. Se aprecia que cada uno tiene sus propias características, todos son diferentes, y han sido obtenidos utilizando distintos valores de los parámetros de correspondencia (en la figura se indican como "(d_{min}, n_{disp}, win_{size}, n_{levels}, n_{iter})"). Esto dificulta la comparación entre técnicas y obliga a recurrir a la utilización de criterios subjetivos para decidir el método más adecuado en función de un

problema concreto. Es decir, según el caso, será preferible un buen comportamiento en zonas sin textura, cierta precisión cerca de los bordes o incluso un tratamiento concreto en las zonas ocluidas. A continuación, resumimos las características de los diferentes mapas de disparidad de la figura 6.2 en función de los parámetros.



(a) **BM_CPU** (-13,16,11,-,-) (b) **BM_GPU** (-13,16,10,-,-) (c) **GC** (-13,16,-,-,7)

(d) **BPVISION** (-12,20,-,1,15) (e) **BP_GPU** (-12,20,-,1,15) (f) **CSBP** (-12,12,-,1,15)

(g) **SGBM** (-14,16,9,-,-) (h) **ELAS** (-13,20,-,-,-) (i) **StereoVAR** (-17,28,-,3,25)

FIGURA 6.2 – Mapas de disparidad.

El primer mapa (figura 6.2(a)) se corresponde al algoritmo BM_CPU. Como se observa, el gradiente de la escala de grises es bastante suave, debido a que la **resolución** (profundidad de bit) proporcionada por el algoritmo es de 16 bits. Es decir, el valor de disparidad de cada píxel está representado por medio de un entero de 16 bits con signo. Como referencia, se puede comparar con el mapa situado a su derecha (BM_GPU) donde se aprecian saltos bruscos entre los distintos niveles de disparidad debido a que el valor de disparidad se representa mediante un entero de 8 bits con signo. A partir de esta comparación de referencia, se puede deducir la **profundidad de bit** del resto de mapas de disparidad:

- BM_GPU, GC, BPVISION y BP_GPU tienen una resolución de 8 bits con signo.
- CSBP, SGBM y ELAS tienen una resolución de 16 bits con signo.
- StereoVAR tiene una resolución de 32 bits en coma flotante.

En vista de los resultados, podemos concretar que **cuanto mayor sea el número de bits** para representar el valor numérico de la disparidad de cada píxel, mayor será la resolución de niveles de disparidad y, por tanto, se puede obtener un mapa más suave y preciso.

Volviendo al mapa de disparidad correspondiente a **BM_CPU**, que ha sido obtenido para 16 niveles de disparidad (n_{disp}) con valor mínimo -13 (d_{min}) y ancho de ventana de 11 píxeles (win_{size}), podemos observar que su principal zona conflictiva es la **superficie sin textura** del terreno de juego donde presenta agujeros. Existen problemas en los bordes de los jugadores, donde no asigna valores de disparidad. Además, se produce la situación de **oclusión parcial** de un jugador por otro que está más cerca de la cámara. En ese caso, ha asignado el mismo valor de disparidad a ambos jugadores.

El mapa de la figura 6.2(b), **BM_GPU** con los mismos parámetros que el anterior, presenta menos ruido en la **superficie sin textura**, que sigue siendo problemática, e incluso reconoce una de las líneas del terreno de juego. El resultado es mejor globalmente a pesar de tener menor resolución (de bit) y de que los saltos entre niveles de disparidad sean más bruscos, aunque también asigna el mismo plano de disparidad a dos jugadores parcialmente ocluidos. También se mejora el resultado en los bordes de los jugadores.

La optimización basada en **GC** ($n_{disp} = 16$, $d_{min} = -13$, $n_{iter} = 7$) se caracteriza por generar un mapa de disparidad formado por **planos fronto-paralelos**, efecto que se pone de manifiesto en la figura 6.2(c). Sin embargo no es un mapa ruidoso, sino que resulta bastante uniforme, respetando la **continuidad** a pesar de los saltos bruscos de disparidad entre los diferentes niveles, sin presentar problemas en la **zona sin textura**. No contiene demasiados *outliers* (grupo de píxeles con distinto valor de disparidad que sus vecinos) ya que el algoritmo busca la vecindad más grande posible a la que asignar la misma etiqueta. De nuevo, no distingue entre los dos jugadores parcialmente ocluidos.

Los mapas obtenidos a partir de las optimizaciones basadas en **BP** de las figuras 6.2(d) y 6.2(e) siempre son prácticamente **idénticos**, debido a que su implementación es exactamente la misma salvo que una de ellas está optimizada para GPU. Los parámetros utilizados en ambas son: $d_{min} = -12$, $n_{disp} = 20$, $n_{levels} = 1$ y $n_{iter} = 15$. El mapa es un punto intermedio entre GC y BM, aunque se parece más a este último porque también presenta problemas en la **zona sin textura**. Los bordes de los jugadores están, por

lo general, bien resueltos como en BM_GPU, confundiendo de nuevo a los jugadores parcialmente ocluidos.

La optimización **CSBP** produce un mapa **bastante ruidoso** con respecto a los anteriores, aunque es posible que se deba a las particularidades de la implementación utilizada. Los parámetros que dan lugar a este mapa son: $d_{min} = -12$, $n_{disp} = 12$, $n_{levels} = 1$ y $n_{iter} = 15$. Tras el ruido, se puede observar que diferencia bastante bien los **objetos** de la escena y que el **gradiente** de disparidad del fondo es suave. Al igual que los anteriores, presenta problemas en la **zona sin textura**.

El mapa de la figura 6.2(g), que se corresponde a **SGBM** con los parámetros $d_{min} = -14$, $n_{disp} = 16$, y $win_{size} = 9$, no presenta problemas en la **zona sin textura**. Es un mapa intermedio entre GC y BM. Globalmente es más suave, debido a las **restricciones de continuidad** que asume el método y la resolución de bit mayor. Presenta algunas zonas sin valor asignado en las **regiones ocluidas**, cerca de los bordes de los jugadores, pero no resulta demasiado problemático.

Como se ha explicado, **ELAS** tiene problemas en las **zonas sin textura**, donde asigna valores inválidos. Este hecho se comprueba en la figura 6.2(h) ($d_{min} = -13$, $n_{disp} = 20$), que presenta valores de disparidad sólo en los jugadores y la grada. El resto del mapa es bastante **suave** y se observa que este método tampoco es capaz de distinguir entre dos jugadores parcialmente ocluidos.

StereoVAR es el único método con resolución de 32 bits, hecho que se pone de manifiesto medante una **alta suavidad** en el mapa de la figura 6.2(i) ($d_{min} = -17$, $n_{disp} = 28$ y $n_{iter} = 25$). El resultado de esta técnica es bastante diferente del resto. No presenta problemas en la **zona sin textura** dando lugar a un gradiente de disparidad (la figura mostrada tiene bajo contraste debido a las pequeñas diferencias de distancia entre planos de profundidad), al igual que en el fondo. Es capaz de distinguir algunos jugadores parcialmente ocluidos.

Se puede observar que, los algoritmos que dependen de funciones de coste locales (BM_CPU, BM_GPU, SGBM y ELAS) presentan un **marco negro** (valores inválidos) en los laterales y/o en las zonas superior e inferior debido a que, por motivos de implementación, se ignoran las regiones que puedan requerir acceder a píxeles fuera de la imagen.

Durante la realización de los experimentos, hemos llegado a las siguientes conclusiones con respecto a la variación de los parámetros:

- Un valor mayor de n_{disp}, incrementa el **rango de disparidad**, permitiendo más valores intermedios que dan lugar a un mapa más preciso. Dependiendo de la profundidad de bit en la representación numérica de los datos, se percibirá más los saltos entre niveles de disparidad. Todo esto se traduce en un aumento del **contraste** del mapa de disparidad normalizado que se visualiza. Además la región negra que bordea el resultado de los algoritmos será mayor.
- La variación del **mínimo** d_{min} modifica la **intensidad** del mapa, manteniendo el contraste. Valores menores dan lugar a un mapa de disparidad aclarado. Además, el ancho del borde negro que rodea la imagen aumentará en un lateral mientras que disminuirá en el contrario.
- Disminuir el **tamaño de ventana**, win_{size}, da lugar a un mapa más ruidoso, aunque las regiones de píxeles inválidos son de menor tamaño. El incremento de este parámetro reduce el ruido al asignar el mismo valor de disparidad a un mayor número de píxeles, pero provoca que el marco de valores inválidos que rodean el mapa sea más ancho.
- El **número de iteraciones** (n_{iter}) de los algoritmos iterativos (GC, BP y StereoVAR) afectan positivamente sobre la **suavidad** del resultado. Un valor más alto reduce el ruido global del mapa de disparidad pero hay que evitar pasarse para no eliminar bordes importantes.
- Los algoritmos **multirresolución** que utilizan el parámetro n_{levels}, presentan **vecindades más grandes con el mismo valor de disparidad** asignado cuantos más niveles jerárquicos se procesen. Este procesado no respeta bordes, ya que se trabaja sobre versiones submuestreadas de la imagen original.

Por otra parte, analizando los mapas obtenidos por cada técnica, hay que tener en cuenta los **requisitos** concretos **de nuestra aplicación**. En concreto, se necesita un mapa denso que tenga valores de disparidad válidos sobre todo en el entorno de los objetos de la escena y no es necesario que dichos valores sean exactos, sino que es suficiente con conocer las **profundidades relativas** entre ellos. Un mapa ruidoso puede afectar bastante a la inserción de los objetos sintéticos interactivos, por lo que podríamos concluir que un mapa como el de **GC, SGBM o StereoVAR** sería lo más apropiado.

Como hemos visto, es muy complicado extraer una conclusión a partir de la comparación visual de los resultados de cada algoritmo, por lo que es necesario recurrir a criterios objetivos.

6.3.1.2. Calidad objetiva

Realizar una evaluación robusta del rendimiento de los algoritmos estéreo supone un gran reto, ya que cada uno de ellos utiliza unos parámetros distintos. Para hacer viable una comparación crítica, los parámetros deben ser, dentro de lo posible, los mismos para todos los algoritmos. Lograrlo es complicado, sobre todo tratando con algoritmos basados en distintos tipos de técnicas, como ocurre con los locales y globales, siendo difícil encontrar parámetros comunes a ambos. Por lo tanto, el objetivo es definir el conjunto común de parámetros más grande posible.

A continuación vamos a analizar los resultados de los experimentos realizados utilizando los mismos parámetros para todos los algoritmos de correspondencia. Esto facilita la toma de decisión acerca del método más apropiado para resolver nuestro problema.

Selección de parámetros.

Durante el análisis subjetivo, se ha encontrado empíricamente el valor de los parámetros estéreo que, en media, producían los mejores resultados para la mayoría de los algoritmos. De aquí en adelante, todos los resultados han sido obtenidos utilizando los siguientes parámetros:

- **Disparidad mínima**, $d_{min} = -13$ píxeles.
- **Niveles de disparidad** que definen el rango de disparidad, $n_{disp} = 16$ niveles.
- **Ancho de ventana** de los algoritmos locales, $win_{size} = 11$ píxeles.
- **Número de iteraciones** de los algoritmos iterativos, según la velocidad de convergencia de la solución, $n_{iter} = 10$ para las optimizaciones basadas en *belief propagation* (BPVISION, BP_GPU y CSBP) y $n_{iter} = 25$ para StereoVAR y GC.
- **Número de niveles** jerárquicos de los algoritmos multirresolución, $n_{levels} = 1$.

Resumen de resultados.

La tabla 6.1 recoge un resumen comparativo de resultados obtenidos aplicando cada uno de los algoritmos, con los parámetros definidos, sobre algunas de las capturas arriba descritas. Para cada imagen, se muestran los dos indicadores de error global: RMS y PSNR, y la tasa de píxeles erróneos P_e. Se ha resaltado en negrita los valores mejores.

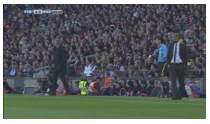
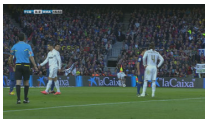
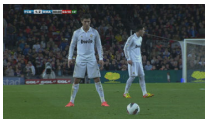
	Captura 1 			**Captura 4** 			**Captura 5** 			**Captura 6** 		
	RMS	PSNR	P_e	RMS	PSNR	P_e	RMS	PSNR	P_e	RMS	PSNR	P_e
BM_CPU	71.5	11.04	15.21	73.22	10.84	1.386	70.55	11.16	11.53	57.61	12.92	6.758
BM_GPU	58.7	12.76	16.86	72.41	10.94	1.482	54.44	13.41	13.08	49.28	14.28	7.89
GC	**16.65**	**23.7**	30.25	**7.511**	**30.62**	6.505	**16.78**	**23.64**	25.53	**13.13**	**25.77**	15.48
BPVISION	33.89	17.53	61.68	23.85	20.58	43.91	34.56	17.36	53.02	32.62	17.86	48.77
BP_GPU	33.87	17.53	61.68	23.81	20.59	43.71	34.63	17.34	53.09	32.49	17.9	48.69
CSBP	33.71	17.57	61.8	23.26	20.8	41.41	34.46	17.38	52.7	32.14	17.99	48.1
SGBM	43.65	15.33	16.46	34.84	17.29	1.542	29.33	18.79	12.4	29.26	18.81	7.479
ELAS	77.98	10.29	**13.67**	86.46	9.394	**1.332**	88.42	9.199	**9.887**	76.6	10.45	**5.853**
StereoVAR	25.52	19.99	53.56	9.856	28.26	11.11	24.85	20.22	45.13	17.44	23.3	33.16

TABLA 6.1 – Resumen comparativo de los resultados obtenidos con algunas capturas.

Como explicábamos en la sección 4.6.2, los valores de **RMS y PSNR** están relacionados de forma inversa, de manera que cuanto menor sea el error global representado por RMS, la PSNR será mayor. Por lo tanto, podemos establecer la **maximización de la PSNR** como primer criterio de selección del algoritmo más adecuado. Se observa que, como se esperaba, GC es el algoritmo con menor error global (mínimo RMS y máximo PSNR) en comparación con el resto. Esto es debido a las restricciones globales que el método considera, cuyo objetivo es minimizar una función de energía (coste o error) sobre toda la imagen.

Por otra parte, la **tasa de píxeles erróneos** depende de la tolerancia que permitamos según el problema concreto. Esta tasa viene dada por la **cantidad de píxeles con valor de disparidad válido**, cuyo valor en la imagen reconstruida **supere un**

umbral de error con respecto a la imagen de referencia. Se puede medir también, contando el número de píxeles de la imagen diferencia normalizada que superen un umbral de intensidad. Lejos de cualquier pronóstico, ELAS minimiza la tasa de píxeles erróneos para todas las capturas. Esto se debe a que P_e sólo tiene en cuenta los píxeles con valor de disparidad asignado, por lo que concluimos que esta técnica es bastante precisa en las regiones donde puede calcular la disparidad. De hecho, una **tasa de píxeles erróneos baja no implica que el error global sea menor**, ya que como se demuestra en la misma tabla, ELAS tiene el RMS más alto (PSNR más bajo). Esto se debe a que, como hemos visto, presenta graves problemas en las áreas sin textura, donde no asigna valores de disparidad (figura 6.2). Para tener en cuenta todos los píxeles del mapa de disparidad, es necesario añadir al análisis la tasa de píxeles inválidos. Por lo tanto, podemos establecer la **minimización de la tasa de píxeles incorrectos (inválidos + erróneos)** como segundo criterio de selección del algoritmo más adecuado.

Para concluir este apartado hay que comentar que los criterios establecidos para la selección del algoritmo más adecuado pueden a su vez favorecer la búsqueda de los parámetros comunes, mediante un proceso de realimentación. Es decir, buscando la maximización de PSNR o minimización de P_{err}, se puede llegar a encontrar unos parámetros de correspondencia que den lugar a resultados subjetivos mejores.

Maximización de PSNR.

La tabla 6.2 presenta una comparación de la PSNR que figura en la tabla 6.1 con respecto a la PSNR obtenida con **parámetros optimizados**, observando que **la PSNR es mayor** en este último caso. Para cada técnica, la fila superior muestra los valores de PSNR obtenidos con los parámetros comunes, mientras que en la fila inferior se representan **los valores optimizados de PSNR**. Se ha resaltado en negrita el valor más alto.

Se observa que en algunos casos coinciden los valores de ambas filas, ya que los parámetros comunes han sido determinados empíricamente en función del conjunto de resultados subjetivos. Por ejemplo, para la captura 1, los métodos BM_CPU, BP_GPU y ELAS devuelven el mejor resultado con los parámetros de nuestro experimento. Lo mismo ocurre en la captura 4 aplicando ELAS, en la captura 5 con BM_CPU, CSBP y ELAS, y en la captura 6 con SGBM y ELAS.

	Captura 1	**Captura 4**	**Captura 5**	**Captura 6**
BM_CPU	11.04	10.84	11.16	12.92
	11.04	11.26	11.16	12.97
BM_GPU	12.76	10.94	13.41	14.28
	12.89	12.32	14.56	15.3
GC	**23.7**	**30.62**	**23.64**	**25.77**
	23.71	**30.68**	**23.67**	**26.03**
BPVISION	17.53	20.58	17.36	17.86
	17.6	22.66	17.53	18.92
BP_GPU	17.53	20.59	17.34	17.9
	17.53	22.65	17.48	18.6
CSBP	17.57	20.8	17.38	17.99
	17.57	22.44	17.38	18.14
SGBM	15.33	17.29	18.79	18.81
	16.41	18.21	18.81	18.81
ELAS	10.29	9.394	9.199	10.45
	10.29	9.394	9.199	10.45
StereoVAR	19.99	28.26	20.22	23.3
	22.21	28.73	23.32	25.74

TABLA 6.2 – **Comparación de PSNR con el caso mejor:** para cada técnica, la fila superior muestra la PSNR con los parámetros comunes de la tabla 6.1, mientras que la fila inferior presenta los valores optimizados de PSNR. En negrita, el mejor resultado.

	Captura 1	**Captura 4**	**Captura 5**	**Captura 6**
BM_CPU	35.84	22.026	29.72	18.338
BM_GPU	30.48	21.722	23.42	16.199
GC	30.25	6.505	25.53	15.48
BPVISION	61.68	43.91	53.02	48.77
BP_GPU	61.73	43.71	53.09	48.69
CSBP	61.8	41.41	52.7	48.1
SGBM	**23.948**	**6.103**	**15.437**	**10,393**
ELAS	38.23	30.142	38.677	26.693
StereoVAR	53.56	11.11	45.131	33.16

TABLA 6.3 – **Tasa de píxeles incorrectos (%).**

	Captura 1	**Captura 4**	**Captura 5**	**Captura 6**
BM_CPU	137	127.6	134.5	131.7
BM_GPU	**168.7**	**161.3**	**163**	**166.4**
GC	0.1197	0.1108	0.0643	0.1338
BPVISION	1.173	1.059	1.11	1.077
BP_GPU	3.996	4.007	3.974	3.978
CSBP	14.13	14.31	14.21	14.13
SGBM	29.7	28.98	28.42	28.68
ELAS	8.397	7.588	7.852	7.711
StereoVAR	4.515	4.717	4.468	4.314

TABLA 6.4 – **Tasa de cuadros por segundo (fps).**

Hemos comentado en el apartado anterior que GC es el mejor resultado porque se trata de una **técnica** global cuya función de energía toma restricciones de orden y suavidad entre píxeles adyacentes, de manera **que** se **minimiza el error global** (tabla 6.1). En la tabla 6.2 vemos que, aunque los resultados no coinciden con el caso mejor, se aproximan bastante. Un algoritmo de optimización global como GC tiene la ventaja de permitir una **variación de parámetros** relativamente **flexible** manteniendo un buen comportamiento frente al error.

La PSNR (o RMS) se calcula a partir de la imagen de referencia reconstruida con los píxeles inválidos marcados de un color que difiere bastante del valor original correspondiente en la imagen de referencia. Como consecuencia, la maximización de la **PSNR no es independiente del resto de estadísticas de error**, como la tasa de píxeles inválidos. Por este motivo, hay que tener en cuenta el análisis de la tasa de error, que se explica a continuación.

Tasa de píxeles incorrectos: P_{err}.

La tasa de píxeles incorrectos (P_{err}) es la suma de la **tasa de píxeles inválidos** (P_i) y la **tasa de píxeles erróneos** (P_e). Recordamos que definimos un píxel como inválido cuando el algoritmo de correspondencia no es capaz de determinar el valor de disparidad, mientras que un píxel es erróneo cuando el valor reconstruido supera un umbral de error respecto al de referencia.

Al tratarse de una suma, la tasa de píxeles incorrectos se verá incrementada en caso de aumentar el valor de alguno de sus sumandos, manteniendo constante el otro. Por lo tanto, puede darse la situación de dos algoritmos muy distintos con P_{err} similar, pero uno de ellos presentar un porcentaje muy alto de píxeles inválidos, mientras que el otro tener una tasa muy alta de píxeles erróneos frente a inválidos. Por lo tanto, hay que tomar una **decisión de compromiso entre ambas tasas de píxel**.

En nuestra aplicación particular, nos interesa obtener un **mapa de disparidad lo más denso posible**, es decir, que todos los píxeles de la imagen tengan un valor de disparidad válido asignado. Sin embargo, **no es imprescindible que** los valores de disparidad **sean demasiado precisos**, ya que es más importante distinguir de forma relativa los planos de profundidad que centrarse en su valor real. Esto se traduce en la

minimización de la tasa de píxeles inválidos a costa de permitir un incremento (preferiblemente pequeño) del porcentaje de píxeles erróneos.

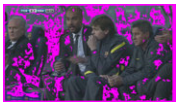
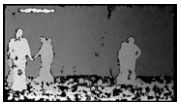
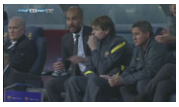
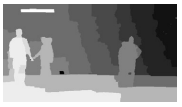
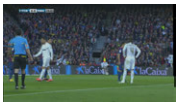
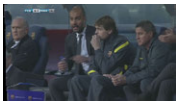
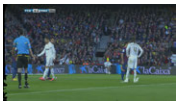
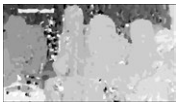
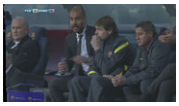
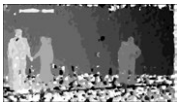
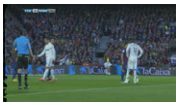
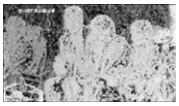
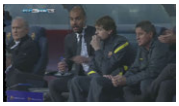
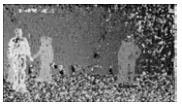
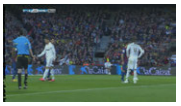
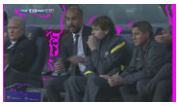
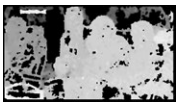
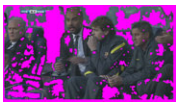
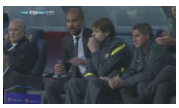
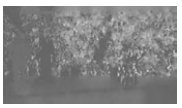
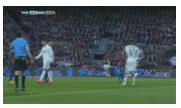
En la tabla 6.3 se resume la tasa de píxeles incorrectos resultante de nuestro experimento, resaltando en negrita el mejor resultado. **SGBM** es la técnica que **minimiza la tasa de píxeles incorrectos**. Es más, si se consulta la tasa de píxeles erróneos en la tabla 6.1, se puede comprobar que la proporción es mucho mayor frente a la tasa de píxeles inválidos. Así, para la captura 1, con el 23,948 % de píxeles incorrectos, el 16,46 % se corresponde con píxeles erróneos dejando un 7,488 % de píxeles inválidos. De manera similar ocurre con el resto de capturas, aunque existe una excepción en la captura 4, donde el porcentaje de píxeles inválidos es 4,561 % frente a 1,332 % de erróneos, aunque el total es suficientemente bajo como para admitirlo. Esta captura se diferencia de las anteriores, porque está tomada a una distancia más cercana y cuenta con menos regiones de baja textura.

En cuanto al resto de algoritmos, comparando el valor de P_{err} de la tabla 6.3 con el de P_e en la tabla 6.1, los **métodos de correspondencia de bloques** (BM_CPU y BM_GPU) tienen una proporción similar de píxeles inválidos e incorrectos. Esto es, del total de P_{err}, aproximadamente la mitad es P_i y la otra mitad P_e. Las **técnicas basadas en optimización global** (GC, BPVISION, BP_GPU y CSBP) no devuelven valores de disparidad inválidos, por lo que P_i es cero y toda la tasa de píxeles incorrectos se corresponde con píxeles erróneos. Dentro de estas técnicas, GC es la que minimiza el error, ya que los métodos basados en *belief propagation* generan mapas bastante ruidosos. En una situación similar se encuentra **StereoVAR**, que proporciona una tasa de píxeles inválidos muy próxima a cero. Por último, **ELAS** presenta una tasa de píxeles inválidos muy alta porque, como hemos comentado, tiene problemas en las zonas sin textura y, en el caso de la captura 4, en la zona del fondo, siendo bastante preciso sobre los objetos de la escena.

6.3.1.3. Calidad subjetiva frente a calidad objetiva

En la tabla 6.7 se presenta un resumen de los resultados obtenidos al aplicar cada uno de los algoritmos de correspondencia sobre las capturas 4 y 5 utilizando los

parámetros definidos para los experimentos de calidad objetiva. En este caso, visualizamos el mapa de disparidad junto con la imagen de referencia reconstruida con los píxeles inválidos marcados, y el valor de la tasa de píxeles inválidos.

Algoritmo	Mapa	Reconstrucción	P_i	Algoritmo	Mapa	Reconstrucción	P_i
BM_CPU			20.64	BM_CPU			18.19
BM_GPU			20.24	BM_GPU			10.34
GC			**0**	GC			**0**
BPVISION			**0**	BPVISION			**0**
BP_GPU			**0**	BP_GPU			**0**
CSBP			**0**	CSBP			**0**
SGBM			4.561	SGBM			3.037
ELAS			28.81	ELAS			28.79
StereoVAR			0	StereoVAR			0.001

Captura 4 **Captura 5**

TABLA 6.7 – **Resumen de resultados.** Para cada algoritmo se muestra el mapa de disparidad, la imagen de referencia reconstruida, con los píxeles inválidos marcados en magenta, y el porcentaje de píxeles inválidos.

En los **algoritmos que utilizan ventana de agregación**, se observa que gran parte de los píxeles inválidos se concentran en el borde que rodea la imagen, que se produce por motivos de implementación. En este sentido, la elección de los parámetros es determinante, porque un menor tamaño de ventana reducirá ese borde, disminuyendo P_i, a costa de generar un mapa más ruidoso. Por lo que se podría mejorar la tasa P_{err} sobre todo en los algoritmos BM_CPU y BM_GPU. Sin embargo, en nuestro caso interesa más un tamaño de ventana mayor que reduzca el ruido. Análogamente, un mayor **número de iteraciones** suaviza el ruido aunque no preserve los detalles.

Por otra parte, los algoritmos basados en optimización global tienen una P_i con valor cero porque devuelven un valor de disparidad válido para todos los píxeles de la imagen. Como consecuencia, se tendrá una imagen de referencia reconstruida bastante densa. En una situación similar se encuentra StereoVAR, con una P_i muy próxima a cero.

ELAS es un caso excepcional, donde la tasa de píxeles inválidos es máxima porque sólo es capaz de calcular la disparidad en zonas con mucha textura. Consecuentemente, se obtiene una imagen de referencia reconstruida incompleta.

6.3.2 Eficiencia

La eficiencia de los algoritmos de correspondencia viene dada por su capacidad de **procesar las imágenes en el menor tiempo posible**. Es decir, para comparar la eficiencia, medimos la **tasa de cuadros por segundo** que cada método puede computar. Se mide en fps (*frames per second*) y es inversamente proporcional al coste computacional y complejidad del algoritmo. El resultado depende del hardware sobre el que se ejecuta el proceso. Así, la utilización de la GPU puede incrementar enormemente la eficiencia de los algoritmos basados en CPU.

En la tabla 6.4 se recogen los resultados correspondientes a la ejecución de los algoritmos en el equipo empleado para la realización del experimento. El equipo consta de un procesador *Intel® Core™ 2* (2,67 GHz), 4 GB de memoria RAM y una tarjeta gráfica *NVidia® GeForce® 9600 GT™*. El harware descrito es relativamente antiguo (5 años aproximadamente), por lo que los resultados se pueden mejorar únicamente utilizando un equipo más moderno del laboratorio.

Como se puede comprobar, la tasa obtenida por cada algoritmo permanece prácticamente constante para todas las capturas. La tasa de cuadros por segundo mínima requerida por una aplicación en **tiempo real** son 25 fps, por lo tanto, las únicas técnicas que superan este umbral son BM_CPU, BM_GPU y SGBM. Como era de esperar, los métodos basados en optimización global tienen una eficiencia menor debido a su elevada complejidad, alcanzando el mínimo con GC. Vemos también que la optimización de BP para GPU (BP_GPU) duplica la tasa de BPVISION en CPU, siendo el mismo algoritmo, y que la reducción de coste computacional por parte de CSBP triplica la capacidad de procesamiento. ELAS y StereoVAR alcanzan unas tasas similares a las de los métodos basados en BP.

BM_GPU es el método que alcanza mayor eficiencia, con una tasa de cuadros por segundo comprendida entre 161 y 169 fps, seguido de cerca por **BM_CPU**, entre 127 y 137 fps. Estos resultados son más que suficientes para una aplicación en tiempo real. De hecho, permiten incluir etapas de preprocesado y posprocesado para mejorar los resultados de calidad, siempre y cuando estos procesos no añadan demasiado retardo. **SGBM** se encuentra en el límite del tiempo real, con tasas comprendidas entre 28 y 30 fps. Sin embargo, este algoritmo está implementado para CPU, por lo que una posible optimización para GPU lograría incrementar la eficiencia manteniendo la calidad de este tipo de técnica.

6.4 Conclusiones

Para finalizar el capítulo debemos seleccionar el algoritmo que más se adecúa a nuestro problema atendiendo a todos los criterios explicados anteriormente. A modo de resumen, enumeramos las conclusiones tomadas a lo largo de los experimentos:

- En el **análisis subjetivo de calidad** llegamos a la conclusión de que lo más adecuado es escoger una técnica que dé lugar a un **mapa de disparidad** lo más **denso** posible, es decir, que sea capaz de calcular la disparidad para el máximo número de píxeles de la imagen. Además, se debe **evitar un mapa ruidoso**, por lo que buscamos un algoritmo que genere un **mapa suave**. Finalmente, debe conocerse las profundidades relativas entre los distintos objetos de la escena sin ser necesario que los valores de disparidad sean exactos. Los algoritmos que parecen más apropiados según estos requisitos son: **GC, SGBM y StereoVAR**.

- En el **análisis objetivo de calidad** hemos establecido dos criterios de selección:
  - **Maximización de la PSNR (o minimización de RMS):** de las tablas 6.1 y 6.2 extraemos que **GC minimiza el error global** debido a su planteamiento como minimización de una función de energía con restricciones de orden y suavidad sobre toda la imagen. Sin embargo, la PSNR depende del resto de estadísticas de error.
  - **Minimización de la tasa de píxeles incorrectos (P_{err})**: se debe alcanzar una decisión de compromiso entre P_i y P_e, ya que por ejemplo un método como **ELAS** es bastante preciso en las regiones donde es capaz de calcular la disparidad (P_e mínima según la tabla 6.1) a costa de tener una altísima tasa de píxeles inválidos (tabla 6.7). Para nuestra aplicación, es más importante minimizar la tasa de píxeles inválidos P_i, para obtener un mapa denso permitiendo cierta tolerancia al error. De la tabla 6.3, extraemos que **SGBM** minimiza la tasa de píxeles incorrectos con una tasa de píxeles erróneos (P_e) aceptable según la tabla 6.1, logrando un mapa bastante denso por su baja tasa de píxeles inválidos (P_i).
- En el **análisis de eficiencia** veíamos que, en el contexto de las aplicaciones en **tiempo real**, se requiere como mínimo una capacidad de procesamiento de 25 cuadros por segundo. De todos los algoritmos, solo tres cumplen este requisito: **BM_GPU y BM_CPU** superan los 160 y 130 fps, respectivamente, mientras que **SGBM** está en el límite del tiempo real con unos 30 fps. Además hay que tener en cuenta el hardware sobre el que se ejecutan los algoritmos, ya que se puede aumentar la eficiencia utilizando un equipo más moderno.

Viendo las conclusiones podemos afirmar que **SGBM** cumple la mayoría de ellas. Se trata de un algoritmo que genera un mapa de disparidad denso, suave y continuo, sin agujeros y visualmente agradable donde se puede distinguir los diferentes planos de profundidad sin saltos bruscos. El único requisito que, a priori, no cumple es la maximización de la PSNR. Sin embargo, se acepta por alcanzar un valor medio-alto situado por debajo de los métodos globales y por encima de los locales. Además, esta técnica logra un compromiso entre la tasa de píxeles inválidos (mínima) y la tasa de píxeles erróneos (aceptable) con una proporción de inválidos muy inferior a la de erróneos. Finalmente, alcanza una tasa de cuadros por segundo aceptable para aplicaciones en tiempo real, permitiendo una posible optimización para GPU o utilización de hardware más avanzado

para incrementar dicha tasa.

La figura 6.3 muestra los mapas de disparidad obtenidos aplicando el algoritmo SGBM sobre las capturas de nuestros experimentos. Se puede comprobar subjetivamente que se comporta bastante bien en todas las situaciones (escenas cercanas, lejanas, zonas sin textura o bordes de los objetos).

Para concluir, tenemos que comentar que el resultado alcanzado es coherente y se corresponde con lo teóricamente esperado, ya que una técnica de tipo semiglobal reúne algunas virtudes de los métodos basados en optimización global junto con otras de los algoritmos locales, suponiendo un punto intermedio entre eficiencia y calidad.

(a) Captura 1 (b) Captura 2 (c) Captura 3

(d) Captura 4 (e) Captura 5 (f) Captura 6

FIGURA 6.3 – Mapas de disparidad obtenidos por el algoritmo SGBM sobre las capturas utilizadas en los experimentos.

Capítulo 7

Presupuesto

7.1 Introducción

En este capítulo se presenta el presupuesto asociado al desarrollo íntegro de este Proyecto de Fin de Carrera. Como hemos explicado, este trabajo surgió inicialmente para buscar la solución de una parte de un proyecto más amplio (*ImmersiveTV*), pero tras la finalización, ha sido extendido en el tiempo de manera independiente con un objetivo más amplio de realizar un estudio más detallado sobre el comportamiento de las técnicas y algoritmos de correspondencia estéreo. A continuación se muestra un desglose de los distintos tipos de coste.

7.2 Costes del proyecto

La tabla 7.1 recoge los costes del proyecto desglosados según el tipo. El trabajo comenzó en marzo de 2012 y ha finalizado en diciembre de 2013, por lo que se consideran diez meses del primer año y el segundo año completo. La realización del presente PFC ha estado a cargo de un becario, con una dedicación de media jornada (436€) hasta junio de 2013 (incluido) y, a partir de julio con dedicación de jornada completa (872€) durante los 6 últimos meses. La suma del importe de la beca durante los periodos descritos es el **coste de personal**.

$$\text{Coste de Personal} = 436€/\text{mes} \cdot (10 + 6)\ \text{meses} + 872€/\text{mes} \cdot 6\ \text{meses} = 12.208€$$

Conceptos	Descripción	2012		2013		Coste Total
		Dedicación	*Coste 2012 (10 meses)*	*Dedicación*	*Coste 2013 (12 meses)*	
Costes de Personal	Becario PFC: Eduardo Sánchez	50 %	4.360€	50 % (M1-M6) 100 % (M7-M12)	7.848€	12.208€
Pequeño Equipamiento (amortización)	Estación de trabajo + tarjeta gráfica de altas prestaciones (amortización a 2 años)	100 %	375€	100 %	450€	825€
Material Fungible	Renovación anual de licencia de software *Microsoft Visual Studio* y *Matlab*®	100 %	0€	100 %	0€	0€
Viajes y dietas	Costes de viaje y dietas para asistencia a conferencia científica especializada para difusión de resultados del proyecto	100 %	0€	100 %	1.200€	1.200€
Otros Gastos	Gastos inscripción en conferencia científica	100 %	0€	5 %	600€	600€
Costes Indirectos	Costes generales de funcionamiento (57,88 % de los costes de personal)	100 %	2.523,57€	100 %	4.542,42€	7.065,99€
TOTAL		**7.258,57€**		**14.640,42€**		**21.898,99€**

TABLA 7.1 – **Presupuesto del proyecto.**

El **activo no corriente inmovilizado material** utilizado ha sido una estación de trabajo con tarjeta gráfica de altas prestaciones, que suponen un coste de pequeño equipamiento, con las especificaciones siguientes:

- Procesador *Intel® Core™ 2* (2,67 GHz).
- Memoria RAM de 4 GB.
- Tarjeta gráfica *NVidia® GeForce® 9600 GT™*.

Se considera dedicación completa, puesto que equipo ha estado dedicado exclusivamente al desarrollo del proyecto. Los **costes de pequeño equipamiento** corresponden a la **amortización** a dos años de los equipos descritos, valorados en unos 900 euros, durante el periodo considerado de 10 y 12 meses:

$$\text{Pequeño equipamiento} = 900€ \cdot \frac{10 \text{ meses}}{24 \text{ meses}} + 900€ \cdot \frac{12 \text{ meses}}{24 \text{ meses}} = 825€$$

El **activo no corriente inmovilizado intangible** se corresponde con las aplicaciones informáticas y las licencias de software utilizadas. Para el desarrollo de este proyecto se ha utilizado el siguiente software:

- Sistema operativo *Microsoft™ Windows 7 Enterprise 64 bits* (Requiere licencia).
- Entorno de desarrollo *Microsoft™ Visual Studio Professional 2010* [170] (Requiere licencia).
- Entorno de cálculo numérico *Matlab®* versión R2011a (Requiere licencia).
- Librería OpenCV versión 2.4.0 (licencia de software libre BSD de código abierto).
- Librería NVidia® CUDA™ versión 4.2 [153](Licencia de producto distribuible (*Licensed Deliverables*)).
- Librería Qt versión 4.8.0 [172, 173] (Licencia gratuita GNU LGPL).
- Entorno de simulación Linux para Windows *Cygwin™* (Licencia gratuita GNU GPL de software libre).
- Librería de evaluación de algoritmos de correspondencia estéreo densa de Middlebury [16, 23] (Acuerdo de licencia gratuita de Microsoft Research MSR-SCLA).
- Librería de minimización MRF [20] (no requiere licencia, requiere cita de trabajos originales).
- Librería libELAS [149, 150] (Licencia gratuita GNU GPL de software libre).
- Librería bp-vision [144, 151] (Licencia gratuita GNU GPL de software libre).

De todo el software descrito, solamente requiere licencia el sistema operativo, el entorno de desarrollo *Visual Studio* y *Matlab*. Sin embargo, estas licencias pertenecen a la Universidad Politécnica de Madrid que las pone a disposición de la comunidad universitaria. Por lo tanto, los costes asociados a la renovación anual de licencias no son imputados al proyecto.

Como consecuencia del trabajo realizado en este proyecto, se prevé realizar una **publicación de resultados** en una conferencia o revista durante el año 2014. Para lo cual, en caso de ser aceptada, se estima un coste de inscripción de 600€además del coste del viaje necesario para realizar la presentación. El coste conjunto estimado asociado a la difusión de los resultados asciende a 1.800€, que se contabiliza en el año 2013 porque es el periodo en el que se ha desarrollado dicho trabajo.

Por último, hay que añadir unos **costes generales de funcionamiento**. Se trata de una tasa específica de la universidad aplicada a las diferentes escuelas que forman parte de ella. Así, para la ETSIT supuso en el año 2012 el 57,88 % de los costes de personal, unos 7.065,99€.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1 Lecciones aprendidas y conclusiones

A continuación resumimos las principales ideas y conclusiones extraidas a lo largo de los capítulos que componen esta memoria.

Uno de los objetivos de los productores de contenido multimedia es **aumentar la inmersividad** del usuario. Por este motivo, durante los últimos años se ha extendido la utilización de contenido 3D, sobre todo en los cines. Pero aún queda bastante por hacer en el área de difusión de contenidos de televisión 3D para los hogares. Estándares como DVB-3DTV y HDMI v1.4a, surgieron inicialmente para garantizar la **compatibilidad con la infraestructura de difusión HD existente**, de manera que es una buena decisión apostar por formatos *frame compatible*. Concretamente en el proyecto *ImmersiveTV* **se utiliza el formato *Side-by-Side***.

Otro reto es **aumentar la interactividad** para que el espectador deje de ser un elemento pasivo y se convierta en un agente activo capaz de modificar la escena que visualiza. Para lograrlo, en el proyecto *ImmersiveTV* se ha desarrollado un **módulo de inserción de objetos sintéticos interactivos** que requiere información de profundidad de la escena para calcular las oclusiones con los diferentes elementos de la misma en tiempo real. Se trata de una aplicación que se ejecuta en el receptor del usuario cuando se produce un evento determinado, mostrando un objeto con el que el usuario puede interactuar dentro de la escena.

En el contexto de la televisión 3D en tiempo real, existen varias **maneras de obtener la información de profundidad**. Por un lado, se puede capturar el contenido utilizando **cámaras de profundidad y emplear un formato de transmisión como el *2D + Depth*** que empaqueta una vista junto con la información de profundidad. Esto permitiría simplificar los equipos en el lado del usuario, sin embargo, la mayoría de dispositivos de visualización utilizan formatos *frame compatible* por lo que es más adecuado mantener el formato de transmisión *Side-by-Side*. Por otra parte, se puede obtener la información de profundidad a partir de dos vistas de la escena utilizando **técnicas de correspondencia estéreo o *stereo matching***. La información de profundidad se puede calcular tanto en transmisión como en recepción. Realizar este proceso en el lado del transmisor implica modificar parte de la infraestructura existente de televisión, ya que sería necesario transmitir la información adicional incrementando el ancho de banda necesario y generando mayor retardo. Por lo tanto, **es preferible realizar la correspondencia estéreo en el lado del receptor**.

El primer requisito de los algoritmos de correspondencia es que las imágenes de entrada estén convenientemente alineadas horizontalmente, cumpliendo la restricción epipolar. Se ha comprobado que **el contenido de prueba está rectificado**, sin embargo esto no es lo habitual. El hecho de contar con un contenido alineado correctamente depende de la configuración de cámaras utilizadas en la captura, es decir, tanto de la distorsión que introducen las ópticas (matriz de parámetros intrínsecos), como de la posición relativa de las cámaras en el espacio (matriz esencial), o del correcto alineamiento vertical del *rig* o soporte estéreo.

La **rectificación** es un proceso muy costoso y poco preciso que dificulta el trabajo en tiempo real. Puesto que nuestro contenido está alineado, **no ha sido necesario aplicar rectificación**. Por lo general, en la difusión de televisión 3D no se dispone de la información de calibración de todas las cámaras utilizadas, por lo que es recomendable considerar la **rectificación no calibrada**. Algoritmos como el de Hartley, RectifKitU o *Uncalibrated Stereo Vision* (explicados en el capítulo 3) son ejemplos interesantes.

La **selección de un algoritmo de correspondencia** depende de las **características del contenido** utilizado, que en nuestro caso se trata de imágenes o vídeo real de alta resolución de un evento en directo, con objetos en movimiento y cambios

rápidos entre puntos de vista. En ocasiones se tienen capturas situadas cerca de la escena, mientras que otras capturas son lejanas. Cuanto mayor es la resolución, mayor es el tiempo de procesado o retardo y el coste computacional. Los movimientos rápidos generan error residual y los cambios repentinos entre puntos de vista afectan al rango de disparidad. La distancia con respecto de la escena puede dar lugar a valores de disparidad pequeños o indetectables. Por último, las condiciones de iluminación de un evento en directo no están controladas y los fenómenos meteorológicos influyen sobre el resultado al producirse reflejos, sombras y regiones con sobreexposición.

Las **técnicas de correspondencia estéreo** se pueden clasificar mayoritariamente en dos grupos: **locales** y **globales**, aunque existen otros. Las técnicas basadas en **optimización global** obtienen un mapa denso (todos los píxeles de la imagen tienen valor de disparidad), y por lo general, el resultado es más preciso, ya que la función de energía que minimizan asume restricciones de suavidad y continuidad sobre la imagen completa. El principal inconveniente es el elevado coste computacional y el tiempo de proceso. Por otra parte, los **métodos locales** son más rápidos pero menos precisos, asumiendo restricciones a nivel local sobre un conjunto de píxeles vecinos, ignorando el resto. Por lo tanto, **es necesario alcanzar un compromiso entre calidad y eficiencia**. El tipo de **técnica de correspondencia semiglobal** resulta de especial interés, ya que trata de reunir ventajas de ambos grupos, minimizando una función global y asumiendo restricciones a nivel local.

La librería **OpenCV** es bastante completa porque cuenta con implementaciones de algoritmos de correspondencia de varios tipos, facilitando la integración y comparación dentro de un mismo entorno. Dispone de técnicas locales, globales y semiglobales, además de algunas versiones basadas en GPU.

El módulo de inserción de objetos requiere un **mapa de disparidad denso** pero **no es necesario que sea preciso**. Es decir, es deseable que todos los píxeles de la imagen tengan un valor de disparidad válido asignado, pero no tiene que ser exacto. Es suficiente con poder **distinguir entre los diferentes planos de profundidad** de la escena para facilitar el cálculo de las oclusiones con los distintos objetos. Esto admite cierta **tolerancia de error**.

En este proyecto **hemos desarrollado una herramienta para evaluar algoritmos de correspondencia estéreo** sobre imágenes, secuencias de imágenes y vídeo, implementada en *Visual C++*, usando OpenCV y con GUI basada en Qt. Para elaborar un **entorno común de evaluación** de algoritmos de correspondencia, primero hay que encontrar el **conjunto de parámetros comunes** más grande posible. Nuestra herramienta permite seleccionar el valor de disparidad mínima (d_{min}), el número de niveles de disparidad (n_{disp}), el ancho de ventana de agregación (win_{size}), el número de iteraciones (n_{iter}) y el número de niveles jerárquicos (n_{levels}). También **se ha realizado la adaptación de un conjunto de algoritmos** para añadirlos al entorno, además de **modificar aquellos que no admiten valores negativos de disparidad**.

Según los puntos anteriores, una adecuada **selección del algoritmo y de los parámetros** determina el resultado, ya que, dentro de cada uno de los tipos de técnicas comentados, existen diferentes algoritmos de correspondencia que tienen un comportamiento distinto en las **regiones problemáticas** (sin textura, cerca de bordes de objetos o discontinuidades, y oclusiones) debido a las **diferentes restricciones** que aplican (semejanza, unicidad, suavidad u orden).

Un **análisis subjetivo**, mediante inspección visual del mapa obtenido por cada técnica aplicada sobre la misma captura, permite concluir que los algoritmos más apropiados son **GC, SGBM y StereoVAR**, ya que generan un **mapa suave y poco ruidoso**. Además, se ha podido comprobar que la zona más problemática es aquella con ausencia de textura debido a la presencia de múltiples ambigüedades, por lo que se deduce que este tipo de regiones suponen el mayor reto en la investigación de *stereo matching*.

Encontrar un **criterio objetivo** fiable para la **evaluación de la calidad de algoritmos** tan diferentes entre sí **sin** disponer de un ***ground truth* de referencia** es un desafío, y obliga a buscar medios indirectos de comparación. En este trabajo, hemos propuesto la **reconstrucción de la imagen de referencia** (vista izquierda) a partir de la vista derecha y el mapa de disparidad calculado. Comparando la imagen reconstruida con la original, se obtienen dos **medidas de error global: RMS y PSNR**. Con ayuda de la **imagen de error normalizada**, entre la reconstruida y la original, se calculan dos estadísticas más: **tasa de píxeles inválidos** (P_i) y **tasa de píxeles erróneos** (P_e), cuya suma es la **tasa de píxeles incorrectos** (P_{err}).

Hemos establecido dos **criterios** en el análisis objetivo de la calidad:

- Atendiendo a la **maximización de la PSNR** (o minimización de RMS), se obtiene que **GC** minimiza el error global.
- Atendiendo a la **minimización de la tasa de píxeles incorrectos (P_{err})** y considerando que se debe minimizar P_i admitiendo un márgen de P_e, se obtiene que **SGBM** genera un mapa bastante denso debido a su baja tasa de píxeles inválidos.

El otro aspecto a tener en cuenta es la **eficiencia**. En el contexto de las aplicaciones en tiempo real se debe alcanzar **como mínimo una tasa de 25 cuadros por segundo (fps)**. La eficiencia **depende del hardware** sobre el que se ejecutan los algoritmos, ya sea por las características del equipo de trabajo (memoria RAM), como por el procesador sobre el que se ejecutan las instrucciones (CPU o GPU). Los algoritmos basados en GPU considerados en este trabajo están optimizados para la arquitectura CUDA™ de NVidia®. Para evaluar la eficiencia se ha medido la tasa de cuadros por segundo (*frames per second*), obteniendo una tasa próxima a 160 fps con **BM_GPU** y unos 130 fps con **BM_CPU**. El algoritmo **SGBM**, con 30 fps también es apto para trabajar en tiempo real.

De los análisis, se puede concluir que **el algoritmo semiglobal SGBM cumple con los requisitos** de nuestra aplicación: genera un mapa denso y suave en el que se puede distinguir los planos de profundidad y logra un compromiso entre calidad objetiva y eficiencia. Es un algoritmo que minimiza la tasa de píxeles inválidos, y a pesar de no minimizar el error global, su valor es admisible. Además, alcanza una eficiencia bastante aceptable a pesar de tratarse de una implementación para CPU, estando abierto a una posible optimización para GPU que puede multiplicar la tasa de cuadros por segundo.

8.2 Trabajo futuro

A partir de las ideas generales y conclusiones del proyecto se puede extraer un conjunto de ideas para el futuro, ya sea para continuar, mejorar o modificar el enfoque del trabajo realizado. A continuación resumimos posibles tareas o líneas de trabajo futuro.

La correspondencia estéreo o *stereo matching* es una disciplina en constante investigación y desarrollo. Continuamente, aparecen actualizaciones de los algoritmos existentes para mejorarlos, o bien nuevos algoritmos que obtienen mejor rendimiento y calidad que los anteriores, actualizando el estado del arte. De hecho, durante el periodo de algunos meses que ha llevado la elaboración de esta memoria, hemos comprobado que se han evaluado nuevos algoritmos en la página de Middlebury [23]. Por otra parte, los algoritmos utilizados en nuestra herramienta son bastante básicos y ha pasado tiempo desde que fueron creados (lo que no hace que sean menos válidos), pero han servido para ilustrar aspectos generales de la correspondencia estéreo. Todo esto conduce a que la tarea futura más inmediata sea la de **incluir en la herramienta algoritmos más modernos**, además de **actualizar la clasificación de tipos de algoritmos** y **ampliar los parámetros de entrada** disponibles actualmente, adaptándolos a las nuevas modificaciones.

La librería OpenCV es bastante completa y sus algoritmos de correspondencia han sido muy útiles en este trabajo. En relación con el punto anterior, se plantea la búsqueda y utilización de **otras librerías existentes de software libre**.

Como se ha comprobado, el algoritmo SGBM alcanza un rendimiento suficiente para trabajar en tiempo real, tratándose de una implementación para CPU. La **optimización** de este algoritmo **para GPU** incrementaría considerablemente el rendimiento. Es más, la **búsqueda de algoritmos basados en GPU** en vez de en CPU daría lugar a una herramienta mucho más potente. Hasta ahora, los algoritmos basados en GPU que hemos considerado están optimizados para la tecnología NVidia® CUDA™, limitando las características del equipo utilizado para las prueba, ya que esta tecnología requiere hardware específico compatible con dicha arquitectura de NVidia®. Por lo tanto, se abre una línea de investigación de algoritmos optimizados para GPU, en general.

Como se ha comentado, finalmente no se ha incluido un módulo de rectificación, por lo que sería interesante estudiar la posibilidad de añadir una etapa previa a la correspondencia estéreo en la que se compruebe si el contenido de entrada está alineado, y si no es así, aplicar **rectificación no calibrada**. Sin embargo sabemos que es un proceso muy costoso y es complicado alcanzar una eficiencia adecuada para trabajar en tiempo real.

En el estado del arte de la correspondencia estéreo existen técnicas que hacen uso de otras herramientas de visión artificial, como el reconocimiento de la **geometría de la escena** o la **segmentación**, que sirven de apoyo para obtener resultados de mejor calidad. De nuevo, añaden coste computacional, siendo objeto de estudio la utilización de estas herramientas en tiempo real. La segmentación es un campo tan amplio o más que la correspondencia estéreo, existiendo un gran número de técnicas diferentes, por lo que la investigación y conocimiento de esta rama supone un trabajo adicional.

En caso de utilizar algoritmos de correspondencia con elevada eficiencia, se puede **incluir etapas de preprocesado y posprocesado** que permitan mejorar la calidad de los resultados obtenidos en la etapa de correspondencia. Estos procesos de filtrado o transformación de imagen también se pueden optimizar para GPU, con el objetivo de minimizar el retardo que introducen.

Un aspecto a destacar al trabajar con contenido en formato de vídeo es la **variabilidad de los parámetros de correspondencia**. Es decir, en un vídeo, y sobre todo al tratarse de un evento deportivo, hay mucho movimiento tanto de los objetos que pertenecen a la escena, como por cambios de cámara o zoom. Todas estas variaciones descritas requieren una modificación de los parámetros de correspondencia en tiempo real y se ha comprobado es que muy difícil modificarlos manualmente en base a un criterio subjetivo. Por lo tanto, es probable que sea necesario implementar la adaptación dinámica de parámetros o recurrir a algoritmos de correspondencia adaptativos o variacionales.

Este trabajo ha surgido ante la necesidad de buscar una solución al problema que se planteaba en el proyecto *ImmersiveTV*, donde se trabaja con el contenido de un evento deportivo en directo. Sin embargo, se podría extender para ser utilizado en **otro tipo de aplicaciones**. Para ello sería necesario **experimentar con otros tipos de contenido**, que presenten dificultades variadas. Generalizar la herramienta supone

un enorme reto, ya que hasta ahora, los trabajos realizados en este campo han estado enfocados a resolver problemas muy concretos.

Otra tarea posible es la **mejora de los criterios de evaluación** de algoritmos de correspondencia estéreo. Algunos entornos de evaluación existentes son capaces de calcular las estadísticas de error diferenciando entre las regiones problemáticas: zonas sin textura, oclusiones y discontinuidades. Sin embargo, nuestra herramienta sólo calcula las tasas de error sobre la imagen completa y para reconocer los errores en las regiones mencionadas, es necesario visualizar tanto el mapa de disparidad como la imagen reconstruida y de error normalizado. También se puede **introducir nuevas medidas de error**.

Por último, hemos de comentar que se ha presentado una publicación con los resultados de este trabajo, destinada a la conferencia VISAPP 2014, que ha sido rechazada. Algunos comentarios de los revisores iban enfocados a la actualización de los algoritmos incluidos en la herramienta. Siguiendo estas recomendaciones, se plantea la **elaboración de un artículo** para una revista o conferencia a corto plazo.

Anexo I

Código de las modificaciones sobre los algoritmos

En este anexo se recopila el código de las modificaciones realizadas sobre los algoritmos de correspondencia que se describen en la sección 5.3. Se han clasificado en dos grupos: algoritmos que pertenecen a la librería basada en GPU de OpenCV y algoritmos que se encuentran fuera del entorno de OpenCV. El criterio de clasificación es la facilidad de integración dentro de un mismo entorno, que viene dada por el conjunto de elementos que tienen en común, como por ejemplo, pertenecer al mismo espacio de nombres o compartir un archivo de cabeceras.

I.1 Algoritmos de la librería basada en GPU de OpenCV

I.1.1 BM_GPU

Núcleo de StereoBM

```
template<int RADIUS>
__global__ void stereoKernel(unsigned char *left, unsigned char *right, size_t img_step, PtrStep<signed char> disp, int dmin, int ndisp){
  extern __shared__ unsigned int col_ssd_cache[];
  volatile unsigned int *col_ssd = col_ssd_cache + BLOCK_W + threadIdx.x;
  volatile unsigned int *col_ssd_extra = threadIdx.x < (2 * RADIUS) ? col_ssd + BLOCK_W : 0;
  // Antes: int X = (blockIdx.x * BLOCK_W + threadIdx.x + ndisp + RADIUS);
  int X = (blockIdx.x * BLOCK_W + threadIdx.x + (dmin + ndisp - 1) + RADIUS);
  #define Y (blockIdx.y * ROWSperTHREAD + RADIUS)
  unsigned int* minSSDImage = cminSSDImage + X + Y * cminSSD_step;
  signed char* disparImage = disp.data + X + Y * disp.step;
  int end_row = ::min(ROWSperTHREAD, cheight - Y - RADIUS);
  int y_tex;
  int x_tex = X - RADIUS;
```

Núcleo de StereoBM (cont.)

```
  if (x_tex >= cwidth)
    return;
  // Antes: for(int d = STEREO_MIND; d < ndisp; d += STEREO_DISP_STEP)
  for(int d = /*dmin*/literal(dmin); d < dmin + ndisp - 1; d += STEREO_DISP_STEP){
    y_tex = Y - RADIUS;
    cv_bis::gpu::device::stereobm::InitColSSD<RADIUS>(x_tex, y_tex, img_step, left, right, d, col_ssd);
    if (col_ssd_extra > 0)
      if (x_tex + BLOCK_W < cwidth)
        cv_bis::gpu::device::stereobm::InitColSSD<RADIUS>(x_tex + BLOCK_W, y_tex, img_step, left, right, d, col_ssd_extra);
    __syncthreads(); //before MinSSD function
    // Antes: if (X < cwidth - RADIUS && Y < cheight - RADIUS)
    if (X < cwidth - RADIUS && Y < cheight - RADIUS && X >= 0){
      uint2 minSSD = cv_bis::gpu::device::stereobm::MinSSD<RADIUS>(col_ssd_cache + threadIdx.x, col_ssd);
      if (minSSD.x < minSSDImage[0]){
        // Antes: disparImage[0] = (unsigned char)(d + minSSD.y);
        disparImage[0] = (signed char)d + (signed char)minSSD.y;
        minSSDImage[0] = minSSD.x;
      }
    }
    for(int row = 1; row < end_row; row++){
      int idx1 = y_tex * img_step + x_tex;
      int idx2 = (y_tex + (2 * RADIUS + 1)) * img_step + x_tex;
      __syncthreads();
      cv_bis::gpu::device::stereobm::StepDown<RADIUS>(idx1, idx2, left, right, d, col_ssd);
      if (col_ssd_extra)
        if (x_tex + BLOCK_W < cwidth)
          cv_bis::gpu::device::stereobm::StepDown<RADIUS>(idx1, idx2, left + BLOCK_W, right + BLOCK_W, d, col_ssd_extra);
      y_tex += 1;
      __syncthreads(); //before MinSSD function

      // Antes: if (X < cwidth - RADIUS && row < cheight - RADIUS - Y)
      if (X < cwidth - RADIUS && row < cheight - RADIUS - Y && X >= 0){
        int idx = row * cminSSD_step;
        uint2 minSSD = cv_bis::gpu::device::stereobm::MinSSD<RADIUS>(col_ssd_cache + threadIdx.x, col_ssd);
        if (minSSD.x < minSSDImage[idx]){
          // Antes: disparImage[disp.step * row] = (unsigned char)(d + minSSD.y);
          disparImage[disp.step * row] = (signed char)d + (signed char)minSSD.y;
          minSSDImage[idx] = minSSD.x;
        }
      }
    } // for row loop
  } // for d loop
}
```

I.1.2 BP_GPU

Núcleo para calcular el *data cost* de BP_GPU para cada valor de disparidad

```
template <int cn, typename D> __global__ void comp_data(const DevMem2Db left, const PtrStepb right, PtrElemStep_<D> data){
  const int x = blockIdx.x * blockDim.x + threadIdx.x;
  const int y = blockIdx.y * blockDim.y + threadIdx.y;
  if (y > 0 && y < left.rows - 1 && x > 0 && x < left.cols - 1) {
    const cv::gpu::device::uchar* ls = left.ptr(y) + x * cn;
    const cv_bis::gpu::device::stereobp::PixDiff<cn> pixDiff(ls);
    const cv::gpu::device::uchar* rs = right.ptr(y) + x * cn;
    D* ds = data.ptr(y) + x;
    const size_t disp_step = data.step * left.rows;
    // Antes: for (int disp = 0; disp < cndisp; disp++)
    for (int disp = cdmin; disp < cdmin+cndisp; disp++) {
      // Antes: if (x - disp >= 1)
      if(x - disp >=1 && x - disp <= left.cols - 1) {
        float val = pixDiff(rs - disp * cn);
        // Antes: ds[disp * disp_step] = cv::gpu::device::saturate_cast<D>(fmin(cdata_weight * val, cdata_weight * cmax_data_term));
        ds[(disp-cdmin) * disp_step] = cv::gpu::device::saturate_cast<D>(fmin(cdata_weight * val, cdata_weight * cmax_data_term));
      } else {
        // Antes: ds[disp * disp_step] = cv::gpu::device::saturate_cast<D>(cdata_weight * cmax_data_term);
        ds[(disp-cdmin) * disp_step] = cv::gpu::device::saturate_cast<D>(cdata_weight * cmax_data_term);
      }
    }
  }
}
```

Núcleo para calcular la disparidad minimizando el coste

```
template <typename T>
__global__ void output(const PtrElemStep_<T> u, const T* d, const T* l, const T* r, const T* data,DevMem2D_<short> disp){
  const int x = blockIdx.x * blockDim.x + threadIdx.x;
  const int y = blockIdx.y * blockDim.y + threadIdx.y;
  if (y > 0 && y < disp.rows - 1 && x > 0 && x < disp.cols - 1){
    const T* us = u.ptr(y + 1) + x;
    const T* ds = d + (y - 1) * u.step + x;
    const T* ls = l + y * u.step + (x + 1);
    const T* rs = r + y * u.step + (x - 1);
    const T* dt = data + y * u.step + x;
    size_t disp_step = disp.rows * u.step;
    int best = 0;
    float best_val = cv::gpu::device::numeric_limits<float>::max();
    // Antes:for (int d = 0; d < cndisp; ++d)
    for (int d = cdmin; d < cdmin+cndisp; ++d){
      /* Antes: float val  = us[d * disp_step]; val += ds[d * disp_step]; val += ls[d * disp_step];
                                                val += rs[d * disp_step]; val += dt[d * disp_step]; */
      float val  = us[(d-cdmin) * disp_step];
            val += ds[(d-cdmin) * disp_step];
            val += ls[(d-cdmin) * disp_step];
            val += rs[(d-cdmin) * disp_step];
            val += dt[(d-cdmin) * disp_step];
      if (val < best_val){
        best_val = val;
        // Antes: best = d;
        best = d-cdmin; // Deshace el offset aplicado a los índices del belief vector
      }
    }
    disp.ptr(y)[x] = cv::gpu::device::saturate_cast<short>(best);
  }
}
```

I.1.3 CSBP

Núcleo para calcular el *data cost* de CSBP para cada valor de disparidad

```
template <typename T, int channels> __global__ void init_data_cost(int h, int w, int level){
    int x = blockIdx.x * blockDim.x + threadIdx.x;
    int y = blockIdx.y * blockDim.y + threadIdx.y;
    if (y < h && x < w){
        int y0 = y << level;
        int yt = (y + 1) << level;
        int x0 = x << level;
        int xt = (x + 1) << level;
        T* data_cost = (T*)ctemp + y * cmsg_step + x;
        // Antes: for(int d = 0; d < cndisp; ++d)
        for(int d = cdmin; d < cdmin+cndisp; ++d){
            float val = 0.0f;
            for(int yi = y0; yi < yt; yi++){
                for(int xi = x0; xi < xt; xi++){
                    int xr = xi - d;
                    // Antes: if(d < cth || xr < 0)
                    if(d < cth || xr < 0 || xr > (w-1))
                        val += cdata_weight * cmax_data_term;
                    else {
                        const cv::gpu::device::uchar* lle = cleft + yi * cimg_step + xi * channels;
                        const cv::gpu::device::uchar* lri = cright + yi * cimg_step + xr * channels;
                        val += cv_bis::gpu::device::stereocsbp::DataCostPerPixel<channels>::compute(lle, lri);
                    }
                }
            }
            // Antes: data_cost[cdisp_step1 * d] = cv::gpu::device::saturate_cast<T>(val);
            data_cost[cdisp_step1 * (d-cdmin)] = cv::gpu::device::saturate_cast<T>(val);
        }
    }
}
```

I.2 Algoritmos fuera del entorno de OpenCV

I.2.1 Archivo común `image.h`

Inicialización de píxeles de una imagen

```
template <class T> image<T>::image(const int width, const int height, int val){
    w = width;
    h = height;
    data = new T[w * h];  // allocate space for image data
    access = new T*[h];   // allocate space for row pointers
    // initialize row pointers
    for (int i = 0; i < h; i++)
        access[i] = data + (i * w);
    // init to val
    memset(data, val, w * h * sizeof(T));
}
```

Constructor de imagen a partir de una matriz de OpenCV

```
template <class T> image<T>::image(Mat m){
    w = m.cols;
    h = m.rows;
    data = new T[w * h];  // allocate space for image data
    access = new T*[h];   // allocate space for row pointers
    // initialize row pointers
    for (int i = 0; i < h; i++)
        access[i] = data + (i * w);
    // image<T> *im = new image<T>(w, h, false);
    memcpy(data, m.data, w * h * sizeof(T));
}
```

Conversión de imagen a matriz de OpenCV

```
static Mat img2mat(image<uchar> *im){
    int height = im->height();
    int width = im->width();
    Mat m(height,width,CV_8U,im->data);
   return m;
}
```

I.2.2 BPVISION

bpvision.h

```
class Bpvision{
   struct bpvParameters{
      int iter;     // number of BP iterations at each scale
      int levels;   // number of scales
      float disc_k; // truncation of discontinuity cost
      float data_k; // truncation of data cost
      float lambda; // weighting of data cost
      int dmin;     // minimum disparity value
      int dmax;     // maximum disparity value
      int ndisp;    // number of possible disparities
      float sigma;  // amount to smooth the input images
   };
   public:
      // Default constructor
      Bpvision ();
      // constructor, input: bpvParameters
      Bpvision (bpvParameters p);
      void init (int disp_min, int numberOfDisparities, int iterations, int lev);
      void init (bpvParameters p);
      void setDmin(int disp_min);
      void setnDisp(int numberOfDisparities);
      void setIters(int iter);
      void setLevels(int lev);
      int getDmin();
      int getDmax();
      int getnDisp();
      int getIters();
      int getLevels();
      void printBpvParams() const;
      string strBpvParams() const;
      // multiscale belief propagation for image restoration
      image<uchar> *stereo_ms(image<uchar> *img1, image<uchar> *img2);
      // deconstructor
      ~Bpvision (){}
   private:
      // parameter set
      bpvParameters bpvParam;
      // compute message
      void msg(float s1[VALUES], float s2[VALUES], float s3[VALUES],float s4[VALUES], float dst[VALUES]);
      // computation of data costs
      image<float[VALUES]> *comp_data(image<uchar> *img1, image<uchar> *img2);
      // generate output from current messages
      image<uchar> *output(image<float[VALUES]> *u, image<float[VALUES]> *d,image<float[VALUES]> *l,
                           image<float[VALUES]> *r, image<float[VALUES]> *data);
      // belief propagation using checkerboard update scheme
      void bp_cb(image<float[VALUES]> *u, image<float[VALUES]> *d,  image<float[VALUES]> *l,
                 image<float[VALUES]> *r, image<float[VALUES]> *data);
};
```

Asignación del rango de disparidad

```
void Bpvision::setDmin(int disp_min){
   bpvParam.dmin = disp_min;
   bpvParam.dmax = bpvParam.dmin + bpvParam.ndisp - 1;
}
```

Cálculo de mensajes

```
// compute message
void Bpvision::msg(float s1[VALUES], float s2[VALUES], float s3[VALUES], float s4[VALUES],float dst[VALUES]) {
    float val;
    int ndisp = bpvParam.ndisp;
    int dmin = bpvParam.dmin;
    int dmax = bpvParam.dmax;
    // aggregate and find min
    float minimum = INF;
    /* Antes: for (int value = 0; value < VALUES; value++) {
                dst[value] = s1[value] + s2[value] + s3[value] + s4[value];
                if (dst[value] < minimum)
                    minimum = dst[value];
            } */
    for (int disp = dmin; disp <= dmax; disp++) {
        int disparity = disp-dmin;
        dst[disparity] = s1[disparity] + s2[disparity] + s3[disparity] + s4[disparity];
        if (dst[disparity] < minimum)
            minimum = dst[disparity];
    }
    // dt
    // Antes: dt(dst);
    dt(dst, ndisp);
    // truncate
    minimum += bpvParam.disc_k;
    /* Antes: for (int value = 0; value < VALUES; value++)
                    if (minimum < dst[value])
                        dst[value] = minimum;*/
   for (int disp = dmin; disp <= dmax; disp++){
       int disparity = disp-dmin;
       if (minimum < dst[disparity])
           dst[disparity] = minimum;
   }
   // normalize
   val = 0;
   /* Antes: for (int value = 0; value < VALUES; value++)
                  val += dst[value]; */
   for (int disp = dmin; disp <=dmax; disp++)
       val += dst[disp-dmin];
    /* Antes: val /= VALUES;
               for (int value = 0; value < VALUES; value++)
                  dst[value] -= val; */
   val /= ndisp;
   for (int disp = /*0*/dmin; disp /*< ndisp*/ <=dmax; disp++)
      dst[disp-dmin] -= val;
}
```

Cálculo del *data cost* para cada valor de disparidad

```
// computation of data costs
image<float[VALUES]>* Bpvision::comp_data(image<uchar> *img1, image<uchar> *img2) {
    int width = img1->width();
    int height = img1->height();
    int ndisp = bpvParam.ndisp;
    int dmin = bpvParam.dmin;
    int dmax = bpvParam.dmax;
    // Antes: image<float[VALUES]> *data = new image<float[VALUES]>(width, height);
    image<float[VALUES]> *data = new image<float[VALUES]>(width, height,true);
    image<float> *sm1, *sm2;
    if (bpvParam.sigma >= 0.1) {
        sm1 = smooth(img1, bpvParam.sigma);
        sm2 = smooth(img2, bpvParam.sigma);
    } else {
        sm1 = imageUCHARtoFLOAT(img1);
        sm2 = imageUCHARtoFLOAT(img2);
    }
    for (int y = 0; y < height; y++) {
        int xmin= dmax>0?dmax:0;          // Si xmin=dmax>0, al hacer x-disp, dara 0 o >0 cuando dmin <= disp <= dmax
        int xmax= dmin<0?width+dmin:width;// Si xmax=width+dmin, al hacer x-disp, dará <width cuando dmin<=disp<=dmax
        /* Antes: for (int x = VALUES-1; x < width; x++) {
                    for (int value = 0; value < VALUES; value++) {
                        float val = abs(imRef(sm1, x, y)-imRef(sm2, x-value, y));
                        imRef(data, x, y)[value] = LAMBDA * std::min(val, DATA_K);
                    }
                }*/
        for (int x = xmin; x < xmax; x++){
            for (int disp = dmin; disp <= dmax; disp++) {
                float val = abs(imRef(sm1, x, y)-imRef(sm2, x-disp, y));
                imRef(data, x, y)[disp-dmin] = bpvParam.lambda * std::min(val, bpvParam.data_k);
            }
        }
    }
    delete sm1;
    delete sm2;
    return data;
}
```

Cálculo de la disparidad minimizando el *belief vector*

```
// generate output from current messages
image<uchar>* Bpvision::output(image<float[VALUES]> *u, image<float[VALUES]> *d, image<float[VALUES]> *l,
                               image<float[VALUES]> *r, image<float[VALUES]> *data) {
    int ndisp = bpvParam.ndisp;
    int dmin = bpvParam.dmin;
    int dmax = bpvParam.dmax;
    int width = data->width();
    int height = data->height();
    // Antes: image<uchar> *out = new image<uchar>(width, height);
    image<uchar> *out = new image<uchar>(width, height, dmin);

    for (int y = 1; y < height-1; y++) {
        for (int x = 1; x < width-1; x++) {
            // keep track of best value for current pixel
            int best = 0;
            float best_val = INF;
            /* Antes: for (int value = 0; value < VALUES; value++) {
                        float val = imRef(u, x, y+1)[value] + imRef(d, x, y-1)[value] + imRef(l, x+1, y)[value] +
                                    imRef(r, x-1, y)[value] + imRef(data, x, y)[value];*/
            for (int disp = dmin; disp <= dmax; disp++) {
                int disparity = disp-dmin;
                float val = imRef(u, x, y+1)[disparity] +
                            imRef(d, x, y-1)[disparity] +
                            imRef(l, x+1, y)[disparity] +
                            imRef(r, x-1, y)[disparity] +
                            imRef(data, x, y)[disparity]; // E = bq = Mensajes(up + down + left + right) + data_cost
                if (val < best_val) {
                    best_val = val;
                    best = disparity;
                }
            }
            imRef(out, x, y) = best * SCALE;             // Resultado escalado y desplazado un offset dmin
        }
    }
    return out;
}
```

I.2.3 ELAS

Fragmento del código de inicialización

```
// INICIALIZACIÓN DE LA IMAGEN DE DISPARIDAD
// Antes: init disparity image to -10
    if (param.subsampling) {
        for (int32_t i=0; i<(width/2)*(height/2); i++)
            // Antes: *(D+i) = -10;
            *(D+i) = param.disp_min - 1;
   } else {
       for (int32_t i=0; i<width*height; i++)
           // Antes: *(D+i) = -10;
           *(D+i) = param.disp_min - 1;
   }
}
```

Clase Elas

```
class Elas {
   enum setting {ROBOTICS,MIDDLEBURY};
   // parameter settings
   struct elasParameters {
      int32_t disp_min;               // min disparity
      int32_t disp_max;               // max disparity
      float   support_threshold;      // max. uniqueness ratio (best vs. second best support match)
      int32_t support_texture;        // min texture for support points
      int32_t candidate_stepsize;     // step size of regular grid on which support points are matched
      int32_t incon_window_size;      // window size of inconsistent support point check
      int32_t incon_threshold;        // disparity similarity threshold for support point to be considered consistent
      int32_t incon_min_support;      // minimum number of consistent support points
      bool    add_corners;            // add support points at image corners with nearest neighbor disparities
      int32_t grid_size;              // size of neighborhood for additional support point extrapolation
      float   beta;                   // image likelihood parameter
      float   gamma;                  // prior constant
      float   sigma;                  // prior sigma
      float   sradius;                // prior sigma radius
      int32_t match_texture;          // min texture for dense matching
      int32_t lr_threshold;           // disparity threshold for left/right consistency check
      float   speckle_sim_threshold;  // similarity threshold for speckle segmentation
      int32_t speckle_size;           // maximal size of a speckle (small speckles get removed)
      int32_t ipol_gap_width;         // interpolate small gaps (left<->right, top<->bottom)
      bool    filter_median;          // optional median filter (approximated)
      bool    filter_adaptive_mean;   // optional adaptive mean filter (approximated)
      bool    postprocess_only_left;  // saves time by not postprocessing the right image
      bool    subsampling;            // saves time by only computing disparities for each 2nd pixel
      // note: for this option D1 and D2 must be passed with size
      //       width/2 x height/2 (rounded towards zero)
      // constructor
      elasParameters (setting s=ROBOTICS, int dmin=0, int numberOfDisparities=256) {};
public:
   // Default constructor
   Elas();
   // constructor, input: parameters
   Elas (elasParameters p);
   void init (int dmin, int ndisp, setting s=ROBOTICS);
   void init (elasParameters p);
   void setPostprocessOnlyLeft(bool postprocess_only_left);
   bool getPostprocessOnlyLeft();
   void setDmin(int disp_min);
   void setnDisp(int numberOfDisparities);
   int getDmin();
   int getDmax();
   int getnDisp();
   void printElasParams() const;
   std::string strElasParams() const;
   void operator()(uint8_t* I1,uint8_t* I2,float* D1,float* D2,const int32_t* dims);
   // deconstructor
   ~Elas () {}
private:
   // matching function
   // inputs: pointers to left (I1) and right (I2) intensity image (uint8, input)
   //         pointers to left (D1) and right (D2) disparity image (float, output)
   //         dims[0] = width of I1 and I2
   //         dims[1] = height of I1 and I2
   //         dims[2] = bytes per line (often equal to width, but allowed to differ)
   //         note: D1 and D2 must be allocated before (bytes per line = width)
   //               if subsampling is not active their size is width x height,
   //               otherwise width/2 x height/2 (rounded towards zero)
   void process (uint8_t* I1,uint8_t* I2,float* D1,float* D2,const int32_t* dims);
```

Clase Elas (cont.)

```
  struct support_pt {
    int32_t u;
    int32_t v;
    int32_t d;
    support_pt(int32_t u,int32_t v,int32_t d):u(u),v(v),d(d){}
  };
  struct triangle {
    int32_t c1,c2,c3;
    float   t1a,t1b,t1c;
    float   t2a,t2b,t2c;
    triangle(int32_t c1,int32_t c2,int32_t c3):c1(c1),c2(c2),c3(c3){}
  };
  inline uint32_t getAddressOffsetImage (const int32_t& u,const int32_t& v,const int32_t& width) {
    return v*width+u;
  }
  inline uint32_t getAddressOffsetGrid (const int32_t& x,const int32_t& y,const int32_t& d,
                                        const int32_t& width,const int32_t& disp_num) {
    // Antes: return (y*width+x)*disp_num+d;
    return (y*width+x)*disp_num+(d-param.disp_min);
  }
  // support point functions
  void removeInconsistentSupportPoints (int16_t* D_can,int32_t D_can_width,int32_t D_can_height);
  void removeRedundantSupportPoints (int16_t* D_can,int32_t D_can_width,int32_t D_can_height,
                                     int32_t redun_max_dist, int32_t redun_threshold, bool vertical);
  void addCornerSupportPoints (std::vector<support_pt> &p_support);
  inline int16_t computeMatchingDisparity (const int32_t &u,const int32_t &v,uint8_t* I1_desc,uint8_t* I2_desc,const bool &right_image);
  std::vector<support_pt> computeSupportMatches (uint8_t* I1_desc,uint8_t* I2_desc);
  // triangulation & grid
  std::vector<triangle> computeDelaunayTriangulation (std::vector<support_pt> p_support,int32_t right_image);
  void computeDisparityPlanes (std::vector<support_pt> p_support,std::vector<triangle> &tri,int32_t right_image);
  void createGrid (std::vector<support_pt> p_support,int32_t* disparity_grid,int32_t* grid_dims,bool right_image);
  // matching
  inline void updatePosteriorMinimum (__m128i* I2_block_addr,const int32_t &d,const int32_t &w,
                                      const __m128i &xmm1,__m128i &xmm2,int32_t &val,int32_t &min_val,int32_t &min_d);
  inline void updatePosteriorMinimum (__m128i* I2_block_addr,const int32_t &d,
                                     const __m128i &xmm1,__m128i &xmm2,int32_t &val,int32_t &min_val,int32_t &min_d);
  inline void findMatch (int32_t &u,int32_t &v,float &plane_a,float &plane_b,float &plane_c,
                         int32_t* disparity_grid,int32_t *grid_dims,uint8_t* I1_desc,uint8_t* I2_desc,
                         int32_t *P,int32_t &plane_radius,bool &valid,bool &right_image,float* D);
  void computeDisparity (std::vector<support_pt> p_support,std::vector<triangle> tri,int32_t* disparity_grid,int32_t* grid_dims,
                         uint8_t* I1_desc,uint8_t* I2_desc,bool right_image,float* D);
  // L/R consistency check
  void leftRightConsistencyCheck (float* D1,float* D2);
  // postprocessing
  void removeSmallSegments (float* D);
  void gapInterpolation (float* D);
  // optional postprocessing
  void adaptiveMean (float* D);
  void median (float* D);
  // parameter set
  elasParameters param;
  // memory aligned input images + dimensions
  uint8_t *I1,*I2;
  int32_t width,height,bpl;
  // profiling timer
  Timer timer;
};
```

Filtrado de correspondencias inconsistentes

```
void Elas::removeInconsistentSupportPoints (int16_t* D_can,int32_t D_can_width,int32_t D_can_height) {
  // for all valid support points do
  for (int32_t u_can=0; u_can<D_can_width; u_can++) {
    for (int32_t v_can=0; v_can<D_can_height; v_can++) {
      int16_t d_can = *(D_can+getAddressOffsetImage(u_can,v_can,D_can_width));
      // Antes: if (d_can>=0)
      if (d_can>param.disp_min-1) {
        // compute number of other points supporting the current point
        int32_t support = 0;
        for (int32_t u_can_2=u_can-param.incon_window_size; u_can_2<=u_can+param.incon_window_size; u_can_2++) {
          for (int32_t v_can_2=v_can-param.incon_window_size; v_can_2<=v_can+param.incon_window_size; v_can_2++) {
            if (u_can_2>=0 && v_can_2>=0 && u_can_2<D_can_width && v_can_2<D_can_height) {
              int16_t d_can_2 = *(D_can+getAddressOffsetImage(u_can_2,v_can_2,D_can_width));
              // Antes: if (d_can_2>=0 && abs(d_can-d_can_2)<=param.incon_threshold)
              if (d_can_2>param.disp_min-1 && abs(d_can-d_can_2)<=param.incon_threshold)
                support++;
            }
          }
        }
        // invalidate support point if number of supporting points is too low
        if (support<param.incon_min_support)
          // Antes: *(D_can+getAddressOffsetImage(u_can,v_can,D_can_width)) = -1;
          *(D_can+getAddressOffsetImage(u_can,v_can,D_can_width)) = param.disp_min-1;
      }
    }
  }
}
```

Anexo II

Instalación del DVD

En este anexo se describe el software incluido en el DVD y su procedimiento de instalación. El la información descrita en este anexo también está disponible en el archivo `README.txt` en la raíz del DVD.

REQUISITOS MÍNIMOS

- **Sistema Operativo:** *Microsoft™ Windows* 64 bits (o superior).
- **Memoria RAM:** 4 GB.
- **Procesador:** *Intel® Core™ 2*.
- **Disco duro:** aproximadamente 750 MB.
- **Tarjeta gráfica compatible con arquitectura CUDA**: consultar en `https://developer.nvidia.com/cuda-gpus`

CONTENIDO DEL DVD

- `img`**:** directorio de imágenes en formato *side-by-side* y dos vistas separadas.
- `StereoMatching`**:** directorio que contiene el software desarrollado en este PFC junto con las librerías necesarias.
- `vídeos`**:** directorio que contiene un fragmento de vídeo en formato *side-by-side* con contenido del proyecto ImmersiveTV.
- `cudatoolkit_4.2.9_win_64.msi`**:** instalador del SDK de NVidia® CUDA™ 4.2.
- `install.bat`**:** archivo ejecutable de instalación del software.
- `README.txt`**:** documento de texto con las instrucciones de instalación.

PROCEDIMIENTO DE INSTALACIÓN

1. Comprobar que está instalado el software de NVidia® CUDA™ versión 4.2 necesario para la ejecución de nuestra herramienta[1]. Si no se encuentra instalado, ejecutar el archivo de instalación `cudatoolkit_4.2.9_win_64.msi` (también disponible en la sección '*Download Toolkit*' en el sitio web de NVIDIA[2]). Comprobar que se han generado las **variables de entorno** en:

   · Panel de control ≫ Sistema y seguridad ≫ Sistema ≫ Configuración avanzada del sistema ≫ Variables de entorno, o

   · Click derecho en Equipo ≫ propiedades ≫ Configuración avanzada del sistema ≫ Variables de entorno

   Variables de sistema:

   `CUDA_BIN_PATH: %CUDA_PATH%bin`

   `CUDA_INC_PATH: %CUDA_PATH%include`

   `CUDA_LIB_PATH: %CUDA_PATH%lib\x64`

   `CUDA_PATH: %CUDA_PATH_V4_2%`

   `CUDA_PATH_V4_2: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v4.2\`

   `PATH: %CUDA_PATH%bin; %CUDA_PATH%libnvvp;`

2. Ejecutar `install.bat` como administrador, haciendo click con el botón derecho y seleccionar la opción "Ejecutar como Administrador". Se creará un directorio llamado `StereoMatching` en la ruta `C:\Program Files`, en cuyo interior se copiará el contenido del directorio `StereoMatching` del presente DVD.

3. Al finalizar la copia, aparecerá un mensaje que solicitará confirmación para sobreescribir la variable de sistema `Path` (figura II.1). Al indicar "Sí', escribiendo 's' o 'si', se guardará la ruta de instalación en `Path`.

FIGURA II.1 – **Mensaje de confirmación de sobreescritura.**

[1] Ruta habitual de instalación: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v4.2\
[2] https://developer.nvidia.com/cuda-toolkit-42-archive

4. Si el `Path` se ha creado correctamente, aparecerá el mensaje de la figura II.2, indicando que la instalación se ha realizado correctamente. Se debe comprobar que se ha añadido la ruta `C:\Program Files\StereoMatching\bin;` a la variable de sistema `Path` (en el punto 1 se indica cómo encontrar las variables de sistema). Si no se ha añadido, editar la variable `Path` añadiendo `C:\Program Files\StereoMatching\bin;` (no olvidar ';').

FIGURA II.2 – **Mensaje de instalación correcta.**

5. Si todo se ha desarrollado correctamente, al hacer doble click en el archivo ejecutable `StereoGUI.exe` instalado en la ubicación `C:\Program Files\StereoMatching`, debe aparecer la interfaz gráfica de usuario descrita en la memoria del proyecto.

POSIBLES PROBLEMAS

- Si los archivos no se han copiado correctamente, realizar la instalación manual:
  - Copiar el directorio `StereoMatching` con todo su contenido, en cualquier ubicación de su equipo.
  - Añadir la ruta del directorio `(ruta de instalación)\StereoMatching\bin;` a la variable de sistema `Path` (como se explicaba en el apartado anterior).
- Si aparece un mensaje de error porque no encuentra una librería, comprobar que se ha añadido la ruta de instalación en la variable de sistema `Path` (como en el punto anterior).

Bibliografía

[1] M. Slater and S. Wilbur, "A framework for immersive virtual environments (five): Speculations on the role of presence in virtual environments," *Presence: Teleoperators and virtual environments*, vol. 6, no. 6, pp. 603–616, 1997.

[2] "ImmersiveTV. una aproximación a los medios inmersivos." `http://www.immersivetv.es/`, 2010–2012.

[3] F. P. Luque-Oostrom, L. Piovano, I. Galloso, D. Garrido, E. Sánchez, and C. Feijóo, "Experimental prototype merging stereo panoramic video and interactive 3D content in a 5-sided CAVE^{TM}," in *Proceedings of Joint Virtual Reality Conference of ICAT - EGVE - EuroVR (JVRC 2012)*, (Madrid, Spain), pp. 93–96, October 2012.

[4] I. Galloso, F. Luque-Oostrom, L. Piovano, D. Garrido, E. Sánchez, and C. Feijóo, "Foundations of a new interaction paradigm for immersive 3D multimedia," in *2012 NEM Summit: Implementing Future Media Internet towards New Horizons. 2012 NEM Summit Conference Proceedings*, (Istanbul, Turkey), pp. 127–132,140, NEM-Initiative, October 2012.

[5] F. P. Luque, I. Galloso, C. Feijóo, C. A. Martín, and G. Cisneros, "Integration of multi-sensorial stimuli and multi-modal interaction in a hybrid 3DTV system," *ACM Transactions on Multimedia Computing Communications and applications (ACM-TOMCCAP)*, 2014. p. to appear.

[6] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart, "The cave: audio visual experience automatic virtual environment," *Commun. ACM*, vol. 35, pp. 64–72, June 1992.

[7] C. Cruz-Neira, D. J. Sandin, and T. A. DeFanti, "Surround-screen projection-based virtual reality: the design and implementation of the cave," in *Proceedings*

of the 20th annual conference on Computer graphics and interactive techniques, pp. 135–142, ACM, 1993.

[8] "CAVE, Cave Automatic Virtual Environment." `http://cmuq-mis.wikispaces.com/CAVE,+Cave+Automatic+Virtual+Environment`, 2013.

[9] S. Epelbaum, "Historia de la Estereoscopía y sus Aplicaciones," *Archivos de Oftalmología de Buenos Aires*, vol. 81, no. 2, pp. 62–67, 2010.

[10] M. García, "Los secretos de la estereoscopía." Máster en Informática Gráfica, Juegos y Realidad Virtual. Escuela Técnica Superior en Ingeniería Informática (ETSII). Universidad Rey Juan Carlos, 2011.

[11] I. Howard and B. Rogers, *Seeing in Depth*. No. v. 1 in Seeing in Depth, I. Porteous, 2002.

[12] I. Montañana, P. Carmona, and G. Fajarnés, *Expressió gràfica i infografia*. Monografies de la Universitat Politècnica de València sobre ciència, tecnologia i art, Universitat Politècnica de València, 2006.

[13] Z. Arican, S. Yea, A. Sullivan, and A. Vetro, "Intermediate view generation for perceived depth adjustment of stereo video," in *SPIE Optical Engineering + Applications*, pp. 74430U–74430U, International Society for Optics and Photonics, 2009.

[14] N. S. Holliman, "Mapping perceived depth to regions of interest in stereoscopic images," SPIE, 2004.

[15] J. Konrad, "Enhancement of viewer comfort in stereoscopic viewing: parallax adjustment," in *Electronic Imaging'99*, pp. 179–190, International Society for Optics and Photonics, 1999.

[16] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, pp. 7–42, Apr. 2002.

[17] H. Hirschmüller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Computer Vision and Pattern Recognition (CVPR'07), 2007 IEEE Conference on*, pp. 1–8, IEEE, 2007.

[18] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 900–906, IEEE, 2003.

[19] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 9, pp. 1124–1137, 2004.

[20] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields," in *Computer Vision–ECCV 2006*, pp. 16–29, Springer, 2006.

[21] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8, IEEE, 2008.

[22] R. Kalarot, J. Morris, D. Berry, and J. Dunning, "Analysis of real-time stereo vision algorithms on gpu," in *International Conference on Image and Vision Computing New Zealand (IVCNZ)*, 2011.

[23] "Middlebury Stereo Evaluation web page." `http://vision.middlebury.edu/stereo/`, 2002–2013.

[24] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "The KITTI vision benchmark suite." `http://www.cvlibs.net/datasets/kitti/`, 2012–2013.

[25] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Computer Vision and Pattern Recognition (CVPR)*, (Providence, USA), June 2012.

[26] ".enpeda.. Image Sequence Analysis Test Site (EISATS)." `http://www.mi.auckland.ac.nz/index.php?option=com_content&view=article&id=44&Itemid=67`.

[27] M. Urvoy, M. Barkowsky, R. Cousseau, Y. Koudota, V. Ricorde, P. Le Callet, J. Gutiérrez, and N. García, "Nama3ds1-cospad1: Subjective video quality assessment database on coding conditions introducing freely available high quality

3d stereoscopic sequences," in *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pp. 109–114, IEEE, 2012.

[28] J. Banks and P. Corke, "Quantitative evaluation of matching methods and validity measures for stereo vision," *The International Journal of Robotics Research*, vol. 20, no. 7, pp. 512–532, 2001.

[29] R. Szeliski, "Prediction error as a quality metric for motion and stereo," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2, pp. 781–788, IEEE, 1999.

[30] S. Morales and R. Klette, *Prediction error evaluation of various stereo matching algorithms on long stereo sequences.* Multimedia Imaging Technology, 2009.

[31] "Digital Video Broadcasting (DVB): Frame Compatible Plano-Stereoscopic 3DTV (DVB-3DTV)." DVB BlueBook A154, February 2011.

[32] "Digital Video Broadcasting (DVB): Frame Compatible Plano-Stereoscopic 3DTV." ETSI TS 101 547 v1.1.1, January 2012.

[33] *The StereoGraphics Developer's Handbook – Background on Creating Images for CrystalEyes® and SimulEyes®*. StereoGraphics Corporation, 1997.

[34] R. A. Española, *Diccionario de la lengua española.* Real Academia Española, 22 ed., 2001.

[35] "On binocular vision; and on the stereoscope, an instrument for illustrating its phenomena," *British Association Report (pt.2)*, pp. 16–17, 1838.

[36] C. Wheatstone, "Contributions to the physiology of vision.–part the first. on some remarkable, and hitherto unobserved, phenomena of binocular vision," *Philosophical transactions of the Royal Society of London*, vol. 128, pp. 371–394, 1838.

[37] M. Gordon, *The Home Life of Sir David Brewster.* Cambridge Library Collection - Physical Sciences, Cambridge University Press, 2010.

[38] "Delivering 3DTV to Viewers. The unique 3DTV standard for the world." DVB Fact Sheet, August 2011.

[39] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, pp. I–195, IEEE, 2003.

[40] G. Iddan and G. Yahav, "3d imaging in the studio (and elsewhere)," in *Proc. SPIE*, vol. 4298, pp. 48–55, 2001.

[41] F. Dellaert, S. M. Seitz, C. E. Thorpe, and S. Thrun, "Structure from motion without correspondence," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2, pp. 557–564, IEEE, 2000.

[42] A. J. Woods, T. Docherty, and R. Koch, "Image distortions in stereoscopic video systems," in *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*, pp. 36–48, International Society for Optics and Photonics, 1993.

[43] R. Spottiswoode, N. Spottiswoode, and C. Smith, "3-d photography–basic principles of the three-dimensional film," *J. SMPTE*, vol. 59, pp. 249–286, 1952.

[44] "UITS Research Technologies." `http://rt.uits.iu.edu/visualization/avl/`, 2013. Advanced Visualization Lab. Indiana University.

[45] C. Zahumenszky, "Panasonic lanza la AG-3DP1, su segunda cámara 3d profesional." `http://www.xataka.com/videocamaras/panasonic-lanza-la-ag-3dp1-su-segunda-camara-3d-profesional`, 2011.

[46] J. J. Barceló, "La fábrica de ideas. JB steadycam 2000 - v.2.0." `http://jjelectra.blogspot.com.es/2009/11/jb-steadycam-2000-v20.html`, 2009.

[47] "Sky3D web page." `http://www.sky.com/shop/tv/3d/`, 2013.

[48] "Canal+3D web page." `http://www.plus.es/canal/canal-plus-3d/cp3d`, 2013.

[49] A. Vetro, "Frame compatible formats for 3D video distribution," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pp. 2405–2408, IEEE, 2010.

[50] "HDMI High-Definition Multimedia Interface." `http://www.hdmi.org/`, 2003–2013. HDMI Specification v1.4a, March 2010.

[51] "Digital video broadcasting (DVB); Specification for the use of Video and Audio Coding in Broadcasting Applications based on the MPEG-2 Transport Stream." ETSI TS 101 154 v1.10.1, June 2011.

[52] "Information Technology – Coding of audio-visual objects – Part 10: Advanced Video Coding." ITU-T Recommendation H.264 / ISO/IEC 14496-10:2010, March 2010.

[53] "Dolby® Open Specification for Frame-Compatible 3D Systems," tech. rep., Dolby Laboratories, Inc., 2010.

[54] "Ensuring premium-quality distribution of 3D content: A comparative image-fidelity analysis of various frame-compatible formats with and without video compression," tech. rep., SENSIO Technologies Inc., 2011. SENSIO® HI-FI 3D.

[55] F. Speranza, R. Renaud, A. Vincent, and W. J. Tam, "Perceived picture quality of frame-compatible 3DTV video formats," in *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, pp. 640–645, IEEE, 2012.

[56] "Digital Video Broadcasting (DVB): HDTV Service Compatible Plano-Stereoscopic 3DTV." DVB BlueBook A154-3, July 2012.

[57] "Digital video broadcasting (DVB); Plano-Stereoscopic 3DTV; Part 3: HDTV Service Compatible Plano-Stereoscopic 3DTV." ETSI TS 101 547-3 v1.1.1, November 2012.

[58] "3DTV service compatible frame packing format," tech. rep., Sisvel Technology, 2011.

[59] A. Bourge, J. Gobert, and F. Bruls, "MPEG-C part 3: Enabling the introduction of video plus depth contents," in *Proc. of IEEE Workshop on Content Generation and Coding for 3D-television, Eindhoven, The Netherlands*, 2006.

[60] "Information Technology – MPEG video technologies – Part 3: Representation of auxiliary video and supplemental information." ISO/IEC 23002-3:2007, October 2007.

[61] S. Pastoor and M. Wöpking, "3-d displays: A review of current technologies," *Displays*, vol. 17, no. 2, pp. 100–110, 1997.

[62] S. Benton, *Selected papers on three-dimensional displays.* SPIE milestone series, SPIE Optical Engineering Press, 2001.

[63] N. S. Holliman, N. A. Dodgson, G. E. Favalora, and L. Pockett, "Three-dimensional displays: a review and applications analysis," *Broadcasting, IEEE Transactions on*, vol. 57, no. 2, pp. 362–371, 2011.

[64] "Slovakia SuperComputers. 3D Glasses and filters." `http://www.supercomputers.sk/3d-glasses-and-filters/`, 2012.

[65] "El blu-ray 3d llenará las casas de gafas como éstas." `www.itespresso.es/el-blu-ray-3d-llenara-las-casas-de-gafas-como-estas-79382.html`, 2009.

[66] "Est crystal eyes 3 active shutter glasses." `http://www.est-kl.com/cn/products/shutterglasses/reald-stereographics/ce3.html`.

[67] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library.* O'Reilly Media, 2008.

[68] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach.* Always learning, Pearson Education, Limited, 2011.

[69] H. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, no. 293, pp. 133–135, 1989.

[70] T. Huang and O. Faugeras, "Some properties of the e-matrix in two-view motion estimation," in *IEEE Trans. Pattern Analysis and Machine Intelligence 11(12)*, pp. 1310—–1312, 1989.

[71] R. I. Hartley, "Theory and practice of projective rectification," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 115–127, 1999.

[72] J.-Y. Bouguet, "Camera calibration toolbox for Matlab." `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, 2010.

[73] R. Tsai, "A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 4, pp. 323–344, 1987.

[74] Z. Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, pp. 666–673, IEEE, 1999.

[75] Z. Zhang, "A flexible new technique for camera calibration," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 11, pp. 1330–1334, 2000.

[76] A. Fusiello, E. Trucco, and A. Verri, "A compact algorithm for rectification of stereo pairs," *Machine Vision and Applications*, vol. 12, no. 1, pp. 16–22, 2000.

[77] A. Fusiello and L. Irsara, "Quasi-euclidean uncalibrated epipolar rectification," in *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pp. 1–4, IEEE, 2008.

[78] F. Isgrò and E. Trucco, "Projective rectification without epipolar geometry," in *Proceedings of Computer Vision and Pattern Recognition, IEEE Conference on*, vol. 1, (Fort Collins, CO), pp. 94–99, 23-25 June 1999.

[79] T. Brunet and L. Chao, "Uncalibrated Stereo Vision." `http://pages.cs.wisc.edu/~chaol/cs766/`, 2004. Course: CS 766, University of Wisconsin, Madison.

[80] "Open source Computer Vision (openCV)." `http://opencv.org/`, 2008–2013.

[81] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Artificial Intelligence, International Joint Conference on*, pp. 674–679, 1981.

[82] C. Tomasi and T. Kanade, "Detection and tracking of point features," tech. rep., Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

[83] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, IEEE Conference on*, pp. 593–600, 1994.

[84] S. Birchfield, "Derivation of kanade-lucas-tomasi tracking equation." Unpublished, May 1996.

[85] C. Loop and Z. Zhang, "Computing rectifying homographies for stereo vision," tech. rep., Microsoft, April 1999. Microsoft Technical Report MSR-TR-99-21.

[86] S. Birchfield and C. Tomasi, "Depth discontinuities by pixel-to-pixel stereo," in *Computer Vision (ICCV), IEEE International Conference on*, (Bombay, India), pp. 1073–1080, January 1998.

[87] H. Ishikawa and D. Geiger, "Occlusions, discontinuities, and epipolar lines in stereo," in *Computer Vision—ECCV'98*, pp. 232–248, Springer, 1998.

[88] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 8, pp. 993–1008, 2003.

[89] A. F. Bobick and S. S. Intille, "Large occlusion stereo," *International Journal of Computer Vision*, vol. 33, no. 3, pp. 181–200, 1999.

[90] S. Hermann and T. Vaudrey, "The gradient-a powerful and robust cost function for stereo matching," in *Image and Vision Computing New Zealand (IVCNZ), 25th International Conference of*, pp. 1–8, IEEE, 2010.

[91] H. Hirschmuller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 9, pp. 1582–1599, 2009.

[92] S. Birchfield and C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling," *Pattern Analysis and Machine Intelligence, 1998 IEEE Transactions on*, vol. 20, pp. 401–406, 1998.

[93] F. Zhao, Q. Huang, and W. Gao, "Image matching by normalized cross-correlation," in *Acoustics, Speech and Signal Processing, 2006 IEEE International Conference on. ICASSP 2006 Proceedings*, vol. 2, pp. II–II, IEEE, 2006.

[94] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," *Computer Vision–ECCV94*, pp. 151–158, 1994.

[95] R. Kouskouridas, A. Gasteratos, and E. Badekas, "Evaluation of two-part algorithms for objects' depth estimation," *Computer Vision, IET*, vol. 6, no. 1, pp. 70–78, 2012.

[96] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[97] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006.

[98] P. Dev, "Segmentation processes in visual perception: A cooperative neural model," tech. rep., University of Massachusetts at Amherst, 1974. COINS Technical Report 74C-5.

[99] D. Marr and T. Poggio, "Cooperative computation of stereo disparity," tech. rep., DTIC Document, 1976.

[100] J. Marroquin, "Design of cooperative networks," 1983.

[101] R. Szeliski and G. Hinton, "Solving random-dot stereograms using the heat equation," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'85)*, pp. 284–288, 1985.

[102] D. Scharstein and R. Szeliski, "Stereo matching with nonlinear diffusion," *International Journal of Computer Vision*, vol. 28, no. 2, pp. 155–174, 1998.

[103] C. L. Zitnick and T. Kanade, "A cooperative algorithm for stereo matching and occlusion detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 7, pp. 675–684, 2000.

[104] P. Aschwanden and W. Guggenbuhl, "Experimental results from a comparative study on correlation-type registration algorithms," *Robust computer vision*, pp. 268–289, 1992.

[105] D. N. Bhat and S. K. Nayar, "Ordinal measures for image correspondence," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 4, pp. 415–423, 1998.

[106] O. Faugeras, T. Viéville, E. Theron, J. Vuillemin, B. Hotz, Z. Zhang, L. Moll, P. Bertin, H. Mathieu, P. Fua, *et al.*, "Real-time correlation-based stereo: algorithm, implementations and applications," 1993.

[107] V. S. Kluth, G. W. Kunkel, and U. A. Rauhala, "Global least squares matching," in *Geoscience and Remote Sensing Symposium, 1992. IGARSS'92. International*, vol. 2, pp. 1615–1618, IEEE, 1992.

[108] F. Bignone, O. Henricsson, P. Fua, and M. Stricker, "Automatic extraction of generic house roofs from high resolution aerial imagery," in *Computer Vision—ECCV'96*, pp. 83–96, Springer, 1996.

[109] S. Birchfield and C. Tomasi, "Multiway cut for stereo and motion with slanted surfaces," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1, pp. 489–495, IEEE, 1999.

[110] U. R. Dhond and J. K. Aggarwal, "Structure from stereo-a review," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 19, no. 6, pp. 1489–1510, 1989.

[111] S. Randriamasy and A. Gagalowicz, "Region based stereo matching oriented image processing," in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pp. 736–737, IEEE, 1991.

[112] C. Schmid and A. Zisserman, "The geometry and matching of curves in multiple views," in *Computer Vision—ECCV'98*, pp. 394–409, Springer, 1998.

[113] V. Venkateswar and R. Chellappa, "Hierarchical stereo and motion correspondence using feature groupings," *International Journal of Computer Vision*, vol. 15, no. 3, pp. 245–269, 1995.

[114] P. N. Belhumeur, "A bayesian approach to binocular steropsis," *International Journal of Computer Vision*, vol. 19, no. 3, pp. 237–260, 1996.

[115] I. J. Cox, S. L. Hingorani, S. B. Rao, and B. M. Maggs, "A maximum likelihood stereo algorithm," *Computer vision and image understanding*, vol. 63, no. 3, pp. 542–567, 1996.

[116] S. S. Intille and A. F. Bobick, "Incorporating intensity edges in the recovery of occlusion regions," in *Pattern Recognition, 1994. Vol. 1-Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, vol. 1, pp. 674–677, IEEE, 1994.

[117] Y. Ohta and T. Kanade, "Stereo by intra-and inter-scanline search using dynamic programming," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 2, pp. 139–154, 1985.

[118] C. Tomasi and R. Manduchi, *Stereo without search*. Springer, 1996.

[119] C. Tomasi and R. Manduchi, "Stereo matching as a nearest-neighbor problem," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 3, pp. 333–340, 1998.

[120] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.

[121] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2, pp. 508–515, IEEE, 2001.

[122] S. Roy and I. J. Cox, "A maximum-flow formulation of the n-camera stereo correspondence problem," in *Computer Vision, 1998. Sixth International Conference on*, pp. 492–499, IEEE, 1998.

[123] I. Thomo, S. Malasiotis, and M. G. Strintzis, "Optimized block based disparity estimation in stereo systems using a maximum-flow approach," in *Computer Graphics, Image Processing, and Vision, 1998. Proceedings. SIBGRAPI'98. International Symposium on*, pp. 410–417, IEEE, 1998.

[124] H. Zhao, "Global optimal surface from stereo," in *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, vol. 1, pp. 101–104, IEEE, 2000.

[125] A.-R. Mansouri, A. Mitiche, and J. Konrad, "Selective image diffusion: application to disparity estimation," in *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pp. 284–288, IEEE, 1998.

[126] J. Shah, "A nonlinear diffusion model for discontinuous disparity and half-occlusions in stereo," in *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR'93., 1993 IEEE Computer Society Conference on*, pp. 34–40, IEEE, 1993.

[127] J. Sun, N.-N. Zheng, and H.-Y. Shum, "Stereo matching using belief propagation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 7, pp. 787–800, 2003.

[128] O. Faugeras and R. Keriven, *Variational principles, surface evolution, PDE's, level set methods and the stereo problem.* IEEE, 2002.

[129] P. Fua and Y. G. Leclerc, "Object-centered surface reconstruction: Combining multi-image stereo and shading," *International Journal of Computer Vision*, vol. 16, no. 1, pp. 35–56, 1995.

[130] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," *International Journal of Computer Vision*, vol. 38, no. 3, pp. 199–218, 2000.

[131] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, pp. 807–814, IEEE, 2005.

[132] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *Pattern Analysis and Machine Intelligence (PAMI), 2008 IEEE Transactions on*, vol. 30, pp. 328–341, February 2008.

[133] M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing, "Real-time stereo vision: Optimizing semi-global matching," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, 2013.

[134] M. Bleyer and M. Gelautz, "A layered stereo matching algorithm using image segmentation and global visibility constraints," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 59, no. 3, pp. 128–150, 2005.

[135] C. L. Zitnick and S. B. Kang, "Stereo for image-based rendering using image oversegmentation," *International Journal of Computer Vision*, vol. 75, no. 1, pp. 49–65, 2007.

[136] K. Konolige, "Small vision systems: Hardware and implementation," in *Robotics Research-International Symposium-*, vol. 8, pp. 203–212, MIT PRESS, 1998.

[137] T. Tao, J. C. Koo, and H. R. Choi, "A fast block matching algorthim for stereo correspondence," in *Cybernetics and Intelligent Systems, 2008 IEEE Conference on*, pp. 38–41, IEEE, 2008.

[138] H. Ho, "2d-3d block matching," Master's thesis, Department of Computer Science, Faculty of Science. University of Auckland, New Zealand, 2005.

[139] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods

for markov random fields with smoothness-based priors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 6, pp. 1068–1080, 2008.

[140] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Map estimation via agreement on trees: message-passing and linear programming," *Information Theory, IEEE Transactions on*, vol. 51, no. 11, pp. 3697–3717, 2005.

[141] A. Blake, P. Kohli, and C. Rother, *Markov random fields for vision and image processing*. The MIT Press, 2011.

[142] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 2, pp. 147–159, 2004.

[143] O. Veksler, *Efficient graph-based energy minimization methods in computer vision*. PhD thesis, Cornell University, 1999.

[144] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, pp. 41–54, October 2006.

[145] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in *The British Machine Vision Conference*, pp. 989–998, 2006.

[146] R. Kalarot, J. Morris, and G. Gimel'farb, "Performance analysis of multi-resolution symmetric dynamic programming stereo on gpu," in *Image and Vision Computing New Zealand (IVCNZ), 2010 25th International Conference of*, pp. 1–7, IEEE, 2010.

[147] "MRF minimization page." `http://vision.middlebury.edu/MRF/`, 2006–2013.

[148] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 10, pp. 1568–1583, 2006.

[149] A. Geiger, M. Roser, and R. Urtasun, "Efficient Large-Scale Stereo Matching," in *Asian Conference on Computer Vision*, (Queenstown, New Zealand), November 2010.

[150] A. Geiger, "LIBELAS: Library for Efficient LArge-scale Stereo Matching." `http://www.cvlibs.net/software/libelas.html`, 2012.

[151] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision." `http://cs.brown.edu/~pff/bp/`, 2012.

[152] R. Farber, *CUDA application design and development.* Access Online via Elsevier, 2011.

[153] "CUDA Toolkit Documentation." `http://www.clear.rice.edu/comp422/resources/cuda/html/index.html`, 2013.

[154] NVIDIA™, *NVIDIA CUDA C Programming Guide.* Online available: `http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf`, may 2012.

[155] Q. Yang, L. Wang, and N. Ahuja, "A constant-space belief propagation algorithm for stereo matching," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 1458–1465, june 2010.

[156] S. Kosov, "Multi-view 3D reconstruction with variational method," Master's thesis, Department of Computer Science, Saarland University, Saarbrücken, Germany, January 2008.

[157] V. Kolmogorov and R. Zabih, "Multi-camera scene reconstruction via graph cuts," *Computer Vision–ECCV2002*, pp. 8–40, 2002.

[158] D. Scharstein and C. Pal, "Learning conditional random fields for stereo," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pp. 1–8, IEEE, 2007.

[159] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[160] R. Szeliski and R. Zabih, "An experimental comparison of stereo algorithms," *Vision Algorithms: Theory and Practice*, pp. 1–19, 2000.

[161] G. Egnal, M. Mintz, and R. P. Wildes, "A stereo confidence metric using single view imagery with comparison to five alternative approaches," *Image and Vision Computing*, vol. 22, no. 12, pp. 943 – 957, 2004. Proceedings from the 15th International Conference on Vision Interface.

[162] X. Hu and P. Mordohai, "Evaluation of stereo confidence indoors and outdoors," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 1466–1473, IEEE, 2010.

[163] X. Hu and P. Mordohai, "A quantitative evaluation of confidence measures for stereo vision," 2012.

[164] Z. Liu, "Performance evaluation of stereo and motion analysis on rectified image sequences," tech. rep., Computer Science Department, The University of Auckland, New Zealand, 2007.

[165] S. Morales and R. Klette, "A third eye for performance evaluation in stereo sequence analysis," in *Computer Analysis of Images and Patterns*, pp. 1078–1086, Springer, 2009.

[166] R. Klette, N. Kruger, T. Vaudrey, K. Pauwels, M. Van Hulle, S. Morales, F. I. Kandil, R. Haeusler, N. Pugeault, C. Rabe, *et al.*, "Performance of correspondence algorithms in vision-based driver assistance using an online image sequence database," *Vehicular Technology, IEEE Transactions on*, vol. 60, no. 5, pp. 2012–2026, 2011.

[167] M. Gong, R. Yang, L. Wang, and M. Gong, "A performance study on different cost aggregation approaches used in real-time stereo matching," *International Journal of Computer Vision*, vol. 75, no. 2, pp. 283–296, 2007.

[168] R. Kalarot and J. Morris, "Comparison of FPGA and GPU implementations of real-time stereo vision," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pp. 9–15, IEEE, 2010.

[169] K. Zhu, M. Butenuth, and P. d'Angelo, "Comparison of dense stereo using CUDA," in *Trends and Topics in Computer Vision*, pp. 398–410, Springer, 2012.

[170] "Visual Studio 2010." `http://msdn.microsoft.com/es-es/library/vstudio/dd831853(v=vs.100).aspx`, 2013.

[171] "Open source Computer Vision v2.4 Documentation." `http://docs.opencv.org/`, 2012.

[172] "Qt Project Download Page." `http://qt-project.org/downloads`, 2013.

[173] "Qt 4.8.0 Documentation." `http://qt-project.org/doc/qt-4.8/`, 2013.