EyeBoard: Un Periférico Alternativo Visual

Enrique Turégano Pedruelo

26 de septiembre de 2006

Introducción

En el mundo en el que vivimos, la información es poder y quien posee la información multiplica sus posibilidades de supervivencia. El almacenaje y procesamiento veloz de información facilita y agiliza la obtención de nuevos conocimientos utilizados, entre otros, para la toma de decisiones. Por ello surgió la tecnología informática. Etimológicamente la informática proviene de *información automática*. Es por ello que se ha erigido como la ciencia con mayor avance en la actualidad.

Los ordenadores o computadoras son los sistemas digitales capaces de procesar y almacenar información a partir de un grupo de instrucciones llamado programa. Para interactuar con un ordenador son necesarios dispositivos que permitan introducir y obtener información, estos dispositivos se conocen con el nombre de periféricos. Los periféricos con los que interactúan humanos y máquinas más comunes son: monitor, ratón y teclado.

En el diseño de tales periféricos se ha obviado la igualdad motriz entre humanos, lo cual supone un gran inconveniente para las personas con discapacidad física. Los periféricos de entrada de información más comunes, ratón y teclado están preparados para ser utilizados por personas que no tengan discapacidad en las extremidades superiores. Existen gran variedad de limitaciones en función de la discapacidad del usuario, por lo tanto resulta evidente que un aumento de la variedad de periféricos implica un incremento de personas que pueden interactuar con ordenadores.

Otra disciplina beneficiada de la implementación y creación de nuevos dispositivos es la HCI (Human Computer Interaction) cuyo objetivo es el estudio y la mejora de la interacción entre humanos y computadoras tanto a nivel software (interfaces de usuario) como a nivel hardware (nuevos periféricos).

El proyecto aquí presentado trata de buscar una alternativa a los periféricos tradicionales permitiendo y facilitando a personas discapacitadas el uso de computadores, así como crear un sistema que sirva como primera aproximación a

un dispositivo multimodal innovador.

0.1. Organización del libro

El libro está dividido en cuatro partes principales

Bases Teóricas introduce los conceptos de visión y percepción realizando una revisión histórica de la atención visual, explica los componentes y funciones del ojo humano, y hace un repaso de las técnicas de seguimiento de mirada actuales.

Desarrollo del Sistema explica el proceso de creación del sistema, desde las fases preliminares hasta la especificación completa, pasando por los procesos experimentales que han llevado ha implementarlo de manera eficiente, así como una descripción de todas las herramientas utilizadas.

Manuales explica al usuario con detalle la instalación y uso del sistema, y describe la estructura interna de la aplicación.

Anexos da una muestra de las aplicaciones con las que se puede utilizar el sistema así como posibles ampliaciones y mejoras para futuras versiones.

0.2. Agradecimientos

Al personal del Área de Robótica y Visión Artificial por su apoyo durante todo este tiempo, en especial a José Moreno, Pablo Bustos y Pilar Bachiller.

A Agustín, Álvaro y José por ayudarme con el “maravilloso” mundo del hardware.

A Gonzalo y a Pablo por su ayuda durante la realización de este proyecto.

A mis padres por su apoyo durante estos 5 años (...y decíais que no iba a ser capaz :)).

Índice general

Introducción III

0.1. Organización del libro IV

0.2. Agradecimientos IV

Lista de Figuras 1

I Bases Teóricas 5

1. Introducción a la Visión Humana 7

1.1. Atención Visual 7

1.1.1. Revisión Histórica de la Atención Visual 8

1.1.2. La Atención Visual y los Movimientos de los Ojos 11

1.2. El Ojo Humano 11

1.2.1. El Ojo 13

1.3. Psicofísica Visual 18

1.3.1. Visión Espacial 18

1.3.2. Visión Temporal 19

1.4. Clasificación y modelos de movimientos de ojos 21

1.4.1. Los Músculos Extraoculares 21

1.4.2. Sacádicos 22

1.4.3. Fijaciones 22

1.4.4. Persecuciones Lentas o Movimientos de Seguimiento . . . 22

1.4.5. Vergencias 22

1.4.6. Vestibulares 23
1.4.7. Nistagmos 23

2. Estado del Arte de los Eye Trackers 25
2.1. Técnicas de Seguimiento de mirada 25
2.1.1. Electro-oculografía 25
2.1.2. Lentes de Contacto 25
2.1.3. Foto-oculografía o Video-oculografía 26
2.1.4. Basados en Video con Detección de Pupila y Reflexión córnea 26
2.2. Eye Trackers Comerciales 27
2.2.1. VisionTrak© de Polhemus 28
2.2.2. VRET de la Universidad de Clemson 28

II Desarrollo 29

3. Fases de desarrollo del proyecto 31
3.1. Definición del Problema 31
3.1.1. Objetivos del Sistema 31
3.2. Estudio de Viabilidad 32
3.2.1. Modelo Propuesto 37
3.2.2. Coste del Sistema 38
3.2.3. Estudio de Viabilidad 38
3.3. Análisis del sistema 39
3.3.1. Diagrama de Contexto del Sistema 39
3.3.2. DFD's del Sistema 39
3.3.3. Descripción de los Componentes del DFD del Sistema . . 41

4. Desarrollo Experimental 45
4.1. Desarrollo Experimental del Hardware 45
4.1.1. Cambio de Filtros 45
4.1.2. Sustitución de Baterías por Transformador 47

4.1.3. Adición de un Nuevo LED 47
4.1.4. Cambio de la Posición de la Cámara del Mundo 48
4.2. Desarrollo Experimental del Software 48
4.2.1. Elección del Algoritmo de Detección de Pupila 48
4.2.2. Elección del Modelo Ocular 59
4.2.3. Elección del Método de Calibrado 60
4.2.4. Elección del Método de Interpolación 61
4.2.5. Detección de los Puntos de Calibración 65
4.2.6. Detección del Ojo Abierto 67
4.2.7. Escritura con los Ojos 67

5. Herramientas Utilizadas **71**
5.1. Sistema Operativo 71
5.1.1. Debian GNU/Linux 71
5.2. Programas 72
5.2.1. KDevelop 72
5.2.2. GCC 73
5.2.3. GDB 73
5.2.4. Doxygen 73
5.2.5. Gimp 73
5.2.6. Kimage 74
5.2.7. Kile 74
5.2.8. Microsoft Visio 75
5.3. Librerías 75
5.3.1. Qt 75
5.3.2. IPP 76
5.3.3. STL 77
5.3.4. OpenCV 77
5.3.5. OpenGL 77
5.3.6. FFmpeg 77

III Manuales **79**

6. Manual de Usuario **81**

6.1. Instalación y Puesta a Punto del Hardware 81

6.2. Instalación de las Librerías Necesarias 83

6.3. Compilación e Instalación del Programa 84

6.4. Configuración del Sistema con Eyeboardconfig 84

6.4.1. Fichero de Configuración del Sistema 86

6.5. Calibración y Uso de itune 86

6.6. Escritura con la Mirada con EyeBoard 89

6.7. Realización de Pruebas con infrared 89

7. Manual de Programador **93**

7.1. Diseño Global 93

7.2. Documentación de clases 95

7.2.1. Referencia de la Clase pt::readers::AviReader 95

7.2.2. Referencia de la Clase AviReader 101

7.2.3. Referencia de la Estructura pt::readers::AviReader::AviStreamHeader 102

7.2.4. Referencia de la Estructura pt::readers::AviReader::SAviMainHeader 104

7.2.5. Referencia de la Estructura pt::readers::AviReader::SAviOldIndex 106

7.2.6. Referencia de la Estructura pt::readers::AviReader::SAviOldIndexEntry 107

7.2.7. Referencia de la Estructura pt::readers::AviReader::STagBITMAPINFO 107

7.2.8. Referencia de la Estructura pt::readers::AviReader::STagBITMAPINFOHEADER 108

7.2.9. Referencia de la Estructura pt::readers::AviReader::STagRGBQUAD 110

7.2.10. Referencia de la Estructura pt::readers::AviReader::SWaveFormatEx 110

7.2.11. Referencia de la Clase pt::processing::BaseProcessing . . 111

7.2.12. Referencia de la Clase pt::readers::BaseReader 115
7.2.13. Referencia de la Clase pt::readers::CamReader 118
7.2.14. Referencia de la Clase pt::readers::Capture 123
7.2.15. Referencia de la Estructura pt::readers::Capture::buffer . . 128
7.2.16. Referencia de la Clase pt::math::Definitions 128
7.2.17. Referencia de la Clase pt::math::Distribution 129
7.2.18. Referencia de la Clase pt::eyeboard 131
7.2.19. Referencia de la Estructura pt::eyeboard::SKey 140
7.2.20. Referencia de la Clase pt::processing::EyeProcessing . . . 141
7.2.21. Referencia de la Clase pt::renderers::GLWindow 153
7.2.22. Referencia de la Clase pt::Image 157
7.2.23. Referencia de la Clase pt::math::Int2dPoint 161
7.2.24. Referencia de la Clase pt::math::Kriging 163
7.2.25. Referencia de la Estructura pt::math::Kriging::SPointValue 168
7.2.26. Referencia de la Clase pt::Logger 169
7.2.27. Referencia de la Clase pt::Profiler 173
7.2.28. Referencia de la Clase pt::Profiler::CProfile 176
7.2.29. Referencia de la Clase pt::math::Real2dPoint 180
7.2.30. Referencia de la Clase pt::Singleton< T > 182
7.2.31. Referencia de la Estructura pt::SPixel 183
7.2.32. Referencia de la Clase pt::renderers::VideoWriter 184
7.2.33. Referencia de la Clase pt::processing::WorldProcessing . . 197
7.3. Documentación de Archivos 203
7.3.1. Referencia del Archivo eyeboard/src/avireader.cpp 203
7.3.2. Referencia del Archivo eyeboard/src/avireader.h 204
7.3.3. Referencia del Archivo eyeboard/src/baseprocessing.cpp . 205
7.3.4. Referencia del Archivo eyeboard/src/baseprocessing.h . . 205
7.3.5. Referencia del Archivo eyeboard/src/basereader.cpp . . . 206
7.3.6. Referencia del Archivo eyeboard/src/basereader.h 206
7.3.7. Referencia del Archivo eyeboard/src/camreader.cpp 207
7.3.8. Referencia del Archivo eyeboard/src/camreader.h 207

7.3.9. Referencia del Archivo eyeboard/src/capture.cpp 208
7.3.10. Referencia del Archivo eyeboard/src/capture.h 209
7.3.11. Referencia del Archivo eyeboard/src/distribution.cpp . . . 211
7.3.12. Referencia del Archivo eyeboard/src/distribution.h 211
7.3.13. Referencia del Archivo eyeboard/src/eyeboard.cpp 212
7.3.14. Referencia del Archivo eyeboard/src/eyeboard.h 212
7.3.15. Referencia del Archivo eyeboard/src/eyeprocessing.cpp . 214
7.3.16. Referencia del Archivo eyeboard/src/eyeprocessing.h . . . 214
7.3.17. Referencia del Archivo eyeboard/src/glwindow.cpp 215
7.3.18. Referencia del Archivo eyeboard/src/glwindow.h 216
7.3.19. Referencia del Archivo eyeboard/src/image.cpp 217
7.3.20. Referencia del Archivo eyeboard/src/image.h 217
7.3.21. Referencia del Archivo eyeboard/src/kriging.cpp 217
7.3.22. Referencia del Archivo eyeboard/src/kriging.h 218
7.3.23. Referencia del Archivo eyeboard/src/logger.cpp 218
7.3.24. Referencia del Archivo eyeboard/src/logger.h 219
7.3.25. Referencia del Archivo eyeboard/src/main.cpp 221
7.3.26. Referencia del Archivo eyeboard/src/maths.cpp 222
7.3.27. Referencia del Archivo eyeboard/src/maths.h 223
7.3.28. Referencia del Archivo eyeboard/src/profiler.cpp 223
7.3.29. Referencia del Archivo eyeboard/src/profiler.h 223
7.3.30. Referencia del Archivo eyeboard/src/singleton.cpp 224
7.3.31. Referencia del Archivo eyeboard/src/singleton.h 224
7.3.32. Referencia del Archivo eyeboard/src/videowriter.cpp . . . 225
7.3.33. Referencia del Archivo eyeboard/src/videowriter.h 225
7.3.34. Referencia del Archivo eyeboard/src/worldprocessing.cpp 226
7.3.35. Referencia del Archivo eyeboard/src/worldprocessing.h . 226

IV Anexos **229**

8. Posibles ampliaciones y mejoras del sistema **231**

8.1. Ampliaciones y Mejoras del Hardware 231
8.2. Ampliaciones y Mejoras del Software 233

9. Aplicaciones del sistema **235**

Bibliografía **239**

Índice **240**

Índice de figuras

1.1. Ilusión de Kanizsa 9
1.2. Caminos visuales 12
1.3. Lóbulos corticales 13
1.4. Ojo Humano 14
1.5. Vista lateral del ojo humano 15
1.6. Vista superior del ojo 15
1.7. Lentes enfocando 16
1.8. Vista frontal del ojo 17
1.9. Ángulo visual 19
1.10. Conos y bastones 20
1.11. Conos y bastones 20
1.12. Músculos oculares 21

2.1. EOG 26
2.2. Reflexiones Purkinje 27
2.3. VisionTrak© de Polhemus 28

3.1. Teclado ajustable de Apple 32
3.2. Ratón Apple 33
3.3. Joystick 33
3.4. Trackball de Logitech 34
3.5. Nintendo DS 34
3.6. Tableta Digitalizadora 35
3.7. EyeToy de Sony PlayStation 36

4.1. Soporte para las cámaras 46
4.2. Filtro infrarrojos 46
4.3. Filtro infrarrojos 46
4.4. Diferencias de calidad 47
4.5. Baterías 47
4.6. Alimentación LEDS 48
4.7. Ojo en infrarrojos 49
4.8. Diferencia ojo abierto-cerrado 49
4.9. Parámetros de Hough 54
4.10. Etapas de la captura 56
4.11. Método de las parejas de bordes 57
4.12. Pupila: vista lateral 58
4.13. Método de los pares de gradientes 58
4.14. Modelo geométrico simple 59
4.15. Ejemplo de Kriging 64
4.16. Problema con la doble reflexión 65
4.17. Punto de calibración 66
4.18. Comparativa de histogramas de ojos abierto y cerrado 68
4.19. Predicción del punto de interés 69

5.1. Logo de Debian GNU/Linux 71
5.2. GNU Image Manipulation Program 74
5.3. Logo de la librería Qt 75

6.1. Capturadoras y PCI's libres 82
6.2. Cables RCA de las capturadoras 82
6.3. EyeBoardConfig 85
6.4. itune - configuración 87
6.5. itune - calibración 88
6.6. itune - calibración 88
6.7. EyeBoard 90
6.8. Interfaz de infrared 90

6.9. Comportamiento frente a un concierto 91

7.1. Mapa de estructura global del sistema 94

8.1. Cámara inalámbrica 232

9.1. Ejemplo del cálculo del punto de mirada 236

Parte I

Bases Teóricas

Capítulo 1

Introducción a la Visión Humana

1.1. Atención Visual

Antes de centrarnos en la detección de los ojos, pupilas y el seguimiento de mirada, debemos plantearnos cuáles son las motivaciones que nos impulsan a analizar el comportamiento de los ojos humanos, ¿por qué es tan importante el seguimiento de los ojos?.

Cuando movemos los ojos hacia un determinado lugar, lo que realmente estamos haciendo es poner una zona de nuestro espacio de visión en alta resolución para poder apreciar con detalle lo que está situado en el centro de nuestro punto de vista. Es habitual en los humanos desviar la mirada hacia zonas con menor interés para conseguir una mayor concentración. Por lo tanto podemos asumir que si realizamos un seguimiento de los ojos de un sujeto, podremos saber el camino de atención que está siguiendo el individuo. De esta forma se puede averiguar qué partes de la escena, son más interesantes para el sujeto y además podemos intuir cómo percibe la escena.

En primer lugar debemos definir la atención de forma intuitiva.

El psicólogo James da una buena definición cualitativa de la atención visual:

"Todos sabemos qué es la atención. Es el hecho de que la mente tome conciencia de forma clara de un objeto entre varios posibles. Focalizarlo, y concentrarse en él son esenciales para la atención. Esto implica centrarse en un objeto, descartando los demás"

Solo determinados estímulos pueden ser atendidos por los sentidos, el resto son descartados.

Los seres humanos no pueden atender a todas las cosas a la vez. En general, la atención es utilizada para centrar nuestras capacidades mentales en determinadas entradas de los sentidos, de forma que la mente pueda procesar los estímulos de

interés. Nuestra capacidad es limitada. El cerebro se centra en determinados componentes para aumentar su atención en ellos, en detrimento del resto.

1.1.1. Revisión Histórica de la Atención Visual

Hace más de un siglo que se empezó a estudiar la atención visual. Los primeros estudios se limitaban a simples observaciones oculares. Desde entonces, este campo ha crecido mucho, implicando a varias ramas de la ciencia. En esta sección se muestra una revisión de las ideas principales a lo largo de la historia.

1.1.1.1. El "dónde" de Von Helmholtz

En la segunda mitad del siglo 19, Von Helmholtz (1925) pensaba que la atención visual era un mecanismo esencial en la percepción visual. En su *Tratado sobre óptica fisiológica* dijo: "Dejamos que nuestros ojos vaguen continuamente sobre el campo visual, porque es la única forma que tenemos de ver todos los objetos de nuestro campo tan distintamente como se pueda". También observó que la tendencia natural de la atención se centra en los objetos nuevos. También señaló que la atención puede ser controlada voluntariamente moviendo los ojos.

1.1.1.2. El "qué" de James

En contraste a Von Helmholtz, James(1981) creía que la atención oculta un mecanismo interno similar a la imaginación, o al pensamiento. James definía la atención en términos de "qué", asociados con el centro de nuestra atención. Aunque estaba a favor de los aspectos voluntarios de la atención, también reconocía la atención pasiva, por reflejos, sin esfuerzo e involuntaria.

1.1.1.3. El "cómo" de Gibson

En 1941 Gibson propuso un tercer factor de la atención visual centrado en la intención. Las ideas de Gibson estaban relacionadas con el conocimiento previo del sujeto, con sus reacciones, y si reaccionaba, cómo y con qué tipo de respuestas. La respuesta varía en función de la preparación previa del sujeto. Por ejemplo si se le dice al sujeto que espere palabras representando animales y recibe una palabra que sea "deflín" el sujeto leerá "delfín". Según Gibson, la información previa que pueda tener el sujeto condiciona el "qué hacer" o el "cómo" reaccionar ante diversos estímulos.

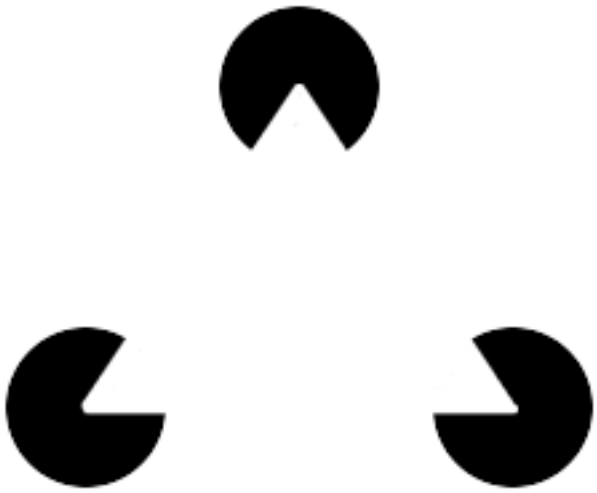


Figura 1.1: Ilusión de Kanizsa

1.1.1.4. El “filtro selectivo” de Broadbent

La atención se ve como un ”filtro selectivo“ que regula la información sensorial. En 1958 Broadbent realizó unos experimentos auditivos en los que los sujetos recibian una secuencia de números por la izquierda {7,2,3} y otros por la derecha {9,4,5}. Los sujetos decían que habian escuchado {7,2,3,9,4,5} o {9,4,5,7,2,3} sin ningún intercalado. La conclusión de Broadbent fue que la información llega en paralelo pero es filtrada selectivamente.

1.1.1.5. La “importancia de los pesos” de Deutsch y Deutsch

En contraste a las ideas de Broadbent, Deutsch y Deutsch propusieron que todos los mensajes sensoriales son analizados al nivel más alto, sin necesidad de un filtro selectivo. Rechazaron la idea de un filtro selectivo y la capacidad limitada del sistema de atención, argumentaban que si el filtro existiese, tendría que ser al menos tan complejo como la capacidad del sistema en si misma. Propusieron la existencia de estructuras centrales con “pesos importantes” que determinaban la selección, es decir, la atención era el resultado de la importancia o relevancia interactuando con la información.

En los años 60 Anne Treisman propuso una teoría unificada en la que unía las teorías de Broadbent con las de Deutsch y Deutsch. Aún suponiendo que la teoría de Treisman fuera verdadera, queda un problema por resolver, referido a la forma de integrar la escena: ¿Cómo se mezclan todas las piezas en nuestra mente para formar algo coherente?. Un ejemplo del problema de la integración de la escena es

la "ilusión de Kanizsa" de la figura 1.1. En ella se ve el contorno de un triángulo a pesar de que éste está definido solo por las muescas en los círculos. Aún no se sabe completamente cómo el cerebro hace para integrar esta imagen.

1.1.1.6. El Recorrido Ocular de Yarbus y Noton y Stark

Las primeras grabaciones de los movimientos de los ojos sirvieron para poner en duda la hipótesis de Gestalt, que afirmaba que el reconocimiento es un proceso de un solo paso en paralelo. Las primeras grabaciones sirvieron para mostrar que el reconocimiento visual es en serie.

Yarbus en 1967 midió los movimientos de los ojos de ciertos sujetos sobre una imagen después de proponerle a los mismos ciertas cuestiones sobre la imagen. Yarbus demostró patrones de visión secuenciales sobre ciertas regiones de la imagen.

Noton y Stark realizaron sus propios experimentos y extendieron los resultados de Yarbus, concluyendo que a pesar de proponer determinadas cuestiones a los sujetos, éstos tienden a fijarse en determinadas zonas de interés. El recorrido ocular sobre estas regiones era diferente entre los sujetos. Por ejemplo, dado un cuadrado, éstos tienden a fijarse en las esquinas, pero cada uno en un orden diferente.

1.1.1.7. El Centro de Atención de Posner

Posner sugirió que el centro de atención es capaz de moverse por la escena. Posner *et al.* diferenciaron el centro de atención del punto de mira, y dijeron que el mecanismo de atención era diferente de los movimientos de los ojos. Según ellos, por un lado se encontraba la atención y por otro la orientación.

1.1.1.8. El Pegamento de Treisman

Treisman volvió a unificar las teorías en la Teoría de la Integración de Rasgos de la atención visual. Básicamente la atención es el pegamento que integra los diferentes rasgos de forma que el objeto sea percibido como un todo. La atención selecciona aquellos aspectos en función de "dónde" se encuentren y no según "qué" sean. La Teoría de La Integración de Rasgos (FIT) es una teoría particularmente importante en la atención visual.

1.1.1.9. La Ventana de Kosslyn

Kosslyn propuso un modelo más refinado de la atención visual en el que describe la atención como un aspecto selectivo del procesamiento perceptual, y propone una "ventana" de atención encargada de filtrar el campo visual. La ventana es necesita-

da, dado que hay mucha más información en el campo visual de la que puede ser procesada. La novedad de la ventana de atención es que puede ser reescalada.

1.1.2. La Atención Visual y los Movimientos de los Ojos

Los movimientos de los ojos pueden apoyar la teoría del "qué" y del "dónde". La visión se comporta muy probablemente como un proceso cíclico compuesto por los siguientes pasos:

1. Dado un estímulo, como una imagen, en primer lugar se ve la escena entera mediante la visión periférica en baja resolución. En este punto, los rasgos más interesantes sobresalen en el campo de visión centrando la atención en ellos.
2. Los ojos se centran en la primera región que llama la atención.
3. Una vez que los ojos han realizado su movimiento, la fóvea se dirige a la región de interés, y la atención se centra en la percepción del rasgo en alta resolución.

Este modelo es correcto pero incompleto, dado que existen varios asuntos no explicados:

1. Si únicamente el estímulo visual dirige la atención, ¿qué es exactamente lo que nos atrae de él?
2. Si el estímulo visual es el que atrae la atención, ¿para qué necesitamos los movimientos oculares voluntarios?
3. ¿Cuál es la relación entre la atención y los movimientos oculares?

En el siguiente capítulo se profundizará en el modelo.

1.2. El Ojo Humano

Existe una gran cantidad de información respecto al Sistema Humano Visual (HVS). Algunas características de la visión pueden ser estudiadas en función de los movimientos oculares y otras mediante la organización interna de las neuronas del cerebro.

Según la literatura neurofisiológica y psicofísica, la visión humana se produce realizando rápidas fijaciones sobre pequeñas áreas de interés. Esto permite que la fóvea perciba los detalles de la escena. La visión de la fóvea cubre entre 1 y 5 grados

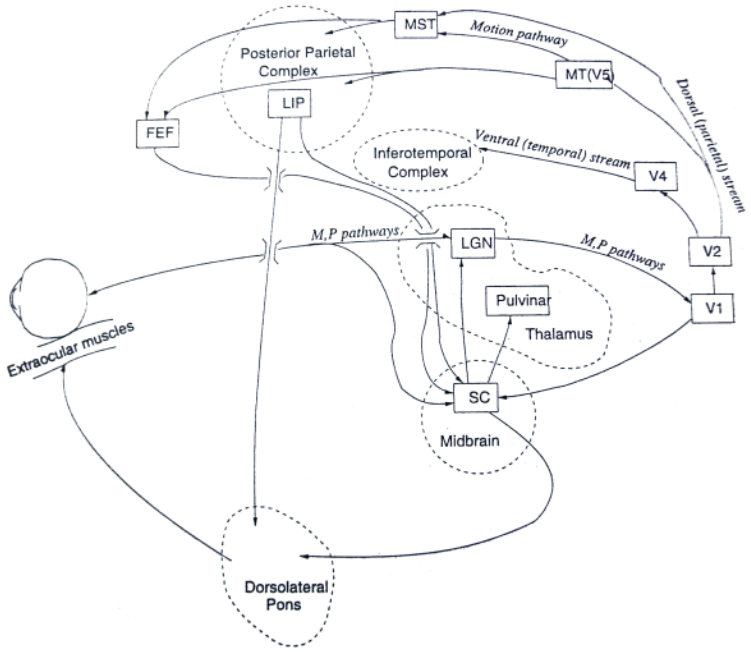


Figura 1.2: Vista simplificada de los caminos visuales. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

permitiendo una alta resolución de una pequeña pate del total, esto supone un 3 % del tamaño de un monitor a una distancia normal. Aproximadamente el 90 % del tiempo de la visión se gasta en fijaciones. Cuando la visión se dirige a una nueva área, los sacádicos reposicionan la fóvea frente a la imagen.

El cerebro está compuesto por un gran número de regiones clasificadas según la función que realizan. Una representación simplificada del cerebro se muestra en la figura 1.2 y el nombre de los lóbulos en la figura 1.3 El Sistema Humano Visual se encuentra descrito en función de las conexiones entre la retina y las regiones cerebrales, conocidas como caminos visuales. Las siguientes regiones neuronales son particularmente importantes en la percepción visual y en los movimientos oculares:

- Colículo Superior (SC): implicado en la preparación de los movimientos oculares y contribuye a la selección de objetos para mirar mediante sacádicos y persecuciones lentas (smooth pursuits).

- Área V1: se encarga de la detección de estímulos en función de las variaciones de color.

- Áreas V2, V3, V3A, V4, MT: procesamiento de la forma, el color, y el movimiento.

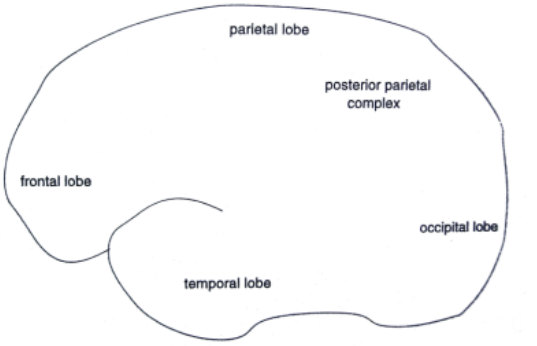


Figura 1.3: Lóbulos corticales. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

- Área V5/MT (Temporal Medio) y MST (Temporal Medio Superior): realiza proyecciones al puente (Pons), y MT proyecta al colículo, proporcionándole señales de movimiento del campo visual.
- Área LIP (Lateral Intra-Parietal): contiene campos receptores que son corregidos antes de cada sacádico.
- Complejo Parietal Posterior (PPC): implicado en las fijaciones.

1.2.1. El Ojo

Es el órgano de la visión (Figura 1.4), encargado de detectar luz. Los humanos poseen dos ojos, cuando se encuentran enfocando el mismo plano, proporcionan una visión tridimensional. A menudo se le conoce con el nombre de "peor cámara del mundo", ya que padece varias imperfecciones ópticas:

- Aberraciones esféricas: efecto prisma en los bordes de las lentes.
- Aberraciones cromáticas: longitudes de onda más cortas (azul) refractan más que longitudes de onda mayores (rojo).
- Curvatura del campo: un objeto plano parece una imagen curva.

Sin embargo, el ojo está dotado con varios mecanismos que reducen las imperfecciones:

- Para reducir la aberración esférica, el iris actúa como un filtro, limitando la entrada periférica de rayos.

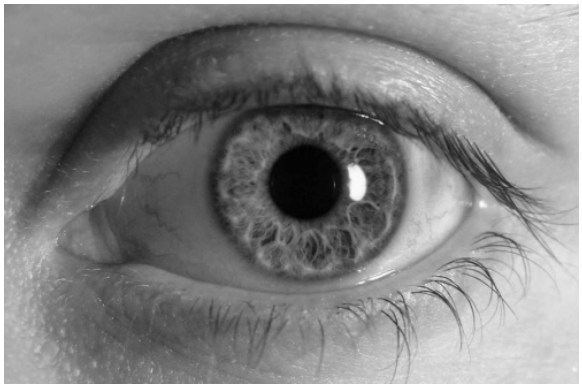


Figura 1.4: Ojo Humano

- Para evitar la aberración cromática, el ojo se centra en producir imágenes nítidas de longitudes de onda intermedias.
- Para evitar la curvatura de campo, la retina se curva compensando este efecto.

Como se puede apreciar en la vista lateral (Figura 1.5), y en la vista superior (Figura 1.6), el ojo se encuentra compuesto por varias partes:

1.2.1.1. La Córnea

La córnea es la estructura transparente situada al frente del ojo, permite el paso de la luz hacia el interior y protegiendo al iris y al cristalino. Posee propiedades ópticas de refracción, representando cerca de 2/3 de la capacidad de enfoque del ojo.

Sirve como estructura continente del globo ocular, junto con la esclerótica. Es uno de los pocos tejidos que no poseen irrigación sanguínea, dado que se nutre de la lágrima y del humor acuoso.

Es muy sensible al tacto, es la porción anatómica del cuerpo humano que posee más terminaciones nerviosas sensoriales.

Por término medio, mide 1/2 milímetro.

Se encuentra compuesta por 5 capas:

- Epitelio
- Membrana de Bownman
- Estroma

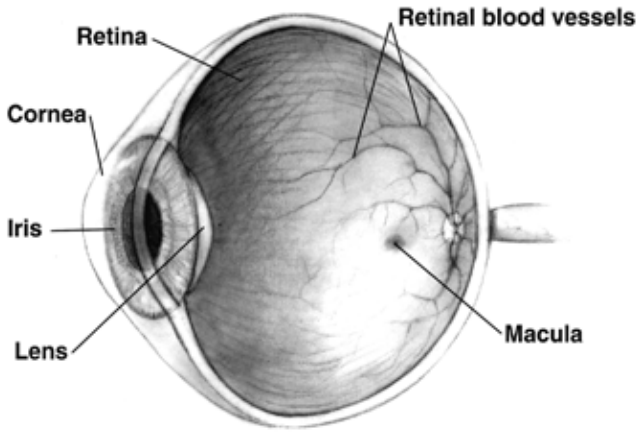


Figura 1.5: Vista lateral del ojo humano

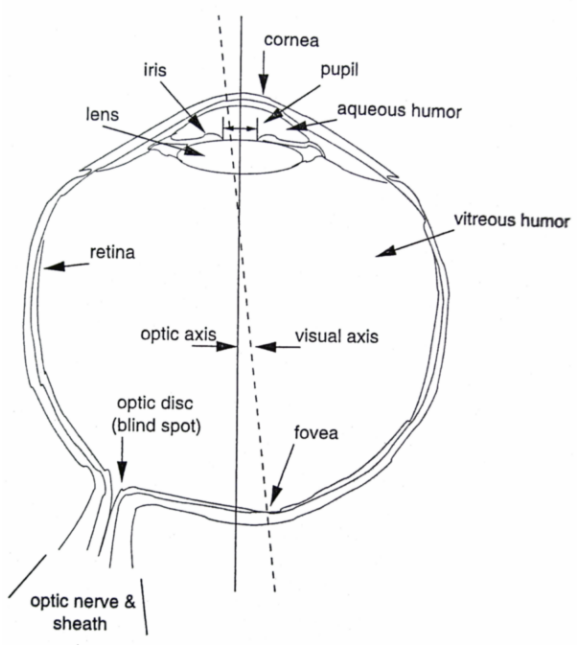


Figura 1.6: Vista superior del ojo humano. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

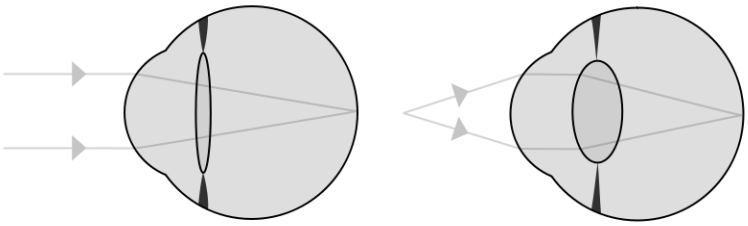


Figura 1.7: Efecto de las lentes al enfocar un objeto a diferentes distancias

- Membrana de Descemet
- Endotelio

1.2.1.2. El Cristalino

Se encarga de refractar la luz para proyectarla en la retina. Se trata de un cuerpo lenticular, trasparente y biconvexo, está situado en la parte anterior del globo ocular, entre el humor acuoso y el humor vítreo. Está compuesto por una parte exterior, llamada corteza, y una interior, llamada núcleo. Su función es enfocar los rayos de luz para que formen una imagen perfecta en la retina.

1.2.1.3. La Esclerótica

Es una membrana de color blanco, gruesa, resistente y fibrosa. Constituye la parte exterior del globo ocular, es fácilmente reconocible por ser la "parte blanca del ojo".

1.2.1.4. La Pupila

La pupila es el parte central del iris. Se trata de una abertura dilatable y contráctil de color negro con la función de regular la iluminación que le llega a la retina, en la parte posterior del ojo. Mide entre 2 y 8 mm.

1.2.1.5. El Limbo

El limbo es la frontera entre la córnea y la esclerótica. No es un tejido como tal, sino más bien una zona de importancia en visión artificial.

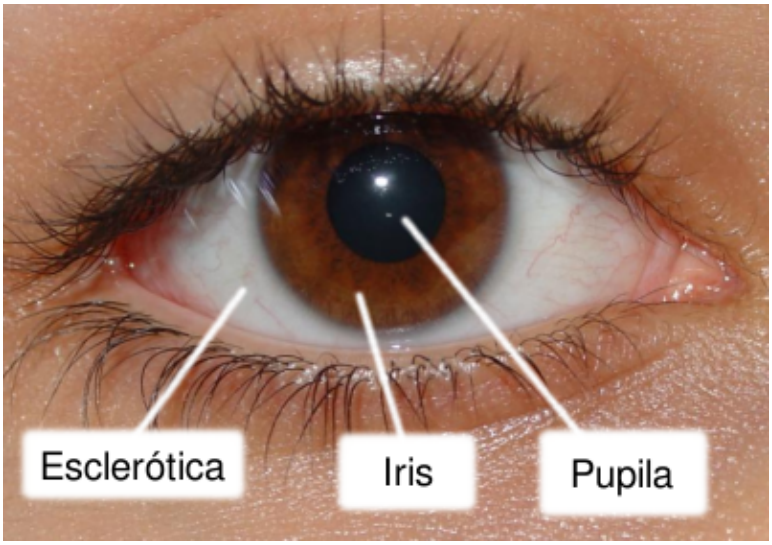


Figura 1.8: Vista frontal del ojo

1.2.1.6. El Iris

El iris es una membrana coloreada y circular que separa las cámara anterior y posterior del ojo. Posee una apertura central de tamaño variables, la pupila que comunica la cámara anterior del ojo con la cámara posterior.

Está constantemente activa permitiendo a la pupila dilatarse (midriasis) o contraerse (miosis).Está función tiene su objetivo en la regulación de la cantidad de luz que llega a la retina.

1.2.1.7. Humor Vítreo

Es un líquido situado en la cámara posterior del ojo, entre la lente y la retina, formado en un 99.98 % por agua y el resto por sales minerales.

1.2.1.8. Humor Acuoso

Es un líquido claro y aguado que fluye por los espacios difíciles en la parte frontal del ojo, limitado por la córnea delante y por el humor vítreo detrás. El humor acuoso permite que los nutrientes circulen por el endotelio corneal y que la presión mantenga a la córnea convexa.

1.2.1.9. La Retina

Se encuentra situada en la parte trasera de la superficie interior del ojo, contiene receptores sensibles a la luz (fotorreceptores), que constituyen la primera etapa de la percepción visual. Los fotorreceptores convierten la luz en impulsos eléctricos, las señales originadas en los receptores, van a a parar a partes a los centros encargados de la visión en el cerebro. Los fotorreceptores se clasifican en conos y bastontes. Los bastones son sensibles a la oscuridad y a la luz sin color (visión nocturna) mientras que los conos responden a la luz de color (visión diurna). La retina contiene aproximadamente 120 millones de bastones.

1.2.1.10. La Fóvea

La fóvea está situada en el centro de la mácula. Es responsable de la visión nítida central. La zona que rodea la fóvea, que posee una menor resolución, se conoce con el nombre de perifóvea.

Mide alrededor de 0.2 mm de diámetro, y posee una alta concetración de conos, y ninguna de bastones. Los conos no se encuentran obstruidos por ninguna capa superior, ni siquiera por vasos sanguíneos. Es responsable de la visión en color en los humanos, que es ligeramente superior a la del resto de mamíferos.

Se encuentra desplazada entre 4 y 8 grados respecto al eje óptico.

Debido a que no tiene bastones, a los humanos nos resulta imposible distinguir con claridad pequeñas luces si las miramos directamente, por ello en condiciones límite, recurrimos a la visión periférica.

1.3. Psicofísica Visual

El rendimiento visual de los humanos es consecuencia de complejos procesos visuales. Conociendo la parte fisiológica del sistema de visión humana, se puede medir el rendimiento de ciertos parámetros.

1.3.1. Visión Espacial

Las características dimensionales de la retina son descritos normalmente en función de los grados visuales de la escena proyectada, definidos como

$$A = 2 \arctan \frac{S}{2D}$$

donde S es el tamaño del objeto en la escena y D es la distancia al objeto (Ver imagen 1.9). La parte central de la fóvea (foveola) mide 0.4mm de diámetro y

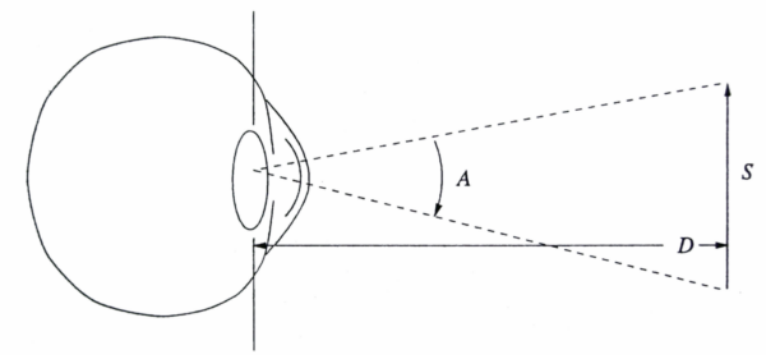


Figura 1.9: Ángulo Visual. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

contiene 25.000 conos. La fóvea en sí mide 1.5mm de diámetro y tiene 10.0000 conos. La mácula (retina central) mide 5mm de diámetro y contiene 650.000 conos. Un grado visual corresponde a aproximadamente 0.3mm de la retina humana. La foveola, midiendo 0.4mm abarca 1.3° mientras que la fóvea y la mácula abarcan 5 y 16.7° respectivamente (ver figura 1.10).

El campo visual útil es de unos 30° aproximadamente. El resto del campo visual apenas tiene potencia de resolución y se utiliza principalmente para la percepción del movimiento. Los conos aumentan de tamaño cuanto más alejados se encuentren de la fóvea, mientras que los bastones son del mismo tamaño. Los conos realizan la mayor contribución a la información que va al cerebro, proporcionando información mas detallada al sistema visual.

1.3.2. Visión Temporal

La respuesta visual humana al movimiento se caracteriza por dos hechos distintos: *la persistencia de la visión* y el *fenómeno phi* (o efecto empalme).

La persistencia de la visión indica que la retina es incapaz de distingir intensidades que cambian rápidamente. Cualquier estímulo que parpadee a una frecuencia superior a 50-60Hz será percibido por la retina como estático. En los cines, por ejemplo, las películas son mostradas a 24 frames por segundo y cada frame es mostrado 3 veces, de forma que la retina percibe esos 72Hz como continuos.

El fenómeno phi explica porqué funcionan las películas y la televisión. Fue descrito por primera vez en 1875 por Erner. Afirmó que cuando dos chispas son producidas en instantes de tiempo muy cercanos, se produce la sensación de movimiento desde la posición de la primera hasta la posición de la segunda.

La zona central de la retina, la fóvea es mucho más sensible al movimiento que

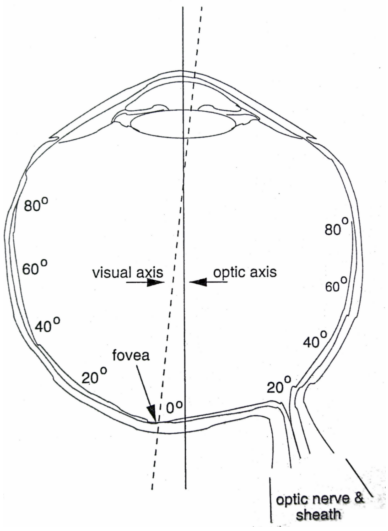


Figura 1.10: Distribución de la densidad de conos y bastones en la retina. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

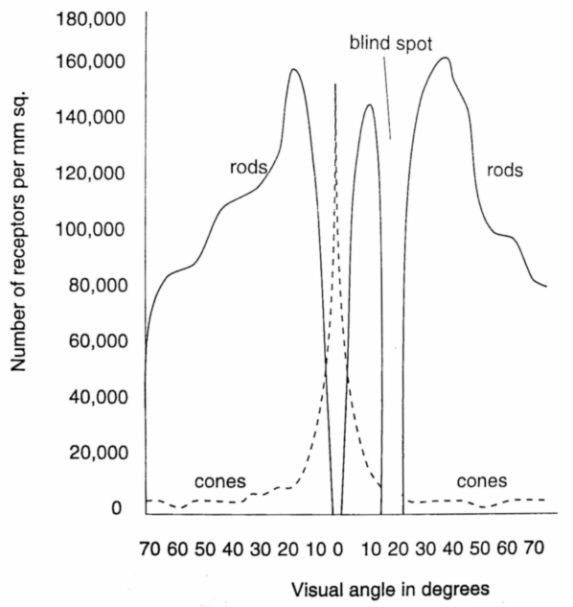


Figura 1.11: Distribución de la densidad de conos y bastones en la retina. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

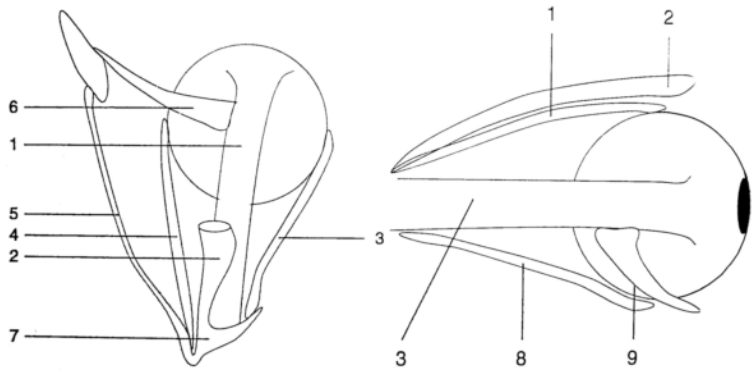


Figura 1.12: Músculos oculares: 1 recto superior, 2 elevador del párpado superior, 3 recto lateral, 4 recto medio, 5 oblícuo superior, 6 tendón reflejado del oblícuo superior, 7 anillo de Zinn, 8 recto inferior, 9 oblícuo inferior. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

las zonas alejadas, disminuyendo esta sensibilidad cuanto más nos alejemos de la fóvea.

1.4. Clasificación y modelos de movimientos de ojos

Todos los movimientos oculares utilizados por los primates son combinaciones de 5 tipos básicos: sacádicos, persecuciones lentas, vergencias, vestibulares y nistagmos. Existen otros movimientos asociados, pero que no son utilizados para el reposicionamiento de la fóvea, como la dilatación de la pupila, o el ajuste de las lentes.

1.4.1. Los Músculos Extraoculares

Los ojos se mueven con seis grados de libertad, tres traslaciones, y tres rotaciones. Existen seis músculos responsables del movimiento ocular: el recto medio y lateral (movimientos hacia los lados), el recto superior e inferior (movimientos hacia arriba y hacia abajo), y los oblícuos superior e inferior (rotaciones). Se encuentran descritos en la figura 1.12. El sistema encargado del movimiento ocular forma un circuito realimentado y las señales que controlan el movimiento de los ojos provienen de regiones del córtex que proporcionan movimientos voluntarios (córtex occipital), involuntarios (colículo superior) y reflejos (canales semicirculares). Este circuito de realimentación es utilizado principalmente para estabilizar el movimiento ocular.

1.4.2. Sacádicos

Son los movimientos más conocidos de los ojos. Son voluntarios (exceptuando la fase REM del sueño) y sirven para poder visualizar distintas partes de una escena. Sirven para que la imagen de la escena sea proyectada en la fóvea, que posee una mayor agudeza visual. Durante la duración del sacádico, mientras el ojo se mueve, se produce un fenómeno llamado *supresión sacádica*, llamado así porque la recogida de información queda prácticamente interrumpida.

Existe una relación entre la cantidad de movimiento que se realiza y la velocidad adquirida por el ojo, cuanto mayor es el ángulo a desplazar, mayor es la velocidad alcanzada. Un sacádico que se mueva 80° alcanzará una velocidad de 700 grados por segundo. Por otro lado, la duración de un sacádico depende de su magnitud, y en general se encuentra entre 30 y 120 milisegundos. Por término medio, un humano pasa 90 minutos al día realizando movimientos sacádicos. Si el sacádico tiene que cubrir más de 30°, la cabeza interviene en el movimiento también. Se ha comprobado que entre la aparición de un estímulo, y el comienzo del movimiento transcurren entre 180 y 300 milisegundos, además entre dos sacádicos consecutivos existe un periodo refractario de unos 150 milisegundos.

1.4.3. Fijaciones

A cada sacádico le sigue una fijación, y entre ellos se producen interesantes interacciones psicológicas. Durante una fijación no sólo se procesa la información recogida por la fóvea, sino también la información periférica, en busca de estímulos.

1.4.4. Persecuciones Lentas o Movimientos de Seguimiento

Son movimientos producidos de forma simultánea por ambos ojos, y cuya finalidad es la de situar sobre la fóvea ciertos estímulos que se están desplazando lentamente. El ojo se mueve a 1-30° por segundo cuando realiza estos movimientos. En principio no son voluntarios, dado que su única funcionalidad es la de centrar la imagen sobre la fóvea, aunque se pueden llegar a controlar.

1.4.5. Vergencias

Son movimientos que realizan ambos ojos aunque en direcciones opuestas, su objetivo pues, es conseguir que aparezca la misma imagen en la retina de ambos ojos, variando la distancia de enfoque. Existen dos tipos: convergencia y divergencia. La convergencia se produce cuando los ojos giran en dirección a la nariz, para enfocar un objeto que se acerca al sujeto. La divergencia es el movimiento contrario,

provocado cuando el objeto se aleja del sujeto. El ojo puede llegar a moverse a 10°/seg, alcanzando una amplitud de 15°.

1.4.6. Vestibulares

Son movimientos involuntarios que se producen cuando la cabeza o el tronco del sujeto se mueven, consiguiendo manter así el objeto en la retina. Este reflejo se conoce con el nombre de Reflejo Vestíbulo Ocular (VOR).

1.4.7. Nistagmos

Es un movimiento ocular coordinado y caracterizado por una oscilación rítmica de los ojos. Se han descubierto dos fases, una lenta (de ida) y otra rápida (de retorno). En la fase lenta, los ojos se mueven para que la retina pueda focalizar un objeto (similar a las persecuciones lentas) mientras que en la fase rápida, el ojo retorna de forma similar a como lo haría un sacádico.

Capítulo 2

Estado del Arte de los Eye Trackers

El dispositivo utilizado para medir los movimientos de los ojos es llamado eye tracker (Seguidor de ojos). Se pueden distinguir dos tipos de técnicas para monitorizar el movimiento: aquellas que miden la posición del ojo respecto a la cabeza y aquellas que miden la orientación del ojo en el espacio, también conocido como point of regard o punto de mirada.

Existen cuatro técnicas básicas de medición de movimiento ocular: electrooculografía (EOG), lentes de contacto, foto o video oculografía y detección por video basada en la pupila y la reflexión córnea.

2.1. Técnicas de Seguimiento de mirada

2.1.1. Electro-oculografía

La electro-oculografía (EOG) fue la técnica de detección de movimientos oculares más popular hace cuarenta años. Está basada en la medida de las diferencias de potencial de la piel que rodea la cavidad ocular. El sujeto ha de llevar unos electrodos sobre la piel que rodea el ojo. En la figura 2.1 se puede apreciar un sujeto con un medidor EOG. Mide rangos de 15-200 μV. Esta técnica mide la posición relativa del ojo respecto a la cabeza, y por lo tanto no vale para averiguar la posición a la que el ojo está mirando, a no ser, que se mida también el movimiento de la cabeza.

2.1.2. Lentes de Contacto

Una de las formas más exactas de medir el movimiento del ojo consiste en una lente de contacto con un pequeño hilo de metal en ella, que se apoya directamente

Figura 2.1: Medición del movimiento mediante electro-oculografía.

sobre el ojo. Las lentes de contactos son grandes, extendiéndose sobre la córnea y la esclerótica. El hilo está arrollado en una vuelta y se encarga de medir las variaciones de campo magnético. A pesar de ser el método mas preciso (es capaz de medir variaciones de 5-10 arcosegundos) es también el método más intrusivo. La inserción de las lentes requiere práctica y mucho cuidado, y el hecho de llevar las lentes implica una incomodidad en los ojos. Este método mide la posición de los ojos respecto a la cabeza, y tampoco es muy útil para medir el punto de interés del ojo.

2.1.3. Foto-oculografía o Video-oculografía

Existe una gran variedad de técnicas bajo esta denominación que se encargan de extraer características de los ojos bajo la rotación y traslación, como la forma de la pupila, la posición del limbo (la unión entre el iris y la esclerótica), las reflexiones córneas, etc. Aunque diferentes, ninguna de estas técnicas proporciona una medición del punto de interés.

2.1.4. Basados en Video con Detección de Pupila y Reflexión córnea

Las técnicas anteriores son válidas para medir el movimiento de los ojos, pero no valen para medir el punto de interés, es decir el lugar exacto a donde el ojo está mirando. Para poder medir esto, se necesita que la cabeza se encuentre quieta, para que la posición del ojo coincida con el punto de interés. Este tipo de trackers basados en video pueden ser de dos tipos, aquellos en los que la cabeza se apoya y se mantiene fija, o aquellos que se llevan integrados en la propia cabeza. Las refle-

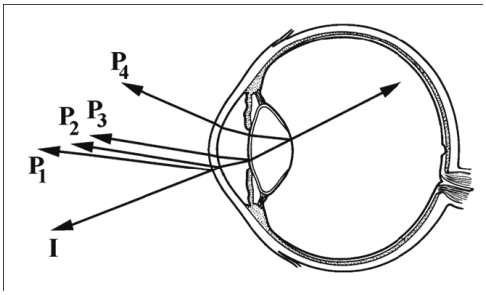


Figura 2.2: Reflexiones purkinje. Duchowski (2003) Eye Tracking Methodology: Theory and Practice

xiones córneas son capturadas a la vez que el centro de la pupila, estas reflexiones son conocidas con el nombre de reflexiones de Purkinje o imágenes de Purkinje. Estos trackers capturan la primera imagen de Purkinje (el mayor de estos brillos) y utilizan el vector formado por los puntos centro de la pupila y centro del brillo como valor de posición de la pupila para el tracker. Se utiliza ese vector y no la posición absoluta de la pupila como valor porque es más invariante a movimientos accidentales del dispositivo y a movimientos de cabeza, que la posición absoluta de la pupila. Con un proceso de calibración previo, estos trackers son capaces de medir el punto de interés.

Existen dos tipos de trackers basados en luz infrarroja: los de pupila oscura y los de pupila brillante. Cuando el haz de luz que ilumina el ojo es coaxial al camino de visión, el ojo actúa como retroreflector, la luz se refleja en la retina y se produce un efecto similar al de los ojos rojos en las fotografías, produciendo una pupila brillante en la captura de la misma. Si existe un desplazamiento entre la luz y el camino de visión, la pupila será capturada con un color muy oscuro: pupila oscura.

2.2. Eye Trackers Comerciales

Los sistemas de seguimiento de mirada han alcanzado una popularidad muy grande en los últimos años dado que tienen aplicación en diversos campos: psicología, psicolinguística, interacción hombre máquina, márketing... Debido a ello han surgido aplicaciones comerciales que tratan de proporcionar productos de alto nivel para satisfacer la demanda actual. Existen principalmente dos sistemas de seguimiento de mirada que han de ser tomados como referentes en el estado actual de los mismos:

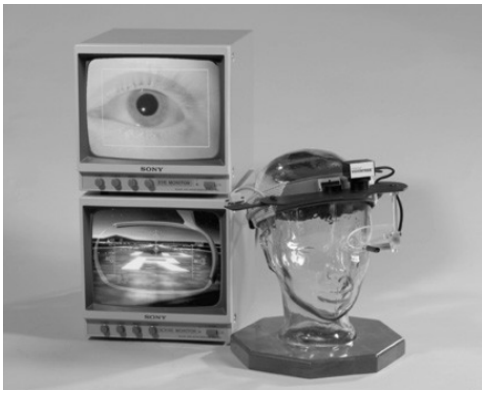


Figura 2.3: VisionTrak© de Polhemus

2.2.1. VisionTrak© de Polhemus

Polhemus es una compañía dedicada al posicionamiento tridimensional, captura de movimiento, escáneres 3D... Productos como Patriot©, Liberty© y Fastrak© sitúan a Polhemus como líderes en el posicionamiento 3D.

Uno de sus productos, llamado VisionTrak©, se encarga de realizar seguimientos de ojos y de objetivos, permitiendo para ello plena libertad de movimientos con la cabeza. Obtiene gran cantidad de datos como número de fijaciones, dilatación de la pupila, proporción de parpadeos, aceleración y velocidad de los ojos.

Utiliza la reflexión de la pupila y la córnea con una cámara infrarroja y otra visible, y utiliza un polhemus como posicionador tridimensional, lo cual lo convierte en un dispositivo con una amplia seguridad en detección. Su versión monocular oscula entre 12000 y 13000 euros, y la binocular ronda los 20000.

2.2.2. VRET de la Universidad de Clemson

El Laboratorio de Realidad Virtual de la Universidad de Clemson diseñó un dispositivo llamado VRET, capaz de funcionar con gran precisión en situaciones con iluminaciones extremas. Utiliza dos o tres cámaras en función de si procesa un ojo o los dos y un sistema de posicionamiento tridimensional llamado Flock of Birds©.

Ambos dispositivos son similares en prestaciones y en precio. Presentan versiones monoculares y binoculares y utilizan sistemas de posicionamiento tridimensional.

Parte II

Desarrollo

Capítulo 3

Fases de desarrollo del proyecto

3.1. Definición del Problema

Es innegable la importancia de los ordenadores en la época actual. Sin embargo, los dispositivos utilizados para interactuar con las computadoras, los periféricos, están diseñados para personas que no tienen ninguna discapacidad, por lo tanto limitan el acceso a las mismas a las personas que sí las tienen. Los dispositivos de entrada de información más comunes, el ratón y el teclado están diseñados para personas que no tengan problemas físicos en las extremidades superiores.

Se desea eliminar, al menos en parte, esta limitación, creando un sistema de interacción hombre-máquina que permita a discapacitados utilizar el ordenador aunque sea de una forma reducida. También se debe crear un primer prototipo de un futuro sistema multimodal.

3.1.1. Objetivos del Sistema

Existen una serie de objetivos que el sistema diseñado ha de tratar de cumplir:

- Construir un sistema con un bajo coste de hardware. Esto permitirá extender su implantación sin imponer limitaciones de presupuesto.
- El hardware utilizado debe de ser lo menos intrusivo posible. El uso del dispositivo no debe ser desagradable de cara al usuario.
- El sistema ha de proporcionar la mayor libertad de movimiento posible al usuario. No se deben imponer restricciones de movilidad.
- Se deben utilizar pocos recursos computacionales. El sistema ha de ser ligero, esto es, que permita la ejecución de otros programas sin apoderarse de todos los recursos.

- Una sola persona debe poder hacer un uso completo del sistema. De esta forma se proporciona una mayor independencia al usuario.
- El sistema debe ser extensible, esto es, debe permitir futuras incorporaciones de nuevos subsistemas para convertirlo en un sistema multimodal.
- El sistema ha de poder ser utilizado por el mayor número de personas posibles.

3.2. Estudio de Viabilidad

Se han realizado exhaustivas búsquedas de dispositivos periféricos con capacidades similares. Se han analizado las características de cada dispositivo, sus ventajas, sus inconvenientes, su precio y las ventajas que proporcionan respecto a los dispositivos periféricos actuales. Los dispositivos periféricos de entrada de datos más comunes son:

Teclado El teclado es el dispositivo más común de entrada de datos. Está formado por un conjunto rectangular de teclas con caracteres impresos sobre ellas (imagen 3.1). Para producir determinados símbolos es necesario la pulsación simultánea de varias teclas. Aproximadamente la mitad de las teclas producen símbolos, mientras que la otra mitad genera órdenes. Para utilizarlo con comodidad es imprescindible no tener discapacidades físicas en las extremidades superiores.

Figura 3.1: Teclado ajustable de Apple

Ratón Es un dispositivo apuntador diseñado para ser utilizado con una mano (imagen 3.2). Se mueve en una superficie bidimensional detectando el movimiento, contiene botones e incluso una rueda que permiten realizar acciones al usuario. Existen varios tipos de ratones en función del mecanismo interno: mecánicos, ópticos e inerciales pero todos están diseñados para humanos sin discapacidad en al menos una mano.

Figura 3.2: Ratón Apple

Joystick Consiste en un stick o "palo" situado sobre una base con botones, que tiene botones y se mueve en dos o tres dimensiones. Son utilizados sobre todo en juegos, e ideales para simuladores de vuelo y control de máquinas. Para su uso se necesitan un par de manos humanas funcionales.

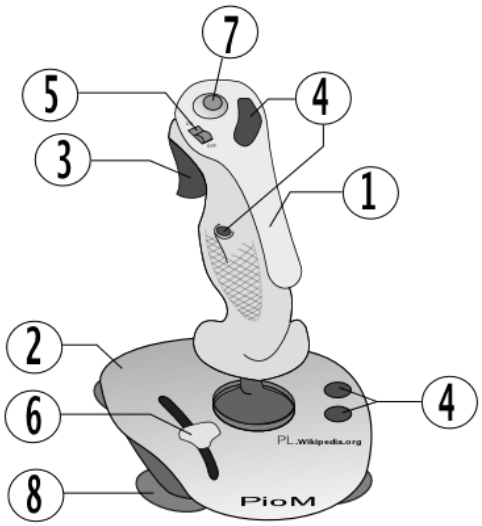


Figura 3.3: Joystick: 1 stick, 2 Base, 3 Gatillo, 4 botones extra, 5 conmutador de disparo automático, 6 Acelerador, 7 Hat switch, 8 Suction Cup

Trackball Es un dispositivo apuntador de forma similar (imagen 3.4) al mouse pero se utiliza moviendo una bola situada en la parte superior del mismo. Es ideal para lugares con carencia de espacio para realizar movimientos con un ratón. Aún así sigue imponiendo restricciones para discapacitados al ser necesaria una mano para moverlo.

Figura 3.4: Trackball de Logitech

Pantalla Táctil Son monitores o pantallas sensibles a la presión, utilizados como dispositivos de entrada de datos. Su uso puede eliminar la utilización de dispositivos apuntadores, muy útil en terminales dispuestos en lugares públicos. Nuevas videoconsolas como la Nintendo DS utilizan las pantallas táctiles como dispositivo de control primario. Sin embargo utilizarlos supone en la mayor parte de los casos realizar mayor movimiento que utilizar un ratón, limitando su uso.

Figura 3.5: Nintendo DS

Tableta Digitalizadora Es un dispositivo que permite introducir imágenes dibujadas a mano y gráficos en los ordenadores de forma similar a como se hace con un lápiz y un papel. Consiste en una superficie plana donde el usuario

puede “dibujar” con un pequeño aparato llamado stylus.

Figura 3.6: Tableta Digitalizadora

Para poder utilizar cualquiera de los dispositivos anteriormente listados, se necesita no tener ninguna discapacidad física en las extremidades superiores. Esto impide el uso de los ordenadores a este tipo de discapacitados, o lo permite no sin grandes dosis de dolor al intentarse éstos adaptar a los requerimientos hardware.

Existen otros sistemas de entrada de datos sin necesidad de mover las extremidades:

- Reconocimiento del habla. El usuario interactúa con el ordenador a través de comandos hablados (ej. “llama a casa” o “consulta mail”). Para ello es necesario un micrófono y software de reconocimiento de voz. El software utiliza Modelos Ocultos de Markov y Redes Neuronales para realizar el reconocimiento.
- Dispositivos visuales. Utilizando microcámaras y algún software de visión artificial se puede conseguir interactuar con el ordenador sin necesidad de contacto con ningún periférico.

El sistema a implementar que debe formar parte de un futuro sistema multimodal ha de estar constituido por un dispositivo visual, al que en un futuro se le añada capacidades de reconocimiento del habla.

Existen multitud de dispositivos que implementan técnicas de visión artificial para realizar reconocimiento de órdenas usadas para interactuar con las máquinas. Como ejemplo se pueden citar los siguientes:

- EyeToy es un dispositivo para la PlayStation 2 parecido a una webcam. Se encarga de procesar las imágenes obtenidas de la cámara permitiéndo así a los usuarios interactuar con los juegos a través del movimiento.

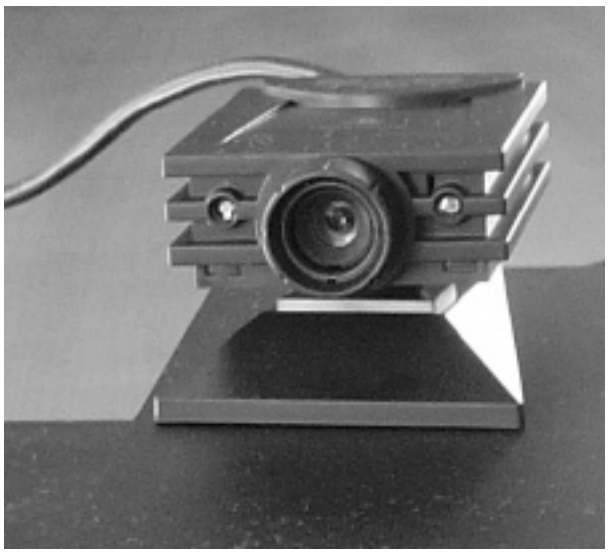


Figura 3.7: EyeToy de Sony PlayStation

- GesturePad de FingerWorks: [4] permite convertir gestos en comandos. Está hecho para ser una alternativa al ratón. Para abrir un fichero, el usuario tiene que rotar su mano como si estuviese abriendo un bote, para cerrarlo basta con el gesto contrario. Está formado por una especie de tableta digitalizadora que capta el roce con los dedos y convierte los gestos en eventos de ratón.

- EagleEyes: es una iniciativa de la universidad de Boston cuyo objetivo es permitir el acceso a los ordenadores a personas discapacitadas. Tienen varios sistemas desarrollados: EagleEyes que es un sistema electro-oculográfico para controlar el ordenador con solo los ojos y CameraMouse para controlarlo moviendo la cabeza. Este último está formado por una cámara web conectada al ordenador.

- EyeTrackers (Seguimiento de ojos): se agrupan aquí un conjunto de técnicas que utilizan para la interacción el ojo humano. Existen varios modelos: con el movimiento restringido de la cabeza, sin restricción, EOG, lentes de contacto y basados en cámaras.

Existen muchos más dispositivos de entrada que utilizen cámaras y visión artificial, pero muchos siguen requiriendo que el usuario realice grandes movimientos para poder controlar el ordenador. El uso de movimientos limita su utilización por lo

tanto la técnica que permite el uso del ordenador a mayor número de usuarios debe ser una basada en un eyetracker. La técnica oculográfica y la basada en lentes de contacto son demasiado intrusivas, lo cual no encaja con los objetivos del proyecto.

3.2.1. Modelo Propuesto

Los periféricos tradicionales solo pueden ser utilizados por personas sin discapacidad en las extremidades superiores. Los dispositivos visuales pueden permitir el uso de los ordenadores por discapacitados, pero siempre y cuando no necesiten grandes movimientos por parte del usuario (ej. EyeToy). El movimiento más sencillo es el movimiento del ojo, por lo tantoy según todo lo visto anteriormente (capítulo 2), la técnica que mejor encaja es un eye tracker basado en vídeo:

- Puede ser utilizado con los ojos e incluso con un solo ojo.
- No existe técnica menos intrusiva que la utilización de cámaras, ya que no requieren contacto ni manipulación por parte del usuario
- Si la cámara está situada sobre el propio usuario, el movimiento estará permitido.
- No se necesita un hardware caro (Polhemus o Flock of Birds) simplemente un par de cámaras y tarjetas capturadoras.
- Permite la incorporación futura de micrófonos o cualquier otro dispositivo que unido al actual permita interacción multimodal.

Según la bibliografía consultada ([23] [22] [20] [19] [31]) el mejor sistema eyetracker visual es aquel formado por dos cámaras, una que apunta al ojo del usuario y otra que apunta hacia lo que está situado enfrente del usuario.

Por lo tanto se van a utilizar dos minicámaras (cámaras espía) montadas sobre un dispositivo preparado para llevar puesto en la cabeza. Una cámara captará longitudes de onda infrarrojas y estará apuntando a la cabeza del usuario, dispuesta a capturar las posiciones en las que se encuentra su pupila, mientras que otra cámara centrará su atención en lo que el usuario tiene frente a sí. Se prefiere utilizar el espectro infrarrojo para capturar posiciones de pupila por los siguientes motivos:

- Es invariante a condiciones de luz, puede trabajar con luces apagadas, encendidas incluso en exteriores con iluminación natural.
- Al capturar el espectro infrarrojo, las reflexiones producidas en el ojo humano no se tienen en cuenta, esto es importante porque la pupila vista de cerca puede llegar a comportarse como un espejo.

El software será desarrollado acorde con los requisitos del hardware. Se realizará un proceso continuo de captura de cada una de las cámaras. Por un lado se realizará la detección de la pupila y el cálculo de su centro, y por otro se detectarán aspectos del mundo. La solución para encontrar una correspondencia entre posiciones de pupila y lugares del mundo (visibles mediante la cámara que apunta hacia delante) es realizar una calibración. En la bibliografía consultada la calibración es realizada por dos usuarios, uno se lleva el sistema en su cabeza y mira a un lugar donde haya una serie de puntos y otro se encargará de utilizar el ordenador que realiza las capturas.

Se desarrollarán dos aplicaciones, una encargada de realizar la calibración, que pueda ser utilizada por un solo usuario, y otra que se encargue de utilizar la información de calibración para hacer algo utilizable por el usuario. Esta última estará formada por un teclado virtual mostrado en el monitor que permita al usuario escribir con el ojo y unos puntos de detección, para que el sistema sepa el lugar exacto al que está mirando.

3.2.2. Coste del Sistema

Componente	Unidades	Precio Unidad	Precio
Tarjeta Capturadora	2	53 €	106 €
Microcámara	2	99 €	198 €
Precio Total			304 €

3.2.3. Estudio de Viabilidad

3.2.3.1. Viabilidad Técnica

El sistema propuesto se enmarca dentro de un nuevo área de investigación llamada HCI que se encarga de estudiar y mejorar la interacción entre humanos y máquinas. Utiliza técnicas novedosas basadas en vídeo e infrarrojos situadas en un campo aún por explorar. El proyecto por tanto tiene un elevado componente de investigación y la viabilidad técnica es impredecible con exactitud. Podemos suponer que realizando una buena detección del ojo, una buena calibración y una buena detección del teclado en pantalla se va a predecir al menos de forma aproximada la posición a la que el usuario está mirando, pero se desconoce por completo la exactitud de tal predicción.

3.2.3.2. Viabilidad Operacional

El uso del sistema propuesto no supone ningun tipo de problema dado que lo único que hay que hacer es mirar a un lugar. El único punto donde puede encontrarse algo

de dificultad es en el proceso de calibración, que puede evitarse leyendo el manual de usuario y con un breve entrenamiento.

3.2.3.3. Viabilidad Económica

El único coste del sistema es el del hardware pero es inevitable. En su desarrollo se han buscado los elementos más baratos que proporcionan las funciones mínimas necesarias. El software desarrollado está licenciado como GPL y distribuido sin coste alguno al igual que el sistema operativo y las librerías necesarias para utilizarlo.

3.3. Análisis del sistema

3.3.1. Diagrama de Contexto del Sistema

3.3.2. DFD's del Sistema

En esta sección se muestran los Diagramas de Flujo de Datos del Sistema. Cada uno de los tres diagramas se corresponde con cada una de las tres aplicaciones del sistema.

Diagrama de Flujo de Datos de itune, aplicación utilizada para calibrar el sistema:

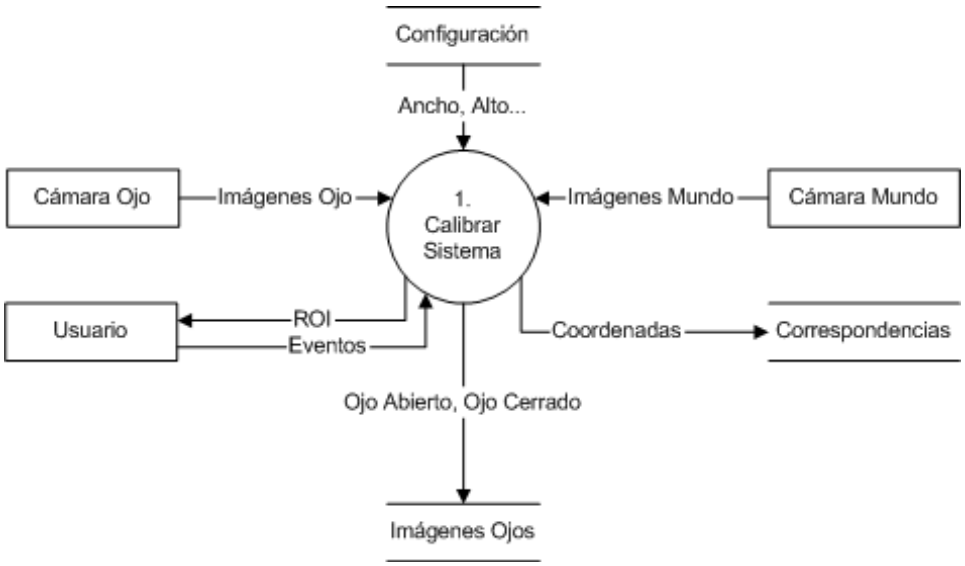


Diagrama de Flujo de Datos de eyeboard, aplicación utilizada para escribir con un solo ojo:

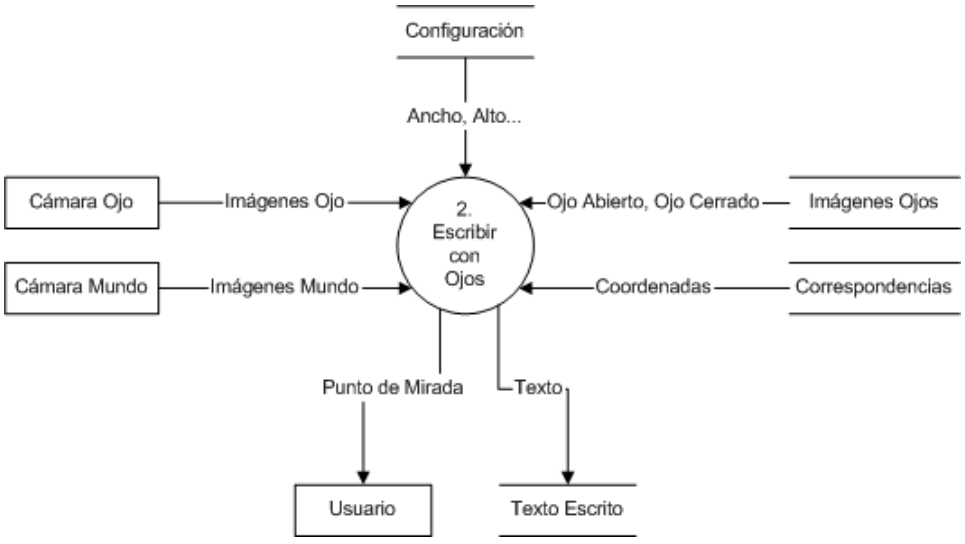
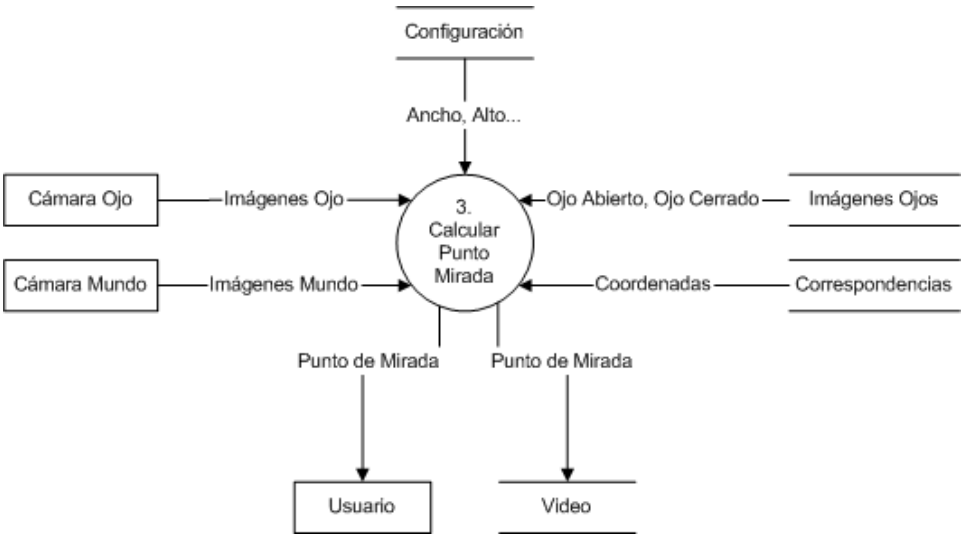


Diagrama de Flujo de Datos de infrared, aplicación para calcular el punto de mirada:



3.3.3. Descripción de los Componentes del DFD del Sistema

3.3.3.1. Entidades Externas

Usuario Representa al usuario que interactúa con el sistema indicando las operaciones que desea realizar.

3.3.3.2. Almacenes

Configuración Fichero que contiene parámetros de configuración de las aplicaciones del sistema: ancho y alto de las imágenes a capturar y procesar.

Correspondencias Fichero que contiene parejas de coordenadas reales bidimensionales. Una indica la posición de la pupila y otra su correspondencia en coordenadas del mundo.

Imágenes Ojos Dos ficheros que contienen imágenes raw de un ojo abierto y un ojo cerrado.

Texto Escrito Contiene el texto que el usuario escribe con la mirada.

Vídeo Contiene una secuencia de vídeo con las imágenes del ojo y del mundo que muestra el punto de interés.

3.3.3.3. Procesos

Calibrar Sistema Proceso que se encarga de realizar la calibración, es decir, buscar correspondencias (parejas) de puntos de posiciones de pupila y posiciones de mundo, almacenándolas en fichero. También guarda una imagen con

el ojo abierto y otra con el ojo cerrado.
Flujos de entrada:

- Ancho, Alto... representa los datos de configuración de la aplicación
- Imágenes Ojo: representa el tren de imágenes proveniente de la cámara del ojo.
- Imágenes Mundo: representa el tren de imágenes proveniente de la cámara del ojo.
- Eventos: corresponde a las órdenes dadas por el usuario para calibrar el sistema.

Flujos de salida:

- ROI: Region of Interest contiene el área de interes, es decir, el área donde se encuentra el ojo del usuario en la imagen de la cámara del ojo.
- Coordenadas: contiene parejas de coordenadas de posiciones de pupila y posiciones de mundo.
- Ojo Abierto, Ojo Cerrado: son dos imágenes con el ojo abierto y cerrado respectivamente que se almacenan en disco para poder distinguir si el ojo está abierto o cerrado.

Escribir con Ojos A partir de las correspondencias de puntos, y de las imágenes de las cámaras, averigua la posición del teclado mostrado en pantalla donde el usuario está mirando. En función de eso muestra un texto por pantalla y lo almacena en disco.
Flujos de entrada:

- Ancho, Alto... representa los datos de configuración de la aplicación
- Imágenes Ojo: representa el tren de imágenes proveniente de la cámara del ojo.
- Imágenes Mundo: representa el tren de imágenes proveniente de la cámara del ojo.
- Ojo Abierto, Ojo Cerrado: son dos imágenes con el ojo abierto y cerrado respectivamente utilizadas para saber si el ojo está abierto o cerrado.
- Coordenadas: contiene parejas de coordenadas de posiciones de pupila y posiciones de mundo.

Flujos de salida:

- Punto de mirada: se muestra al usuario en pantalla el lugar exacto al que está mirando.
- Texto: el texto que el usuario escribe se muestra en pantalla y se escribe en disco.

Calcular Punto Mirada A partir de las correspondencias de puntos, y de las imágenes de las cámaras, averigua la posición del mundo donde el usuario está mirando. El resultado lo almacena en disco en formato de video. Flujos de entrada:

- Ancho, Alto... representa los datos de configuración de la aplicación
- Imágenes Ojo: representa el tren de imágenes proveniente de la cámara del ojo.
- Imágenes Mundo: representa el tren de imágenes proveniente de la cámara del ojo.
- Ojo Abierto, Ojo Cerrado: son dos imágenes con el ojo abierto y cerrado respectivamente utilizadas para saber si el ojo está abierto o cerrado.
- Coordenadas: contiene parejas de coordenadas de posiciones de pupila y posiciones de mundo.

Flujos de salida:

- Punto de mirada: se muestra al usuario en pantalla el lugar exacto al que está mirando y también se escribe en disco.

Capítulo 4

Desarrollo Experimental

Debido a la necesidad continua de pruebas y al ciclo continuo de programación-experimentación en el desarrollo del presente proyecto, se hace nesearia la creación de un nuevo capítulo en el que se detallen las pruebas y experimentos realizados tanto a nivel hardware como a nivel software.

Este desarrollo de prueba y error centrado sobre todo en la algorítmica hace que el proyecto difiera en gran medida del resto de proyectos software en los que el diseño está presente desde las primeras fases del desarrollo.

4.1. Desarrollo Experimental del Hardware

El modelo de hardware final, "el casco", no ha sido el mismo durante todo el desarrollo. En una primera fase sólo se utilizaba el soporte de color verde (Ver 4.1) con una sola cámara apuntando al ojo. Más tarde se añadió un filtro de infrarrojos para esa cámara y se incorporó la segunda cámara. Después se sustituyeron las baterías por un transformador y por último se realizó un pequeño ajuste en la posición de la cámara del mundo.

4.1.1. Cambio de Filtros

En las primeras pruebas del desarrollo, y debido a la tardanza en la entrega de los materiales requeridos, se utilizaron filtros para la cámara de infrarrojos provenientes de carretes velados de cámaras de fotos (Ver 4.2).

Anque estos filtros proporcionan una imagen con una calidad bastante aceptable, no están tan optimizados como los filtros Lee utilizados (Ver imagen 4.4), por ello se sustituyeron. Los nuevos filtros utilizados son unos *Lee Camera Filters nº 87 100mm x 100mm, 730 nanómetros*.

Figura 4.1: Soporte para las cámaras

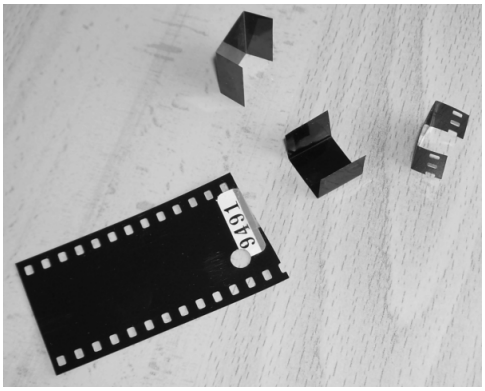


Figura 4.2: Filtro de infrarrojos obtenido de un carrete

Figura 4.3: Filtro de infrarrojos Lee

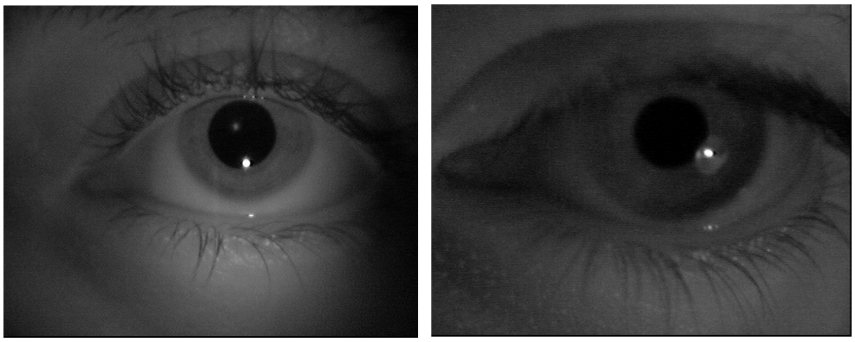


Figura 4.4: Diferencia de calidad en los filtros: a) Filtros Lee. b) Filtro de carrete

4.1.2. Sustitución de Baterías por Transformador

El LED que ilumina con longitudes de onda infrarrojas recibe una corriente de 9V, para alimentarlo al principio se utilizaban baterías de 9V. Debido que el peso

Figura 4.5: Baterías

de la batería incomodaba al usuario en exceso, y a que el consumo del LED era excesivo (era necesario cambiar la pila en cada sesión), se sustituyó la batería por un transformador y se añadió el circuito de la figura 4.6.

4.1.3. Adición de un Nuevo LED

Un solo LED proporcionaba demasiada poca iluminación en el ojo, y la corrección por software incrementaba el ruido en la imagen, debido a ello se añadió un nuevo LED. De esta forma un LED ilumina la parte derecha del ojo y otro la izquierda consiguiendo con eso una iluminación adecuada para el proceso.

Figura 4.6: Transformador y circuito de alimentacion de LEDS utilizados

4.1.4. Cambio de la Posición de la Cámara del Mundo

Debido a que el ojo y la cámara del mundo se encuentra situadas en puntos distintos del espacio, se producen desviaciones angulares en las posiciones aparentes de los objetos. Este efecto se conoce como paralaje, y produce una variación en la posición de los objetos distinta en función de la distancia al mismo.

4.2. Desarrollo Experimental del Software

4.2.1. Elección del Algoritmo de Detección de Pupila

4.2.1.1. Introducción

La pupila del usuario representa el punto de mayor interés de la aplicación desde el punto de vista del programador y del investigador:

- Para el programador representa un reto el hecho de realizar una detección en tiempo real de un objeto que está continuamente en movimiento a una velocidad lo suficientemente elevada como para no poder realizar ninguna predicción de movimiento, y que puede estar parcialmente ocluído por reflexiones córneas o por párpados y que llega a desaparecer cada vez que se cierra el ojo.

- Para el investigador resulta atractivo comprobar el comportamiento del ojo en respuesta a estímulos o acciones que realiza el sujeto.

De su correcta detección depende la exactitud del resto de métodos y técnicas utilizadas. Por ello es esencial que el algoritmo de detección de la pupila sea robusto y preciso, a la par que veloz.

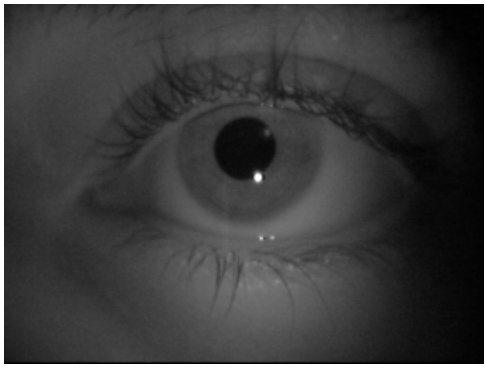


Figura 4.7: Típica imagen de un ojo iluminado en infrarrojos

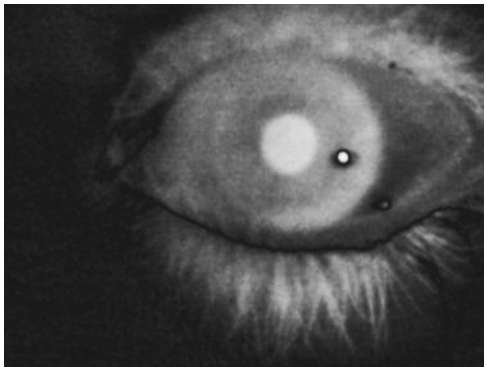


Figura 4.8: Diferencia entre una imagen con el ojo abierto y otra con el ojo cerrado

Para la elección de un algoritmo óptimo se han desarrollado varias alternativas posibles, cada una de ellas ha sido testeada en la verdadera aplicación, y se han anotado sus ventajas e inconvenientes

4.2.1.2. Primer método implementado

Teniendo en cuenta que una pupila iluminada con infrarrojos, posee un nivel de gris mucho menor (más oscuro) que el resto del ojo (Ver figura 4.7), el método se basa en recorrer la imagen y elegir aquél con menor nivel de gris, que se supone que pertenece al interior de la pupila. Una vez detectado el punto interior, se realiza un algoritmo de crecimiento y se elige aquel punto que corresponde con el centro de gravedad de la masa de puntos formada por los puntos recorridos por el algoritmo de crecimiento. Para encontrar el punto más oscuro se han realizado varias aproximaciones:

- Primera aproximación: Búsqueda del punto más oscuro. Como primera aproximación, realizamos un recorrido de toda la imagen, para buscar el punto más oscuro. Éste método tiene varios problemas: en primer lugar el número de puntos recorridos en el algoritmo es $width * height$ lo cual implica una gran carga computacional, en segundo lugar y más importante que el anterior es que el punto más oscuro de la imagen puede estar situado en los bordes de la imagen, debido a los factores de iluminación. Por lo tanto este método queda descartado.

- Segunda aproximación: Eliminación de bordes. Para solucionar los dos problemas anteriores se propone eliminar la búsqueda del punto mínimo en las zonas cercanas al borde de la imagen, de esta forma se reduce el área de búsqueda y se eliminan los puntos oscuros de los bordes. Aunque mejor que la anterior, esta aproximación tiene un fallo: se está presuponiendo que la pupila nunca va a estar en un borde, y la eliminación de los bordes ha de ser realizada "a ojo".

- Tercera aproximación: Búsqueda exacta. Es similar a las búsquedas anteriores, pero en éste caso se busca sólo en la zona en la que se encuentra el ojo. Para averiguar la posición del ojo se realizan dos capturas previas a la calibración del sistema, una de ellas con el ojo abierto y otra con el ojo cerrado. De esta forma, y teniendo en cuenta la diferencia (ver imagen 4.8)entre las dos imágenes se deduce la posición en la que se encuentra el movimiento: la posición exacta del ojo.

El algoritmo se comporta muy bien aunque presenta unos leves problemas:

- Cuando un brillo de la córnea se situa sobre la pupila, el algoritmo recursivo no recorre el brillo, por lo tanto los puntos pertenecientes al brillo, no entrarán a formar parte de la masa de puntos sobre la que se calcula el centro de gravedad, lo cual implica que el centro de gravedad (centro de la pupila) se ve desplazado ligeramente.

- El algoritmo recursivo no recorre el párpado cuando éste oculta parcialmente la pupila, por lo tanto el centro de gravedad se desplaza ligeramente. Sin embargo hay que tener en cuenta que este caso es extremo, que implica que el ojo está apunto de ser cerrado, o que acaba de abrirse. Este caso se da muy pocas veces, y puede ser solucionado considerando que el ojo está cerrado.

4.2.1.3. Segundo método implementado: Transformada de Hough para círculos

Introducción La transformada de Hough es una técnica de extracción de características utilizada en visión artificial. La transformada clásica de Hough se utiliza

para extraer rectas de una imagen. Está basada en el hecho de que por un determinado punto, pueden pasar infinitas rectas con distinta pendiente. El objetivo de la transformada es determinar qué lineas pasan por más puntos. Para ello se realiza un cambio de coordenadas, entre las coordenadas de los puntos candidatos (x, y) y un espacio bidimensional llamado acumulador (r, θ). Para cada punto candidato se realiza la transformada, se "dibuja" una linea senoidal en el acumulador y por último se extraen aquellos puntos del acumulador que cumplen las características deseadas (orientacion y posicion).

Esta transformada se puede generalizar para extraer otro tipo de formas, como por ejemplo un círculo, aunque en este caso la matriz de acumulación sería tridimensional, para poder representar los tres parámetros que definen un círculo: coordenadas x e y del centro y el radio.

La librería OpenCV [9] posee una implementación de la transformada llamada cvHoughCircles, que permite realizar extracciones de círculos (circunferencias) de forma cómoda y rápida, ya que OpenCV está basada en la librería IPP [6].

Ajuste de Parámetros de la Transformada Para comprobar la eficiencia de este algoritmo, se han realizado una serie de pruebas sobre un conjunto de imágenes extremas, es decir, comprenden todos los casos: imágenes con pupilas normales, imágenes con pupilas parcialmente ocluidas por el párpado, imágenes con brillos de la córnea sobre la pupila e imágenes con la pupila muy ladeada. Para marcar los bordes de la pupila se utilizó el mismo algoritmo recursivo de la primera aproximación. Estos puntos borde son los considerados por el algoritmo para ajustar el círculo.

La función cvHoughCircles se ajusta al siguiente prototipo:

```
CvSeq* cvHoughCircles(CvArr* image,
                      void* circle_storage,
                      int method,
                      double dp,
                      double min_dist,
                      double param1=100,
                      double param2=100);
```

- *image* almacena la imagen que contiene los puntos a los que se tiene que ajustar el círculo.
- *circle_storage* es un buffer necesitado por OpenCV para realizar el algoritmo.
- *method* indica el tipo de transformada que se va a realizar. Actualmente sólo CV_HOUGH_GRADIENT está implementado.

- *dp* Resolución del acumulador.
- *min_dist* Distancia mínima entre centros de circulos.
- *param1* es el primer parámetro del método elegido, en el caso de CV_HOUGH_GRADIENT, representa el umbral máximo para el detector de Canny interno a la función.
- *param2* en el caso de CV_HOUGH_GRADIENT representa el mínimo valor del acumulador que debe tener una celda para ser considerado círculo.

Los parámetros que definen de manera más significativa el comportamiento de la transformada son *dp* y *param2*, para elegir el valor óptimo de estos parámetros, se realizó un pequeño programa que realizaba un bucle entre un rango de valores de estos dos parámetros realizando la transformada en cada iteración. Este programa fue probado con tres imágenes distintas, una pupila normal, una pupila ladeada y una pupila ocluida ligeramente por el párpado.

Ajuste de parámetros de la Transformada de Hough

```
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include <iostream>

using namespace std;

IplImage *img = 0;

int main( int argc, char** argv )
{
for(double dp=1;dp<15;dp+=1.0)
    for(int min=10;min<150;min+=4)
    {
        char* filename=argc==2?
            argv[1]:(char*)"test.png";

        //Se carga la imagen del parametro
        if( (img = cvLoadImage( filename, 1)) == 0 )
            return -1;

        //Se preparan las variables
        IplImage* gray=cvCreateImage(cvGetSize(img),8,1);
        CvMemStorage* storage = cvCreateMemStorage(0);

        cvCvtColor(img, gray, CV_BGR2GRAY);
```

```
        //Se muestra el valor de las variables
        cout<<"dp: "<<dp<<", min: "<<min<<"\n";

        //Se realiza la transformada
        CvSeq* circles=cvHoughCircles(gray,
                        storage,
                        CV_HOUGH_GRADIENT,
                        dp,
                        gray->height/8,
                        300,
                        min);

        //Se muestra cada circulo encontrado
        for(int i = 0; i < circles->total;++i)
        {
            float* p = (float*)cvGetSeqElem(circles,i);
            cvCircle(img,
                cvPoint(cvRound(p[0]),cvRound(p[1])),
                cvRound(p[2]),
                CV_RGB(255,0,0),
                2,
                8,
                0);
        }

        //Se muestra el resultado
        cvNamedWindow("Win", 1);
        cvShowImage("Win", img);

        cvWaitKey(0);

        //Se libera el espacio reservado
        cvReleaseImage(&img);
        cvReleaseImage(&gray);
        cvDestroyWindow("Win");
    }
return 0;
}
```

Para cada una de las imagenes de prueba y para cada uno de los valores del parámetro dp, se anotaron los valores de param2 (umbral mínimo) en los que se empieza a detectar el círculo de la pupila (valor mínimo) y el valor a partir del cual ya no se

detecta (valor máximo). La gráfica (Imagen 4.9) muestra el resultado.

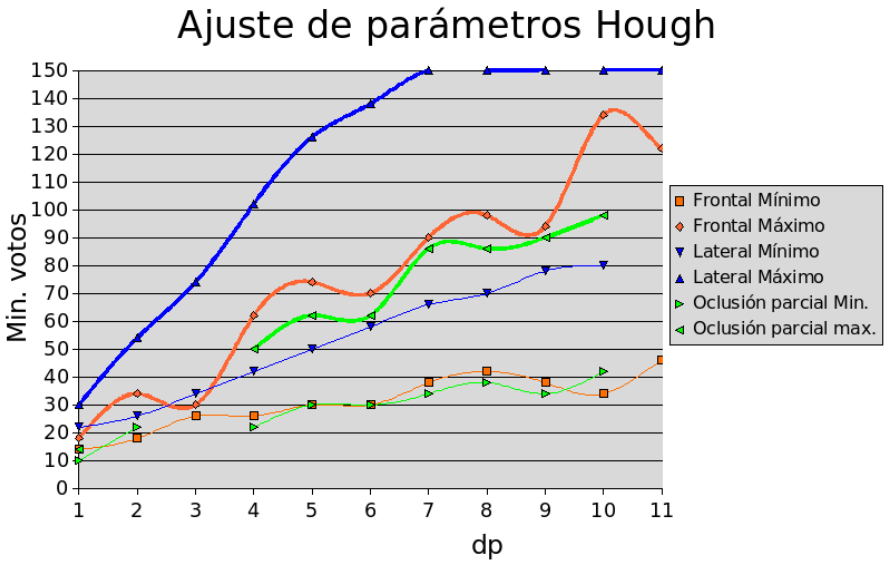


Figura 4.9: Ajuste de parámetros en la Transformada de Hough Circular

Se pueden extraer varias conclusiones de los resultados:

- Cuando la proyección de la pupila se aproxima considerablemente a una forma circular (no se encuentra ladeada), resulta fácilmente identificable, se encuentre parcialmente ocluida o no.
- Cuando la pupila no está ocluida, se obtienen valores más altos en los puntos candidatos del acumulador.
- Cuando la pupila se encuentra ladeada, se necesitan más votos para detectarla.

Como consecuencia de estas pruebas se eligieron los valores de los parámetros que resolvían el problema para todas las imágenes. Para ello se eligió el centro del mayor espacio de detección, de esta forma:

Nombre del Parámetro	Valor óptimo
dp	7.
$param2$	56.

Resultados de las Pruebas Tras el ajuste de parámetros, se integro el método de detección de pupila en el sistema, para observar los resultados en tiempo real.

La transformada de Hough para círculos presento un porcentaje de detección de pupila del 95 % aproximadamente.

Pese a que en un primer momento nos pueda parecer que la detección utilizando Hough es muy robusta pero muy lenta, los resultados muestran lo contrario, el tiempo empleado por la transformada es prácticamente inapreciable para nuestra aplicación, sin embargo la detección del centro deja mucho que desear comparándola con la primera aproximación implementada. El motivo se debe principalmente a que a pesar de que la pupila es circular, está situada sobre una esfera: el ojo. El desplazamiento de la pupila a una posición que no sea perpendicular a la dirección a la cámara provoca una deformación de la proyección (imagen captada por la cámara) del contorno de la pupila, pasando de ser una esfera a una elipsoide (imagen 4.12). Además de ello, la transformada de Hough proporcionada por OpenCV proporciona una resolución demasiado pequeña, es decir, las coordenadas devueltas por el método no trabajan en una resolución de un pixel, sino mayor, lo cual implica que el centro calculado no siempre coincide con el centro real de la pupila.

Debido a estos problemas, la transformada de Hough no será el método empleado en la detección de la pupila.

4.2.1.4. Tercer método implementado: Crecimiento recursivo y morfología binaria

El principal problema que presentaba el primer método era el desplazamiento que sufría el centro calculado cuando se producía un brillo de la córnea dentro de la pupila. Sin embargo este error se puede corregir utilizando morfología binaria de la siguiente forma:

- Crear una imagen nueva de fondo negro.
- Recorrer imagen original, marcando los puntos oscuros (pertenecientes a la pupila) como blancos en la imagen recién creada.
- Realizar una erosión de la nueva imagen de forma que se eliminen los huecos del brillo.
- Realizar una dilatación equivalente a la erosion en la imagen nueva.
- Recorrer la nueva imagen y calcular el centro geométrico de los puntos marcados como interiores a la pupila.

Tras realizar estos pasos se obtiene una figura como la 4.10 en la que se observa la captura de la pupila y el centro geométrico sin realizar operaciones morfológicas, en las dos siguiente imágenes se puede apreciar el proceso morfológico seguido y por último el centro calculado.

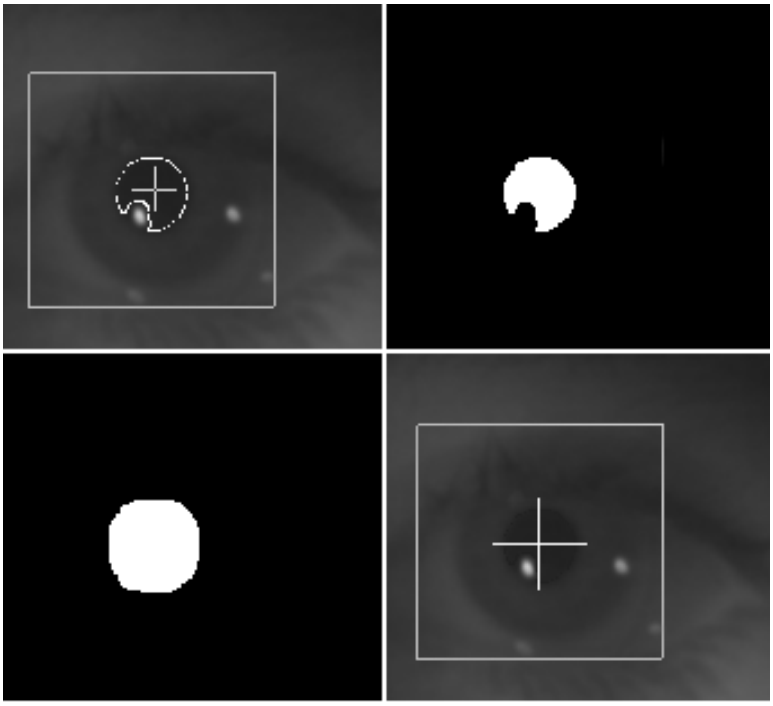


Figura 4.10: Etapas de la captura:
a) Tercera aproximación. b) Se marcan los puntos interiores c) Se realiza morfología binaria. d) Se calcula el centro de gravedad de los puntos blancos.

4.2.1.5. Otras Opciones no Implementadas

Parejas de bordes Existe otro método para averiguar el centro de una circunferencia (o círculo), consiste en detectar el contorno de la pupila por parejas, es decir, un algoritmo recorre una imagen en la que se encuentra una circunferencia desde arriba hasta abajo, línea a línea (líneas horizontales), detectando el límite izquierdo y el derecho del círculo en cada línea. Para cada línea recorrida averigua el punto medio entre los dos bordes obteniendo así una especie de línea vertical. Después realiza este mismo proceso de derecha a izquierda, obteniendo otra línea que corte a la primera (imagen 4.11), consiguiendo averiguar el centro de la circunferencia, que corresponde al centro de la pupila. En condiciones ideales este algoritmo funciona bien, pero fracasaría en el momento en el que alguno de los bordes de la pupila esté ocluido y/o deformado. Esta inperfección el borde se transforma en un desvío de la línea media y el consecuente desvío del centro de la circunferencia. Debido a este fallo, no se ha considerado su utilización.

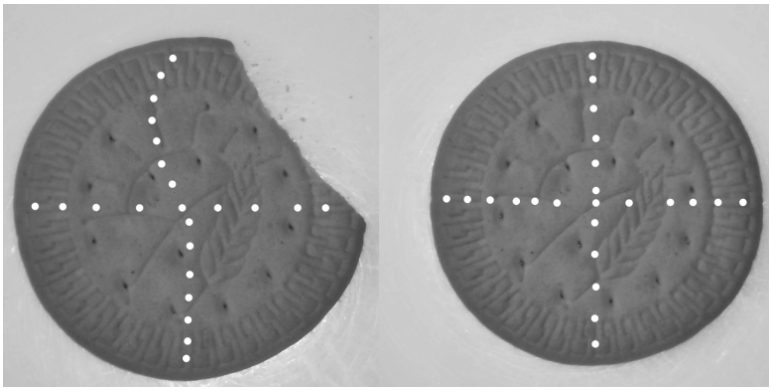


Figura 4.11: Método de las parejas de bordes

Transformada de Hough para elipses La contorno de la pupila tiene forma de circunferencia. Al realizar la proyección sobre la cámara, esta forma se pierde debido a la rotación del ojo y a la curvatura de la córnea (imagen 4.12). El contorno obtenido se asemeja a una elipse, por lo que puede ser aproximado por una transformada de Hough para elipses (EHT). El problema que presenta la transformada de Hough es que pierde eficiencia conforme aumenta la complejidad de la forma buscada. Para realizar una búsqueda de elipses, hay que tener en cuenta al menos 4 parámetros: posición x del centro de la elipse, posición y del centro, relación entre ejes y rotación (aunque se puede representar de otras formas). Debido a esto, la matriz de acumulación tiene que tener 4 dimensiones y por ello la búsqueda de máximos en la misma exige mucho tiempo de proceso. Existen no obstante otros métodos que reducen el número de parámetros para acortar la carga computacional de la transformada, algunos han sido capaz de reducirlos a solamente dos, ver [32]. Una circunferencia es un caso particular de una elipse (con relación 1:1 entre radios mayor y menor), por lo tanto un algoritmo de detección de elipses será capaz de detectar la pupila de manera correcta en cualquier posición.

No se ha realizado la implementación de la transformada de Hough para elipses por ser demasiado costosa computacionalmente y porque el algoritmo elegido se comporta de forma ideal en un tiempo muy probablemente inferior a la EHT.

Detección rápida de círculos utilizando parejas de vectores gradientes: Fast Circle Detection En respuesta a la enorme cantidad de recursos necesitados por los algoritmos basados en Hough, Ali Ajdari *et al.* [15] proponen en un nuevo método para la detección de formas circulares basado en pares de gradientes. Suponiendo que se tiene una superficie circular (imagen 4.13) con una intensidad de color suficientemente diferente al resto de la imagen, se calculan vectores gradientes, y teniendo en cuenta la simetría respecto a un punto (el centro) que presentan los círculos, se agrupan los vectores gradientes en parejas de vectores opuestos, de

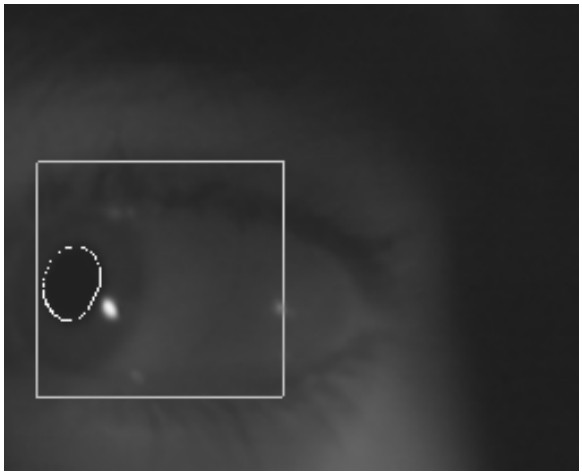


Figura 4.12: La pupila vista desde un lateral presenta un contorno elipsoide no aproximable por Hough circular.

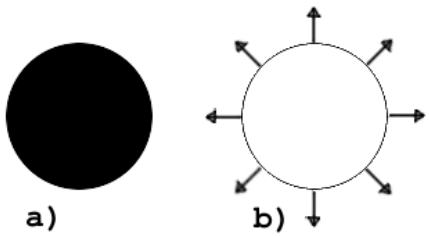


Figura 4.13: a) Ejemplo de círculo. b) Pares de gradientes del círculo

tal forma que cada pareja establece un voto en la zona central del círculo. Más tarde se calcula el centro en función de los votos de las celdas. Este algoritmo es útil también para detectar elipses, dado que las elipses también son simétricas respecto a un punto central.

RANSAC fitting algorithm Random Sample Consensus (RANSAC) es un algoritmo para estimar los parámetros de un determinado modelo matemático. Para ello se basa en un conjunto de datos que contiene outliers y inliners. Los inliers son muestras de datos que realmente se ajustan al modelo matemático elegido, los outliers son datos "infiltrados" o "erróneos" que forman parte de la muestra a la que se debe ajustar el algoritmo.

Un ejemplo básico es el ajuste a una recta real 2D. Un algoritmo simple de mínimos cuadrados se ajusta a la recta, pero produce un mal ajuste a los inliers, porque trata

de ajustarse a *todos* los puntos, incluídos los outliers.

RANSAC soluciona este problema al descartar los outliers, por eso RANSAC sería adecuado para detectar la elipse que forma el contorno de la pupila, descartando los outliers, que en este caso serían puntos que forman parte del conjunto de datos a ajustar, pero que realmente no forman parte del contorno de la pupila, ya sea por un error en los cálculos o por un brillo no controlado de la córnea.

En el caso de detección de pupilas habría que elegir un conjunto de puntos a los que ajustarse. Este conjunto podría estar formado por todos aquellos puntos en los que exista un cambio brusco del nivel de gris. En openEyes ([10]) se utiliza el algoritmo starburst. OpenEyes parte del primer reflejo Purkinje y realiza recorridos radiales en busca de puntos en los que existan cambios de intensidad, y en cada uno de esos cambios vuelve a realizar el proceso. El conjunto de puntos detectados es pasado a un algoritmo de ajuste RANSAC. Se presupone que la mayoría de esos puntos pertenecerán al contorno de la pupila (inliers).

4.2.2. Elección del Modelo Ocular

Entendemos modelo como una idealización de la realidad utilizada para plantear un problema. Una vez asumido un modelo, el resto de las operaciones y procesos tienen que ser realizadas de acuerdo a ese modelo.

Existen varias opciones a la hora de construir un modelo ocular. Un modelo geométrico simple puede asumir que el ojo es una perfecta esfera y la pupila es círculo perfecto impreso sobre la superficie del mismo (Ver imagen 4.14). En él se puede apreciar el primer brillo Purkinje y el centro de la curvatura córnea.

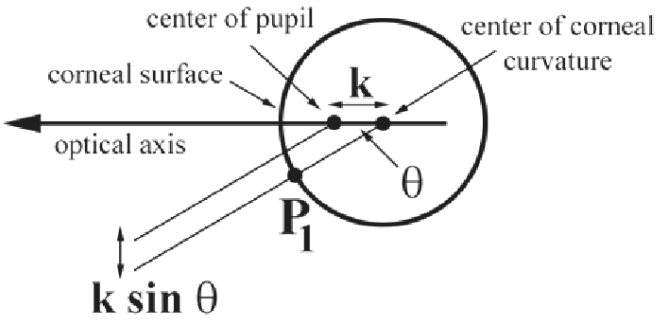


Figura 4.14: Modelo geométrico simple

En función de este modelo, se ha de calcular el centro de la pupila a partir de una imagen 2D y una vez hecho esto realizar los cálculos adecuados para averiguar el ángulo de rotación en el que se encuentra el ojo. Este modelo tiene varios problemas que no se ajustan a la realidad:

- Al igual que las manos de los humanos no son del mismo tamaño y proporciones, el ojo tampoco. Cualquier asunción en los tamaños de las variables del modelo lleva implícito un error de base.
- La proyección de la pupila sobre la cámara no produce un círculo perfecto, sino un elipsoide tanto más deformado cuanto mayor sea el ángulo de rotación del ojo respecto a la cámara que lo grabe.
- El ojo no es una esfera perfecta, y la zona del iris y la pupila presentan una elevación significativa respecto a la superficie de la esclerótica (superficie de la esfera).

El modelo puede ser complicado para ajustarse más a la realidad, por ejemplo asumiendo que el ojo en realidad es la unión de dos esferas, la primera estaría formada por la superficie de la córnea y la segunda por la superficie de la esclerótica. Sin embargo el aumento de la complejidad de la aproximación matemática hace aumentar el número de cálculos del proceso disminuyendo así la eficiencia de la aproximación.

Existe otra alternativa, mucho más simple que la anterior, consiste en olvidarse por completo de la forma esférica del ojo, y centrarse en la pupila, reconociendo su posición. Esta aproximación proporciona unos resultados realmente buenos según Babcock en [10] y [19].

Debido a su simplicidad y eficiencia demostrada, éste será el modelo elegido para el sistema a desarrollar.

4.2.3. Elección del Método de Calibrado

El algoritmo de detección de pupila tiene como objetivos proporcionar de forma robusta, fiable y eficiente una coordenada en un espacio bidimensional. Hasta ahora lo único que podíamos conseguir es realizar un seguimiento fiable de la pupila, detectando incluso parpadeos. Pero llega el momento de ir más allá e integrar la posición de la pupila en el mundo en el que nos movemos.

Dado que uno de los objetivos del sistema es averiguar la posición del mundo al que el usuario está mirando se hace necesario un proceso de calibración que relacione posiciones de pupila con posiciones reales, de ahí surge la necesidad de un sistema de calibrado.

El método de calibración es dependiente del modelo ocular utilizado. En caso de haber elegido un modelo ocular geométrico, el método de calibración debería tratar con vectores (que representarían la dirección a la que el usuario está mirando) y con planos (del mundo real). Como se ha elegido un método más simple, el método de calibrado debe buscar correspondencias entre puntos (coordenadas) de posiciones de pupila y puntos del mundo exterior.

Por ello se ha pensado en presentar frente al usuario una serie de puntos a los que tendrá que ir mirando sucesivamente, mientras el programa calcula la posición de esos puntos (gracias a la camara del mundo) y la posición de la pupila (con la cámara del ojo) capturada cada vez que el usuario mira a un punto.

Dichos puntos formarán una matriz de 3x3, nueve en total, y están situados en un plano visible por el usuario, perpendicular a su eje de visión. Un "punto" es un círculo (o cuadrado) negro de entre 1 y 2 cm de diámetro sobre un plano de color blanco aunque también puede estar dibujado sobre un monitor, siendo de entre 40 y 80 pixeles su diámetro.

El resultado de este proceso son dos arrays de coordenadas bidimensionales reales.

Una vez hecho esto, tenemos nueve posiciones de pupila y sus nueve correspondencias pero ¿qué ocurre con los puntos intermedios? de todos las posibles posiciones en las que puede estar la pupila, sólo 9 tienen una solución. Hay que encontrar una forma de averiguar las correspondencias en todos los puntos intermedios.

De esta forma surge la necesidad de una interpolación, que permitirá encontrar una posición del mundo para cada posición de la pupila.

4.2.4. Elección del Método de Interpolación

La interpolación es necesaria para conseguir que todas las posiciones de la pupila tengan su posición correspondiente en el mundo real. Basándonos en los nueve puntos iniciales, debemos construir un método que permita, predecir de forma eficiente en tiempo y con el menor error posible, el resto de puntos.

Según la bibliografía consultada, existen tres métodos principales de interpolación que nos permiten conseguir nuestro objetivo:

- Interpolación bilineal: Es igual que la lineal en 2 dimensiones pero "extrapolada" a una tercera dimension. La interpolación lineal asume que las variaciones son lineales en cada una de las dos dimensiones de la muestra.
- Interpolación bicúbica: Aproxima los valores por una superficie polinómica bicúbica. Es muy utilizada en tratamiento de imágenes para hacer zoom o "estirar" una imagen.
- Kriging: Es un método de interpolación geoestadística muy utilizado por geógrafos para construir superficies en función de un reducido conjunto de valores. Es conocido con el nombre de predicción óptima.

Los dos primeros métodos tienen un claro inconveniente: las muestras de datos a partir de las cuales se construyen sus polinomios han de ser equidistantes. Hay que tener en cuenta que nuestras muestras de datos provienen de la proyección de un

plano sobre una cámara, están en perspectiva, y por lo tanto las coordenadas de cada punto no van a ser equidistantes.

El método kriging averigua los valores de los puntos intermedios en función de los valores de sus vecinos más cercanos, utilizando como distancia la distancia euclídea. Kriging obtiene una media ponderada de los valores de los vecinos, realizando la ponderación en función de la distancia al punto del que se quiere obtener el nuevo valor y según una función que determina el tipo de kriging a realizar. Existen principalmente cuatro tipos de kriging: lineal, esférico, gaussiano y exponencial. Cada uno de ellos determina la variación de influencia de los vecinos. Existe otro parámetro llamado h que puede ser entendido como "umbral de influencia" y que determina el valor a partir del cual un vecino se considera demasiado lejano, y no se tiene en cuenta al ponderar los pesos.

Veámoslo en detalle:
La estimación de un valor desconocido en el punto P, $Y_{E,P}$ ha de ser calculada usando una media ponderada de los valores conocidos:

$$Y_{E,P} = \sum W_i Y_i$$

El error de estimación será la diferencia entre el valor estimado y el valor real $Y_{A,P}$:

$$\varepsilon_P = (Y_{E,P} - Y_{A,P})$$

Si los pesos utilizados en la estimación suman uno, el valor estimado se dice que es insesgado y la varianza de estimación es:

$$s^2 = \frac{\sum_{i=1}^{n} (Y_{E,P} - Y_{A,P})_i^2}{n}$$

y el error estandard es:

$$s = \sqrt{s^2}$$

La estimación y el error de estimación dependen de los pesos elegidos. Kriging intenta elegir los pesos óptimos que producen el menor error. El procedimiento de kriging empieza con las siguientes ecuaciones:

$$\begin{aligned} W_{1\gamma}(h_{11}) + W_{2\gamma}(h_{12}) + W_{3\gamma}(h_{13}) &= \gamma(h_{1p}) \\ W_{1\gamma}(h_{21}) + W_{2\gamma}(h_{22}) + W_{3\gamma}(h_{23}) &= \gamma(h_{2p}) \\ W_{1\gamma}(h_{31}) + W_{2\gamma}(h_{32}) + W_{3\gamma}(h_{33}) &= \gamma(h_{3p}) \end{aligned}$$

Donde $\gamma(h_{ij})$ es la semivarianza entre los puntos de control i y j correspondiendo a la distancia entre ellos. La matriz es simétrica dado que $h_{ij} = h_{ji}$. Para asegurar que la estimación es insesgada se añade la ecuación:

$$W_1 + W_2 + W_3 = 1$$

También se añade una cuarta variable para asegurar que se obtiene el error de estimación. Esta variable se llama multiplicador de lagrange λ. Por lo tanto, el conjunto de ecuaciones es:

$$\begin{aligned}
&W_{1\gamma}(h_{11}) + W_{2\gamma}(h_{12}) + W_{3\gamma}(h_{13}) + \lambda = \gamma(h_{1p}) \\
&W_{1\gamma}(h_{21}) + W_{2\gamma}(h_{22}) + W_{3\gamma}(h_{23}) + \lambda = \gamma(h_{2p}) \\
&W_{1\gamma}(h_{31}) + W_{2\gamma}(h_{32}) + W_{3\gamma}(h_{33}) + \lambda = \gamma(h_{3p}) \\
&W_1 + W_2 + W_3 + 0 = 1
\end{aligned}$$

Que en notación matricial resulta:

$$\begin{pmatrix} W_{1\gamma}(h_{11}) & W_{2\gamma}(h_{12}) & W_{3\gamma}(h_{13}) & 1 \\ W_{1\gamma}(h_{21}) & W_{2\gamma}(h_{22}) & W_{3\gamma}(h_{23}) & 1 \\ W_{1\gamma}(h_{31}) & W_{2\gamma}(h_{32}) & W_{3\gamma}(h_{33}) & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} * \begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ \lambda \end{pmatrix} = \begin{pmatrix} \gamma(h_{1p}) \\ \gamma(h_{2p}) \\ \gamma(h_{3p}) \\ 1 \end{pmatrix}$$

Veámoslo mediante un ejemplo.
Supongamos un terreno montañoso del cual solo tenemos muestras de la altura de 3 puntos (imagen 4.15) y queremos averiguar cuál es la altura en un punto perteneciente a ese terreno. Los puntos rojos representan los puntos cuyos valores asociados (alturas) son conocidos. Pretendemos averiguar cuál es la altura en el punto verde.

La siguiente tabla muestra las coordenadas y la altura de cada uno de los puntos:

Nombre	Coordenada X	Coordenada Y	Altura
1	1	2	150
2	4	1	110
3	6	4	140
P	3	2	Desconocido

Las distancias entre los puntos conocidos y P son:

	1	2	3	P
1	0	3.16	5.39	2
2	–	0	3.61	1.41
3	–	–	0	3.61

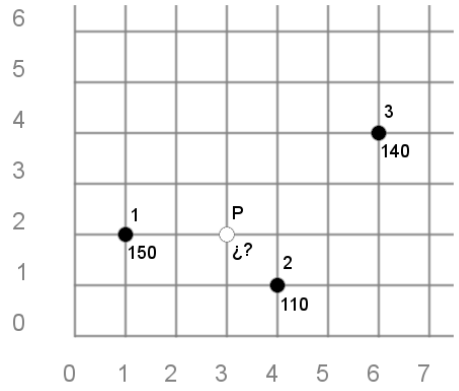


Figura 4.15: Ejemplo de Kriging

Utilizando la semivarianza lineal se obtiene

	1	2	3	P
1	0	12.65	21.54	8
2	–	0	14.42	5.66
3	–	–	0	14.42

y resolviendo las ecuaciones se obtiene:

$$\begin{pmatrix} W_1 \\ W_2 \\ W_3 \\ \lambda \end{pmatrix} = \begin{pmatrix} 0{,}3805 \\ 0{,}4964 \\ 0{,}1232 \\ -0{,}9319 \end{pmatrix}$$

$$Y_{E,P} = 0{,}3805 * (150) + 0{,}4964 * (110) + 0{,}1232 * (140) = 128{,}9 metros$$

En términos generales:

$$(A) * (W) = (B)$$

Una vez resulta la ecuación anterior, y obtenidos los pesos, se obtiene la estimación requerida mediante:

$$Y_{E,P} = W_1 * Y_1 + W_2 * Y_2 + W_3 * Y_3$$

El método utilizado para construir la superficie de correspondencias con los puntos de la pupila será la predicción óptima o Kriging.

Según la bibliografía consultada, muchos eye trackers basados en infrarrojos utilizan como valor para la posición de la pupila, las coordenadas relativas de ésta respecto al primer brillo córneo, porque supuestamente lo hace más invariante a los movimientos de la cabeza, pero esto presenta varios problemas. En primer lugar hay ocasiones en las que no existe un único brillo, dado que la córnea y la esclerótica no están situadas sobre una misma superficie esférica (ver figura 4.16).

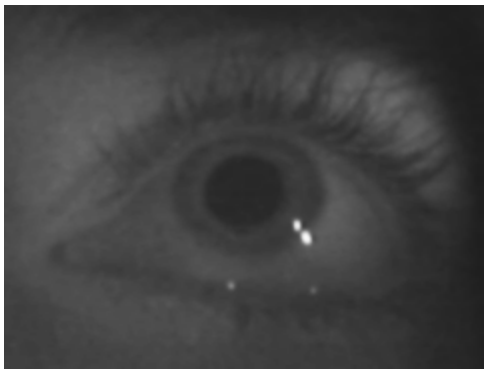


Figura 4.16: Problema con la doble reflexión

Además el hecho de elegir unas coordenadas relativas para la pupila implica un menor rango de coordenadas para la posición de la pupila, lo que provoca una menor resolución en la predicción.

4.2.5. Detección de los Puntos de Calibración

El algoritmo de detección de los puntos de calibración ha sido uno de los más modificados en el desarrollo del proyecto. Según lo visto en la sección del calibrado, un "punto" es un círculo (o cuadrado) negro de entre 1 y 2 cm de diámetro sobre un plano de color blanco aunque también puede estar dibujado sobre un monitor, siendo de entre 40 y 80 pixeles su diámetro.

El primer algoritmo implementado sólo buscaba zonas negras sobre fondo blanco y fallaba cuando las condiciones de luz no eran óptimas. Luego se mejoró, consiguiendo que los colores fueran relativos, es decir, no buscaba un "negro sobre blanco" sino más bien un "gris oscuro sobre gris claro". La segunda implementación era dependiente del tamaño del punto de calibración, dado que buscaba un tamaño determinado. Siguientes modificaciones del algoritmo fueron eliminando paulatinamente la detección de puntos falsos, es decir, puntos que "se parecían" a los de calibración pero que no lo eran.

El algoritmo actual realiza dos tipos de análisis para detectar los puntos, un primer análisis comprueba que realmente se trate de un punto y un segundo análisis selecciona 9 puntos de entre todos los posibles encontrados, descartando los que

considere que por su posición no pertenezcan al conjunto deseado.

En primer lugar, lo que se hace es un recorrido de la imagen en la que se deben encontrar los puntos. Para cada punto, se comprueba que él y sus 4 vecinos sean "oscuros" (con un valor de gris menor a 140). Si eso es así, vamos a suponer que nos encontramos en el interior de uno de esos puntos, lo que hay que hacer ahora es medir la distancia que separa el supuesto centro a cada uno de sus cuatro bordes principales, es decir, se mide la distancia desde ese punto hasta el límite izquierdo del punto, luego el derecho, el superior y el inferior. Si en alguna de esas distancias es superior a 15 pixeles se descarga. Debido a esto, la imagen que captura el punto debe mostrarlo como mucho con un radio de 30 pixeles (que no es la misma imagen que la que se dibuja en pantalla, en la que el punto tiene 40 pixels). Una vez que se tienen esas cuatro distancias, se comprueba que el punto candidato está centrando, y para ello comprobamos que esas cuatro distancias son "parecidas", es decir, que difieren en menos de 5 pixeles. Una vez realizado todo esto, se comprueba que la zona que rodea al punto candidato es más clara (mayor nivel de gris) que el punto central, para lo que se recorre un cuadrado alrededor del punto comprobando que sea cierto. Si todo lo anterior se cumple, se marca el punto como candidato en una nueva matriz (imagen).

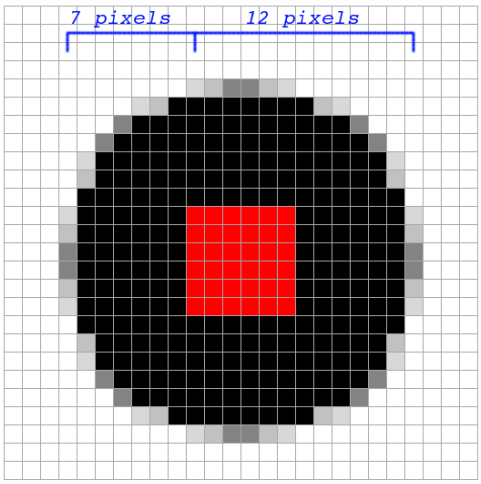


Figura 4.17: Punto de calibración:
En blanco los puntos descartados por ser demasiado claros. Los puntos negros son lo suficientemente oscuros, pero están demasiado cerca del borde. Los puntos marcados en rojo son puntos que originalmente estaban en negro y que además están lo suficientemente centrados como para pasar a la segunda fase del proceso.

En segundo lugar se recorre la imagen creada con todos los puntos candidatos. Dado que cada centro tendrá varios puntos candidatos situados en el centro (y adyacentes), se calcula el centro de gravedad de cada uno de esos grupos de puntos, almacenando el resultado en un vector.

Por último se comprueba que el número de puntos detectados y almacenados en el vector es el número de puntos esperados. Si el resultado fuese menor, es que algún punto se ha salido de la imagen, por lo tanto no se han detectado todos los que se esperaban. Si el resultado fuese mayor, es que ha habido algun punto infiltrado en la muestra y hay que descartarlo. El descarte se produce descartando el punto más alejado del centro del resto de los puntos.

4.2.6. Detección del Ojo Abierto

La posición de la pupila en el ojo no siempre es visible, en un caso ideal, el párpado está totalmente recogido y no oculta la pupila, pero cada poco tiempo se producen parpadeos, dependiendo la frecuencia del mismo de la iluminación, del ambiente, del cansancio del ojo y de la hidratación del mismo. Por ello es importante distinguir correctamente un ojo abierto de un ojo cerrado.

Teniendo en cuenta que para calcular el área de interes (ROI) de la imagen del ojo (que determinará la zona en la que buscar la pupila), es necesario realizar una captura de un ojo abierto, y otra captura de un ojo cerrado, y que dichas capturas se realizan en el proceso de calibración, se puede realizar una comparación entre la imagen actual y las dos imágenes de referencia y averiguar a cuál se parece más.

¿Cómo podemos realizar la compración? ¿Qué distancia utilizamos? Si la comparación la realizamos pixel a pixel (correlación), observaremos que la posición de la pupila influye significativamente en la detección de la apertura del ojo, pudiendo devolver resultados erróneos: ej. un ojo abierto con una pupila apuntando a la derecha está a una distancia similar de una pupila apuntando a la izquierda que de un ojo cerrado.

El histograma soluciona este problema, dado que es invariante a la posición de la pupila, hay *la misma* cantidad de blanco que de negro en una imagen con una pupila apuntando a la derecha que en otra apuntando a la izquierda, lo único que varía es la posición en la imagen de esos valores.

4.2.7. Escritura con los Ojos

Una vez creado el framework que permite detectar la pupila, detectar los puntos de calibración, calibrar el sistema y predecir el punto al que el ojo mira, se necesita construir una aplicación que utilice todas estas características para construir algo útil de cara al usuario. La aplicación propuesta permite que el usuario pueda escribir en el ordenador con sólo mirarlo. La primera parte de la aplicación consiste en averiguar a qué parte del mundo se está mirando, para ello se calibra el sistema, se obtiene la posición de la pupila y se predice el punto de interés, el resultado no es más que una coordenada real en un espacio bidimensional. La coordenada corresponde a la imagen del mundo que el usuario tiene frente a sus ojos (ver imagen

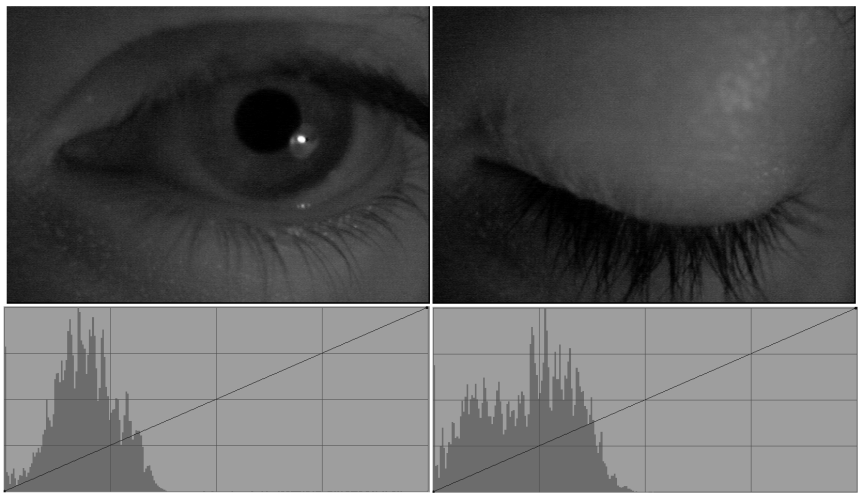


Figura 4.18: Comparativa de histogramas de ojos abierto y cerrado

4.19).

En segundo lugar, el programa ha de mostrar un teclado en pantalla que sea reconocido por el propio programa a través de la cámara del mundo, y que calcule la posición exacta del teclado a la que el usuario está mirando. Para ello hay varias opciones, pero la más cómoda sin duda, es que el programa dibuje varios puntos en posiciones predeterminadas de la pantalla, de forma que a partir de esos puntos reconozca la posición del teclado y lo relacione con la posición a la que se mira. Para ello en primer lugar realiza una interpolación para detectar el punto de interés dentro de la pantalla y luego busca cuál es la tecla que está en esa posición, si es que la hay, y pinta la letra que corresponde a la tecla cuando pasan 0.5 segundos.

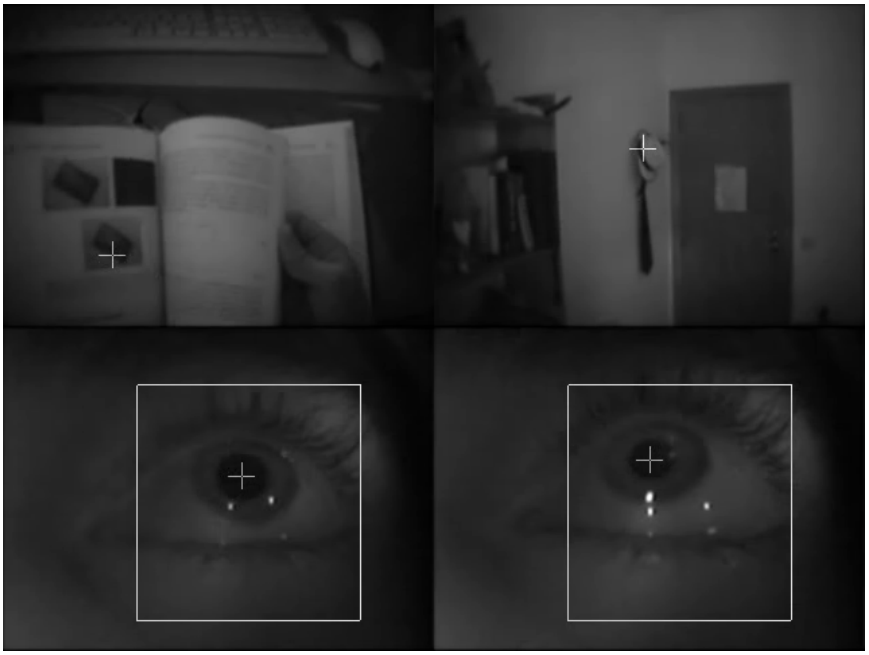


Figura 4.19: Predicción del punto de interés en varias situaciones

Capítulo 5

Herramientas Utilizadas

En el desarrollo del presente proyecto, se han utilizado gran variedad de herramientas. En este capítulo se listará cada una de ellas, indicando el motivo por el que se han usado.

5.1. Sistema Operativo

5.1.1. Debian GNU/Linux

Figura 5.1: Logo de Debian GNU/Linux

Debian GNU/Linux es una distribución de Linux cuyos principios están basados en software libre. Fue creada por el proyecto Debian en 1993. El proyecto Debian fue fundado en 1993 por Ian Murdock con el objetivo de tener una distribución abierta basada en el espíritu de Linux y GNU. Puede ser obtenida de la página oficial de Debian [2].

Debian se caracteriza por:

- La disponibilidad en varias plataformas hardware. La versión 3.1a es compatible con 11 plataformas.
- Una amplia colección de software disponible. La versión 3.1a viene con unos 15490 paquetes.

- Un grupo de herramientas para facilitar el proceso de instalación y actualización del software
- Su compromiso con los principios y valores involucrados en el movimiento del Software Libre.
- No tiene marcado ningún entorno gráfico en especial, ya sea GNOME, KDE u otro.

Debian ha sido el sistema elegido para el desarrollo por los siguientes motivos:

- Posee las ventajas de los sistemas GNU/Linux: eficiencia, estabilidad, configurabilidad.
- Existe una versión de las librerías IPP para GNU/Linux.
- De entre todos los Linux, el sistema de administración de paquetes de debian es el más fácil y eficiente, lo que facilita la instalación de software de forma rápida y cómoda.

Todas las máquinas utilizadas en el desarrollo utilizaban una distribución Debian 3.0 actualizada a los repositorios oficiales de Debian.

5.2. Programas

5.2.1. KDevelop

KDevelop es un entorno de desarrollo integrado para GNU/Linux y otros sistemas basados en UNIX. Está licenciado como GPL y soporta muchos lenguajes de programación. La versión utilizada en el proyecto es la 3.3. Posee un conjunto de características que lo convierten en el mejor entorno de desarrollo para GNU/Linux:

- Editor de código con resaltado de sintaxis e indentación automática.
- Gestión de proyectos para diferentes tipos: Automake, qmake y Ant,.
- Navegador de clases.
- Actúa como front-end para GCC y GDB.
- Generación automática de clases.
- Completado de código.
- Soporte integrado para Doxygen.

Para su instalación se ha utilizado la herramienta `apt-get` de debian y el paquete instalado ha sido `kdevelop`.

5.2.2. GCC

GNU Compiler Collection es un conjunto de compiladores creados por el proyecto GNU disponibles en todos los sistemas basados en UNIX e incluso en algunos de código cerrado, como Windows y Mac OS X. Las primeras versiones fueron escritas por Richard Stallman guiado por la Free Software Foundation. Los compiladores GCC trabajan sobre mayor número de procesadores que ningún otro y están licenciados bajo GPL. En el proyecto se utilizó GCC (g++) para compilar el código escrito en C++, a través de KDevelop. La instalación se realizó automáticamente con `apt-get` seleccionando el paquete `gcc-3.3`.

5.2.3. GDB

GNU Debugger (GDB) es el debugger disponible en los sistemas basados en UNIX que funciona para varios lenguajes de programación. Fue escrito por primera vez por Richard Stallman en 1988. GDB permite monitorizar el estado de las variables internas, así como modificar su valor y realizar llamadas a funciones independientemente del comportamiento del programa. Permite debuggear de forma remota mediante un protocolo serie o TCP/IP. Está licenciado bajo GPL. GDB es utilizado en el proyecto para trazar los programas mediante la interfaz de KDevelop. La instalación se realizó automáticamente con `apt-get` seleccionando el paquete `gdb`.

5.2.4. Doxygen

Doxygen es un generador de documentación automática para lenguajes similares a C. Es software GPL y está portado a la mayoría de los sistemas basados en UNIX e incluso a Mac OS X y Windows. La mayoría del código de Doxygen fue escrito por Dimitri van Heesch. Doxygen utiliza graphviz para crear los diagramas que muestra. Doxygen recibe un directorio y busca en él archivos fuente, devolviendo un conjunto de páginas de documentación en varios formatos a elegir: HTML, CHM, RTF, LaTeX o man. La documentación del framework fue realizada con Doxygen. La instalación se realizó automáticamente con `apt-get` seleccionando el paquete `doxygen` y el paquete `graphviz` (encargado de la realización de los diagramas).

5.2.5. Gimp

GNU Image Manipulation Program (Gimp) es un editor de imágenes GPL. Fue creado originalmente por unos estudiantes de la universidad de California Berkeley. Gimp es una de las herramientas libres más populares. Permite procesar imágenes digitales y fotografías. Es utilizada para crear logos, cambiar y recortar

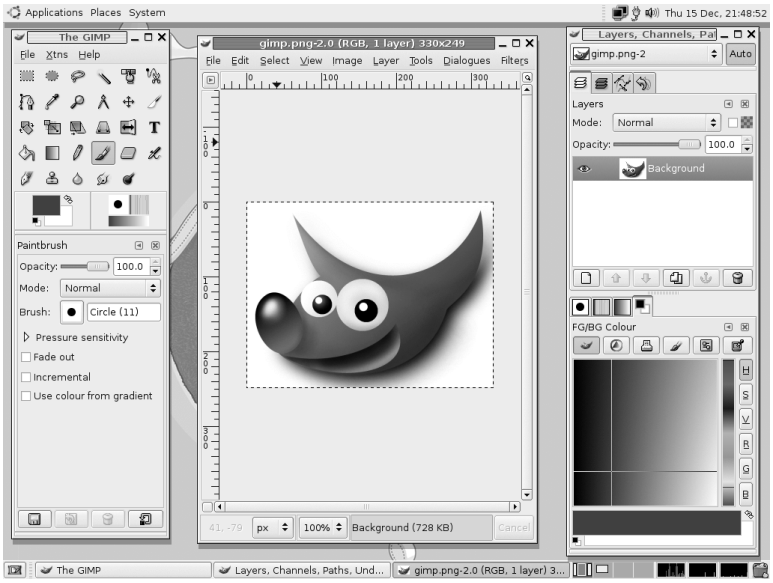


Figura 5.2: GNU Image Manipulation Program

el tamaño de fotografías, cambiar colores, mezclar imágenes, aplicar filtros, crear imágenes . . . En el proyecto se ha utilizado exhaustivamente durante la mayor parte del tiempo, para procesar las imágenes de las cámaras, averiguar transformaciones que mejoran la calidad de las imágenes, crear imágenes sintéticas para ser procesadas por la aplicación en las fases de prueba, mezclar y crear imágenes incluidas en la documentación. La instalación se realizó automáticamente con `apt-get` seleccionando el paquete `gimp`.

5.2.6. Kimage

Kimage es una aplicación GPL de tratamiento de imágenes creada por Enrique Turégano. Originalmente fue diseñada para la asignatura "Tratamiento de Imágenes" de la Universidad de Extremadura. Ha sido utilizada en el proyecto para realizar transformaciones en las imágenes obtenidas de las cámaras como aplicación de filtros y eliminación de ruido. El paquete se puede obtener a través de su autor: eturegano@gmail.com y su instalación se hace descomprimiendo el paquete y ejecutando `./configure` y `make` como usuario, y `make install` como root.

5.2.7. Kile

Kile es una herramienta GPL que permite editar documentos en TEXy LATEX. Puede ser compilada en sistemas basados en UNIX incluído Mac OS X. Funciona bajo

KDE y permite compilar los documentos con un solo click, tiene autocompletado, resaltado de sintaxis, permite la creación de proyectos, permite ver la estructura del documento en un árbol, crea esqueletos rápidamente y permite acceso a comandos y órdenes de LaTeX. Kile ha sido utilizado para realizar la documentación del proyecto. La instalación se realizó automáticamente con `apt-get` seleccionando el paquete `kile` para el entorno y los paquetes `tetex-bin tetex-extra` para LaTeX.

5.2.8. Microsoft Visio

Visio es un programa perteneciente a la suite Office utilizado para crear gráficos y diagramas vectoriales. Funciona sólo bajo Windows. Se ha utilizado para crear los Diagramas de Flujo de Datos del sistema implementado al no existir una alternativa con un funcionamiento razonable bajo GNU/Linux.

5.3. Librerías

5.3.1. Qt

Figura 5.3: Logo de la librería Qt

Qt es un framework de desarrollo de aplicaciones multiplataforma ampliamente utilizado para el desarrollo de interfaces gráficas de programas. El entorno de escritorio KDE (KDE Desktop Enviroment) está basado en Qt. Fue creado inicialmente por una empresa noruega llamada Trolltech, anteriormente conocidos como Quasar Technologies. Qt utiliza C++ estándar y puede ser utilizado para desarrollar aplicaciones en otros lenguajes de programación que dispongan de los correspondientes bindings: Python, Ruby, Perl, Pascal, Java ... Qt tiene una licencia dual: GPL y propietaria. Está disponible, entre otros, para el X11, Windows, y Mac OS X. Las interfaces gráficas de las aplicaciones que utilizan el framework de visión artificial en este proyecto presentado, están realizadas en Qt 3. La instalación se realizó automáticamente con `apt-get` seleccionando los paquete `libqt3-mt-dev` para las librerías y `qt3-designer` para el editor de ventanas.

5.3.2. IPP

Intel Integrated Performance Primitives es una librería de funciones multiplataforma altamente optimizadas utilizables para el desarrollo de aplicaciones multimedia o de tratamiento de datos:

- Codificación y decodificación de video.
- Codificación y decodificación de audio.
- Conversión de imágenes en color.
- Visión artificial.
- Compresión de datos.
- Procesamiento de cadenas.
- Procesamiento de señales.
- Procesamiento de imágenes.
- Reconocimiento de voz.
- Operaciones vectoriales y matriciales.
- Criptografía.

IPP abstrae la funcionalidad multimedia y de tratamiento de señales del procesador. Ello permite el uso transparente de la tecnología MMX, de las extensiones SIMD y de la tecnología XScale. Intel IPP está optimizada para muchos procesadores: Intel® Pentium® 4, Intel® Itanium® 2, Intel® Xeon(TM) e Intel® PCA basados en Intel XScale®. La aplicación desarrollada hace un uso intensivo de IPP, sobre todo al aplicar filtros a imágenes, transformaciones lineales de imágenes, morfología binaria, datos estadísticos de imágenes, obtención de histogramas, conversión de formatos, etc.

Para instalar IPP es necesario descargar el paquete `l_ipp_ia32_p_5.0.xxx.tgz` de la página de intel [6]. Tras descomprimir el paquete (`tar -zxvf l_ipp_ia32_p_5.0.xxx.tgz`) es necesario ejecutar el script `./install` situado en el directorio recién descomprimido. El script de instalación despliega un menú interactivo que nos muestra las opciones que debemos escoger en cada momento. Es necesario seleccionar la opción 1 e introducir el número de registro, una vez hecho esto el script comprobará que el sitema cumple los requisitos software necesarios y nos preguntará el directorio donde se ha de realizar la instalación. Cuando termine la copia de ficheros, el instalador preguntará si deseamos registrarnos.

5.3.3. STL

Standard Template Library (STL) es una librería que proporciona una serie de clases preparadas para ser usadas, como contenedores, iteradores y algoritmos. Utiliza templates, proporcionando polimorfismo en tiempo de compilación. Fue creada originalmente por Alexander Stepanov. Es una librería genérica con componentes altamente parametrizados. STL es utilizada en el proyecto a todos los niveles principalmente para gestionar las cadenas de caracteres y los vectores de elementos. La instalación de STL es automática en GNU/Linux al instalar `glibc` y `gcc`.

5.3.4. OpenCV

OpenCV es una librería de visión artificial de código abierto inicialmente desarrollada por Intel. Es utilizada para el procesamiento en tiempo real de imágenes. En el proyecto se ha utilizado OpenCV para comprobar el resultado de la detección de círculos mediante la transformada de Hough, así como para mostrar imágenes en la primera parte del proyecto. En la versión final no se hace uso de esta herramienta. Los paquetes necesarios para utilizar OpenCV son `libcv0.9.7-0`, `libcvaux0.9.7-0`, `libhighgui0.9.7-0`.

5.3.5. OpenGL

Open Graphics Library (OpenGL) es una librería multiplataforma que permite desarrollar aplicaciones que utilizan gráficos en 2D y 3D. Posee una serie de extensiones que permiten aprovechar las constantes evoluciones tecnológicas. A nivel básico, OpenGL es una especificación, es decir, un documento que describe una serie de funciones. A partir de esta especificación, los fabricantes de hardware crean implementaciones, de esta forma se aprovecha la aceleración hardware cuando sea posible. La especificación de OpenGL está revisada por el OpenGL Architecture Review Board (ARB). El ARB consiste en una serie de compañias interesadas en construir una API consistente y popular. Fue desarrollada inicialmente por SGI y está presente en la mayoría de sistemas operativos. La implementación utilizada es Mesa que es compatible en código con OpenGL y licenciada con licencia MIT. Los paquetes a instalar son `xlibmesa-dri`, `xlibmesa-gl` y `xlibmesa-gl-dev`.

5.3.6. FFmpeg

FFmpeg es una librería GPL que permite realizar conversiones de audio y video. Incluye libavcodec, una librería de codecs de audio y video. FFmpeg está desarrollada en GNU/Linux pero puede ser compilada para la mayoría de sistemas operativos. Se ha utilizado FFmpeg para guardar a disco las secuencias de

video obtenidas de las cámaras. Los paquetes necesarios para su utilización son `libavcodec-dev`, `libavcodec0d` y `libavcodeccvs`.

Parte III

Manuales

Capítulo 6

Manual de Usuario

En este capítulo se explicarán los pasos necesarios para conseguir hacer funcionar el sistema, y se realizará una descripción de cada uno de los programas que lo componen, explicando paso a paso los pasos necesarios para poder escribir con los ojos. La aplicación permite que la calibración y puesta a punto del sistema se pueda hacer con una sola persona.

Existen cuatro programas distintos dentro del sistema implementado, cada uno para una necesidad:

- eyeboardconfig se utiliza para establecer parámetros globales del sistema, tales como la resolución del monitor, el ancho y el alto de la captura...
- itune sirve para realizar la calibración del sistema y comprobar que las cámaras estén bien colocadas y funcionando correctamente.
- eyeboard sirve para introducir texto en el ordenador con sólo mirar un teclado virtual mostrado en el monitor.
- infrared sirve para predecir el lugar de interes, es decir, el lugar hacia donde nuestros ojos están mirando.

6.1. Instalación y Puesta a Punto del Hardware

Para tener un sistema funcional completo, lo primero es instalar el hardware requerido. Para ello es necesario conectar dos tarjetas capturadoras a dos slots PCI libres de la máquina (imagen 6.1).

Después es necesario conectar los alimentadores del casco a una fuente de alimentación (ej. una regleta) y los cables de las cámaras (imagen 6.2) a las capturadoras.

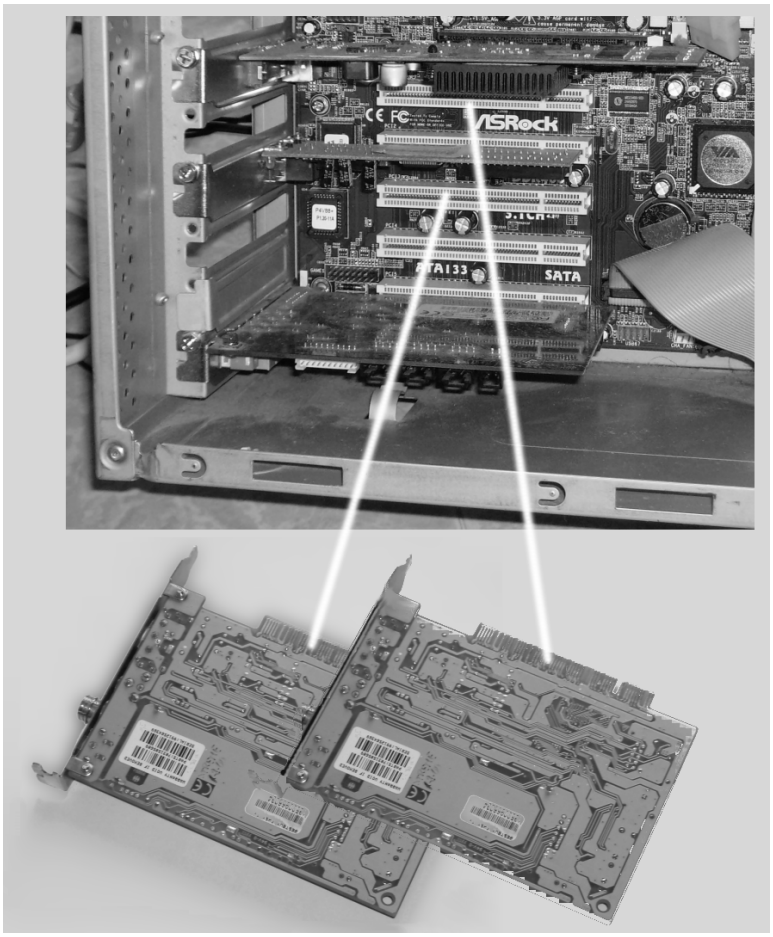


Figura 6.1: Capturadoras y PCI's libres

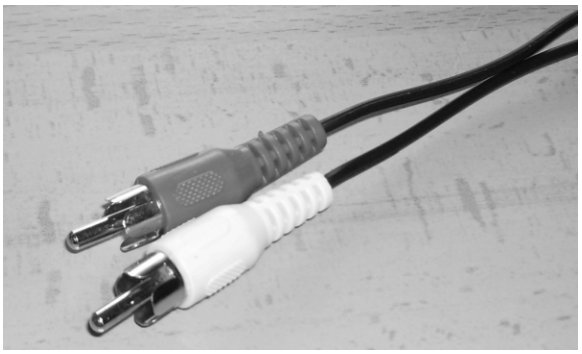


Figura 6.2: Cables RCA de las capturadoras

Por último, para comprobar el correcto funcionamiento del sistema, debemos encender la máquina y arrancar nuestro sistema GNU/Linux. Lo primero es indicarle al sistema que debe cargar un módulo que reconozca las tarjetas capturadoras. Este módulo se llama `bttv` y se carga con el comando `[root@ricercar: ]# modprobe bttv`. Para evitar tener que ejecutar este comando en cada arranque del sistema, se puede añadir al fichero `/etc/modules` una línea que contenga el nombre del módulo (`bttv`). Esto se puede hacer con el comando `[root@ricercar: ]# echo bttv ››/etc/modules`. Una vec cargado el módulo, y para comprobar que la instalación de las tarjetas, los cables y los alimentadores ha sido correcta es necesario ejecutar algún programa que realice una captura de video y la muestre por pantalla. Existen varias opciones:

- Utilizar el programa suministrado "itune". Sólo es necesario ejecutarlo y mostrará dos ventanas con las imágenes obtenidas de las cámaras.
- xawtv: es una aplicación de televisión para X11 que permite realizar captura de dispositivos de vídeo. Para ejecutarlo es necesario ejecutar `[enrique@ricercar: ]$ xawtv -c /dev/video0 -nodga`. El parámetro `/dev/video0` representa el dispositivo de captura, podría ser también `/dev/video1`. El parámetro `-nodga` es opcional.
- tvtime: es una aplicación de televisión que permite realizar capturas de dispositivos de vídeo. Para ejecutarlo basta con `[enrique@ricercar: ]$ tvtime`

Tras comprobar que las cámaras, los cables y los alimentadores funcionan correctamente, podemos pasar a instalar las librerías y el programa.

6.2. Instalación de las Librerías Necesarias

La instalación de las librerías se ha detallado en el capítulo herramientas. No obstante, el siguiente comando instalaría parte de las librerías necesarias:

```
[root@ricercar: ]# apt-get install libqt3-mt-dev
glibc gcc xlibmesa-dri xlibmesa-gl xlibmesa-gl-dev
libavcodec-dev libavcodec0d libavcodeccvs
```

La única librería no disponible en los repositorios de Debian es IPP. Para instalarla es necesario obtener una copia de la misma en la página de Intel ([6]). La instalación de la librería se detalla en el propio paquete que contiene la librería y en el capítulo de herramientas.

6.3. Compilación e Instalación del Programa

Para compilar los programas es necesario disponer del compilador GCC, que normalmente se encuentra instalado. Los cuatro programas se distribuyen en tarball's por lo que pueden ser compilados e instalados en cualquier distribución GNU/Linux. Para poder utilizar cualquiera de los cuatro paquetes es necesario realizar una serie de pasos:

- En primer lugar hay que descomprimir el paquete en cuestión con permisos de usuario: `[enrique@ricercar: ]$ tar zxvf eyeboardconfig-0.1.tar.gz`. Se creará un directorio llamado `eyeboardconfig`.
- Dentro de ese directorio es necesario ejecutar el comando `[enrique@ricercar: ]$ ./configure` que prepara la compilación del programa con permisos de usuario.
- Después hay que compilar el programa con `[enrique@ricercar: ]$ make` como usuario para compilar el programa.
- Por último realizar la instalación como superusuario con `[enrique@ricercar /home/enrique/eyeboardconfig]# make install` que copiará el binario en el directorio `/usr/bin`

Para ejecutar el programa basta con ejecutar `[enrique@ricercar: ]$ /usr/bin/eyeboardconfig`. La compilación e instalación de `itune`, `eyeboard` y `infrared` es igual a la anterior.

Una vez realizados estos pasos, se puede eliminar el directorio temporal donde se encuentra el código del paquete. Para desinstalar el programa hay que ejecutar el comando `[enrique@ricercar: /home/enrique/eyeboardconfig]# make uninstall`

La compilación e instalación de itune, eyeboard e infrared se realiza de la misma manera que la de eyeboardconfig, anteriormente explicada.

La ubicación de las librerías IPP puede tener que ser especificada para poder compilar el programa. Esto se hace de forma cómoda con el IDE KDevelop. En Proyecto/Opciones de proyecto/Opciones de configure/Opciones del preprocesador hay que introducir los directorios en los que se encuentran las librerías de IPP para poder compilar.

6.4. Configuración del Sistema con Eyeboardconfig

El programa eyeboardconfig sirve para configurar los parámetros que determinan el comportamiento global del sistema. Con eyeboardconfig se pue-

de elegir el tamaño de las imágenes que se procesan, la velocidad de captura...El programa se ejecuta escribiendo `[enrique@ricercar: ]$ /usr/bin/eyeboardconfig`. La interfaz de eyeboardconfig se puede ver en la imagen 6.3

Figura 6.3: Interfaz del programa EyeBoardConfig

Las parámetros configurables del sistema son:

- Ancho Captura: representa el ancho de las imágenes capturadas de las cámaras. Sólo se aceptan valores múltiplos de 32 entre 160 y 640. Las imágenes provenientes tanto de la cámara del ojo como de la cámara del mundo tendrán esta anchura.
- Alto Captura: representa el alto de las imágenes capturadas de las cámaras. Sólo se aceptan valores múltiplos de 32 entre 120 y 480. Las imágenes provenientes tanto de la cámara del ojo como de la cámara del mundo tendrán esta altura.
- Velocidad: representa el número de veces por segundo que se debe realizar el proceso de captura y proceso de las imágenes. Si la velocidad es demasiado elevada, se ajusta para realizar el proceso tan rápido como se pueda.
- Duración tecla: representa el tiempo necesario que el usuario ha de estar mirando una tecla para que la tecla se considere pulsada.
- Ancho Monitor: representa el ancho de la pantalla del usuario.

- Alto Monitor: representa el alto de la pantalla del usuario.
- Interpolación Mundo: contiene el tipo de interpolación a realizar en el cálculo del punto de mirada y la influencia (distancia) de los puntos.
- Interpolación Teclado: contiene el tipo de interpolación a realizar en el cálculo del punto exacto al que el usuario está mirando y la influencia de los puntos de detección en el proceso.
- Guardar Vídeo: si la casilla está seleccionada, se guarda un vídeo llamado outputXX.avi en el directorio de ejecución donde XX es el siguiente número en secuencia.

6.4.1. Fichero de Configuración del Sistema

El fichero de configuración del sistema tiene el sigiente aspecto:

```
width = 320
height = 240
width_screen = 1152
height_screen = 864
fps = 25
ms = 500
video = "false"
gravedad = "false"
int_world_type = "Spherical"
int_world_dist = 200
int_eye_type = "Spherical"
int_eye_dist = 40
```

y se encuentra situado en `.eyeboard/eyeboard.cfg` dentro del directorio `home` del usuario que ejecute la aplicación. Cada línea corresponde a una clave y un valor separados por un símbolo igual, separado de éstos por un espacio. Las cadenas de caracteres han de ir entrecomilladas. El programa EyeBoardConfig se encarga de generarlo automáticamente, por lo que el usuario no debería tocar este fichero.

6.5. Calibración y Uso de itune

La calibración es el proceso a través del cual el sistema reconoce posiciones del ojo y sus posiciones asociadas del mundo exterior. Es necesaria para poder ejecutar el programa de escritura. itune es el programa utilizado para calibrar el sistema. Para ejecutar itune es necesario `[enrique@ricercar: ]$ /usr/bin/itune`.

El usuario ha de llevar puesto “el casco” para utilizar itune. Una vez en ejecución, el programa itune tiene el aspecto de la figura 6.4.

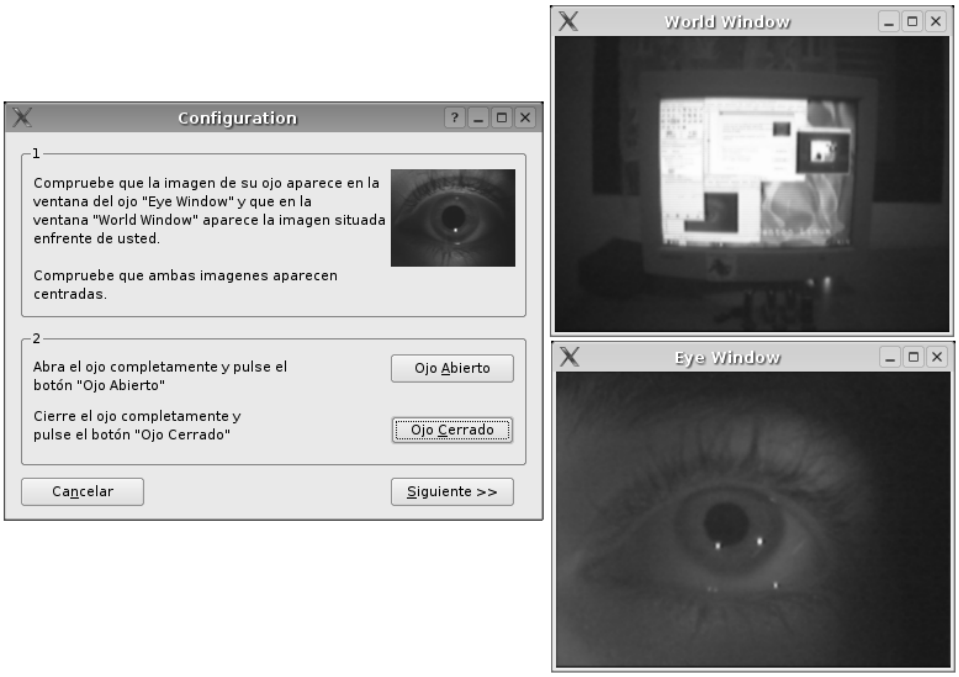


Figura 6.4: Interfaz del programa itune

La ventana de la izquierda aconseja sobre el uso del programa y las ventanas de la derecha muestran las imágenes del mundo y del ojo respectivamente. Para utilizar itune es necesario que el ojo aparezca centrado en la ventana del ojo y que el monitor también esté bien situado en la ventana del mundo. Si las imágenes del ojo y del mundo aparecen intercambiadas en sus respectivas ventanas, será necesario invertir el orden de conexión de los cables a las capturadoras.

Una vez centradas las imágenes, el sistema tiene que aprender qué es un ojo abierto y qué es un ojo cerrado. Para ello basta con pulsar sus respectivos botones. Abrimos el ojo y pulsamos el botón de “Ojo Abierto” y luego lo cerramos y pulsamos el de “Ojo Cerrado”. Una vez hecho esto, el sistema nos mostrará el área de interés de la imagen del ojo, es decir, aquella zona central donde se encuentra la pupila.

Tras los pasos anteriores, es necesario pulsar el botón “Siguiente” que nos llevará a la ventana de calibración. La ventana de calibración tiene el aspecto de la imagen 6.5.

En el programa de calibración aparecen 9 puntos repartidos por la pantalla y una pequeña imagen difuminada que contiene la imagen que la cámara del mundo está viendo. Para calibrar el sistema, el usuario ha de mirar cada uno de los 9 puntos y

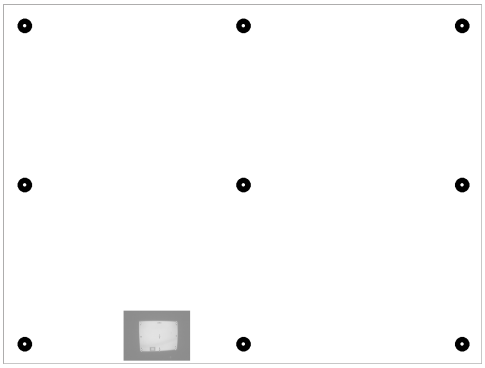


Figura 6.5: Interfaz del programa itune para la calibración

pulsar su número asociado en el teclado numérico. El punto de calibración superior izquierdo es el primero, por lo tanto el usuario ha de mirarlo, y una vez hecho esto, ha de pulsar la tecla "1" del teclado numérico, despues ha de mirar la segunda tecla, que está situada a la derecha de la primera, y pulsar la tecla "2" del teclado numérico, el proceso ha de repetirse para las 9 teclas. Cada punto de calibración tiene una zona interior de otro color, cuando está de color blanco indica que el punto está siendo reconocido y dispuesto para ser calibrado. Cuando está de color verde indica que ya ha sido calibrado, por lo que no hace falta pulsar su tecla asociada. Cuando los puntos se ponen de color rojo, indica que no están siendo reconocidos por el programa. La imagen difuminada de la parte inferior de la ventana contiene la imagen que la cámara del mundo (la que apunta hacia afuera) y se utiliza para comprobar que el monitor sigue centrado durante la calibración.

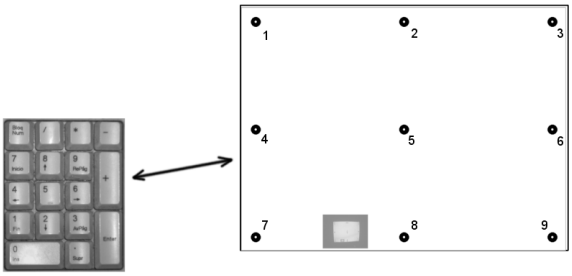


Figura 6.6: Teclado numérico y teclas asociadas

Si los puntos no son reconocidos, y se ponen de color rojo puede deberse a:

- La imagen de la cámara del mundo es defectuosa, compruebe que la pequeña imagen situada en la parte inferior del programa itune contiene el monitor situado enfrente de forma centrada.

- La distancia del usuario al monitor no es la correcta y los puntos no se detectan. Pongase a una distancia de unos 50-100 centímetros
- El usuario está realizando demasiados movimientos con la cabeza, la imagen no es estable y no se pueden detectar los puntos.
- Las condiciones de luz son extremas. Pruebe a establecer unas condiciones normales de iluminación.

Una vez calibrados los puntos aparecerá un mensaje en una ventana indicando que la calibración ha sido realizada con éxito. Tras esto se puede pasar a utilizar el programa eyeboard.

6.6. Escritura con la Mirada con EyeBoard

El programa EyeBoard sirve para escribir en un teclado virtual mostrado en pantalla con un solo ojo. Para poder utilizarlo es necesario ejecutar antes el programa `itune` para calibrar el sistema. Para lanzar la aplicación basta con ejecutar `[enrique@ricercar: ]$ eyeboard`. La interfaz del programa se muestra en la imagen 6.7. El programa muestra una serie de teclas en pantalla y una cruz de color rojo. La cruz de color rojo se mueve automáticamente a la tecla que el usuario está mirando. En la parte superior de la pantalla aparecerá el texto que el usuario está escribiendo. Cuando la cruz roja permanece en el interior de una tecla durante cierto tiempo, se añade su letra correspondiente al texto superior. En la parte exterior de la pantalla se muestran una serie de puntos. Son los llamados puntos de detección, utilizados para reconocer la posición de la pantalla y el teclado. Los parámetros de configuración más importantes que definen el comportamiento del programa son:

- gravedad: si se encuentra activada, el cursor rojo se vera “atraído” por el centro de cada tecla de forma que su posición se acerque al centro de la misma.
- ms: define el tiempo en milisegundos que debe permanecer el cursor rojo sobre una tecla para que sea considerada como pulsada. Un valor de 400-500 ms suele ser adecuado para la mayoría de los casos.

6.7. Realización de Pruebas con infrared

Infrared permite calcular la posición a la que estamos mirando sin necesidad de tener que mirar solamente a un teclado. La interfaz del programa infrared se puede ver en la imagen 6.8.

Figura 6.7: Interfaz del programa EyeBoard

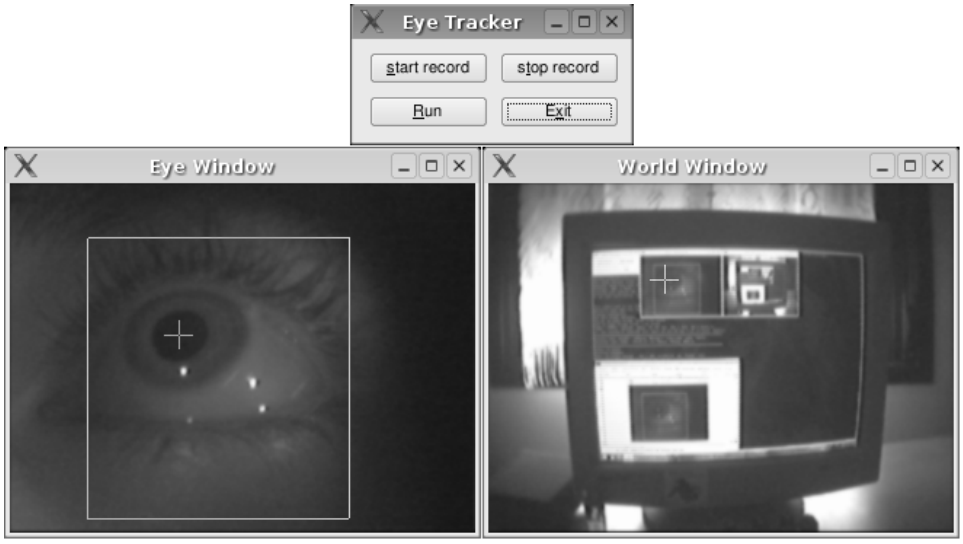


Figura 6.8: Interfaz de infrared

La interfaz es muy sencilla, el botón “run” sirve para que la aplicación comience a capturar, procesar y mostrar las imágenes de las cámaras. El botón “exit” es utilizado para terminar la aplicación. “start record” y “stop record” sirven para empezar a registrar las secuencias de video en ficheros y para terminar de hacerlo, respectivamente.

Infrared registra las secuencias de video de las cámaras y las graba en un fichero de video. Con este programa se puede observar el comportamiento de nuestros ojos frente a estímulos provenientes de películas, libros, páginas web que aparecen en el monitor. Este programa es especialmente útila para el estudio de la cognición, la percepción y algunos aspectos de la psicología humana. La imagen 6.9 muestra un ejemplo en el que el usuario se encuentra viendo un concierto musical.

Figura 6.9: Comportamiento frente a un concierto

Capítulo 7

Manual de Programador

Se intentarán dar unas nociones globales sobre el diseño del sistema implementado en este capítulo. Para implementar el sistema se ha elegido el lenguaje C++ debido a su eficiencia, velocidad y potencia, y se ha utilizado su capacidad de orientación a objetos para estructurar el código. Se ha hecho uso de la programación genérica gracias a STL.

7.1. Diseño Global

El reto al que nos enfrentamos en el proyecto es principalmente un procesamiento de imágenes y un cálculo asociado a las formas encontradas en esas imágenes. Existen dos tipos de secuencias de imágenes: aquella que provienen de la cámara que apunta al ojo y aquella que viene de la cámara del mundo, que apunta hacia el frente del usuario. Por lo tanto la primera acción a realizar es una captura de un dispositivo de imágenes. En segundo lugar es necesario realizar un procesamiento de las imágenes capturadas y obtener el significado de las acciones del sujeto en base a las posiciones de ciertos objetos en las imágenes. Por último es necesario, aunque no siempre tiene que serlo, el mostrar las imágenes capturadas y procesadas en pantalla. En resumen, las acciones a realizar son:

- Capturar imagen.
- Procesar imagen.
- Mostrar imagen.

En las primeras fases del proyecto, la captura se hacia de ficheros de video sin comprimir, por lo que se implementó un módulo para leer de videos en formato AVI sin comprimir: `AviReader`. En las siguientes fases del proyecto, la captura se hacía directamente del dispositivo, por lo que se implementó una clase llamada

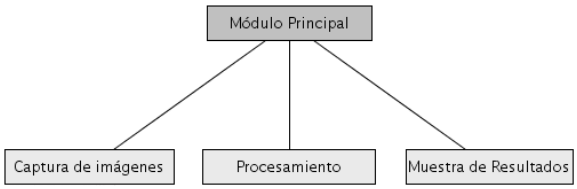


Figura 7.1: Mapa de estructura global del sistema

CamReader. Cada cámara tenía asociada en tiempo de ejecución una instancia de la clase encargada de la captura. Por otro lado el procesamiento es dependiente del contenido de la secuencia de vídeo por lo que se han implementado dos clases (`EyeProcessing` y `WorldProcessing`) que heredan de una clase común: `BaseProcessing`. Para mostrar las imágenes del proceso se ha implementado una clase llamada `GLWindow` que se encarga de crear una ventana en la que aparecen las imágenes que el usuario de la clase quiere mostrar.

La estructura de datos más importante del sistema es la clase `Image` que contiene un buffer con los pixeles de la imagen que representa. Para acelerar el intercambio entre las clases, se utilizan punteros a los datos de las imágenes.

Las clases de procesamiento `EyeProcessing` y `WorldProcessing` se encargan de analizar la imagen en busca de la pupila y los puntos de calibración o de detección (depende del estado en el que nos encontremos) y con esos datos, el módulo central se encarga de realizar los cálculos de posiciones (el lugar al que el usuario está mirando). Para realizar el proceso es necesario una clase llamada `Kriging` que se encarga de realizar los métodos matemáticos necesarios para los cálculos.

Existen otras clases con menor importancia cuya finalidad es proporcionar las herramientas necesarias a la parte central del proceso:

- `Distribution` almacena grandes cantidades de datos bidimensionales y realiza cálculos con ellos.
- `VideoWriter` se encarga convertir secuencias de vídeo en ficheros de vídeo. Se utiliza para dejar registro de todos los movimientos y capturas que se producen en el proceso.
- `Logger` se encarga de guardar en un fichero de texto un log con los eventos de la aplicación. Es útil para comprobar que todo se realiza de forma correcta.
- `Profiler` ha sido utilizado para medir el rendimiento de cada una de las partes del sistema.

- `Singleton` Gracias a este patrón de diseño, podemos acceder a la clase `Logger` y a la clase `Profiler` desde cualquier parte del programa.

El conjunto de clases anteriormente comentadas se encuentran agrupadas en namespaces, de forma que aquellas que realizan una función similar (ej. captura de vídeo de un fichero y captura de vídeo de un dispositivo) están en el mismo namespace.

El núcleo del sistema puede ser instanciado en cualquier programa, de forma que realice las acciones necesarias para cada una de las tareas. Actualmente hay tres programas que hacen uso del framework principal:

- itune: se encarga de obtener correspondencias entre posiciones de pupila y posiciones de los puntos de calibración y guardarlas en un fichero.

- eyeboard: se encarga de leer el fichero de correspondencias, obtener la posición de la pupila en cada momento y realizar una interpolación para saber a que lugar del mundo se encuentra mirando el usuario. Despues realiza otra interpolación para saber a que parte del teclado está mirando exactamente y por último dibuja el teclado y el cursor en la posición a la que se está mirando. Se utiliza para escribir con los ojos.

- infrared: su función es similar a la de eyeboard, solo que a infrared sólo le interesa la posición del mundo a la que el usuario está mirando. Permite grabar secuencias de vídeo con las imágenes de las cámaras y la posición a la que se está mirando. Se utiliza para comprobar a que parte del mundo está mirando el usuario.

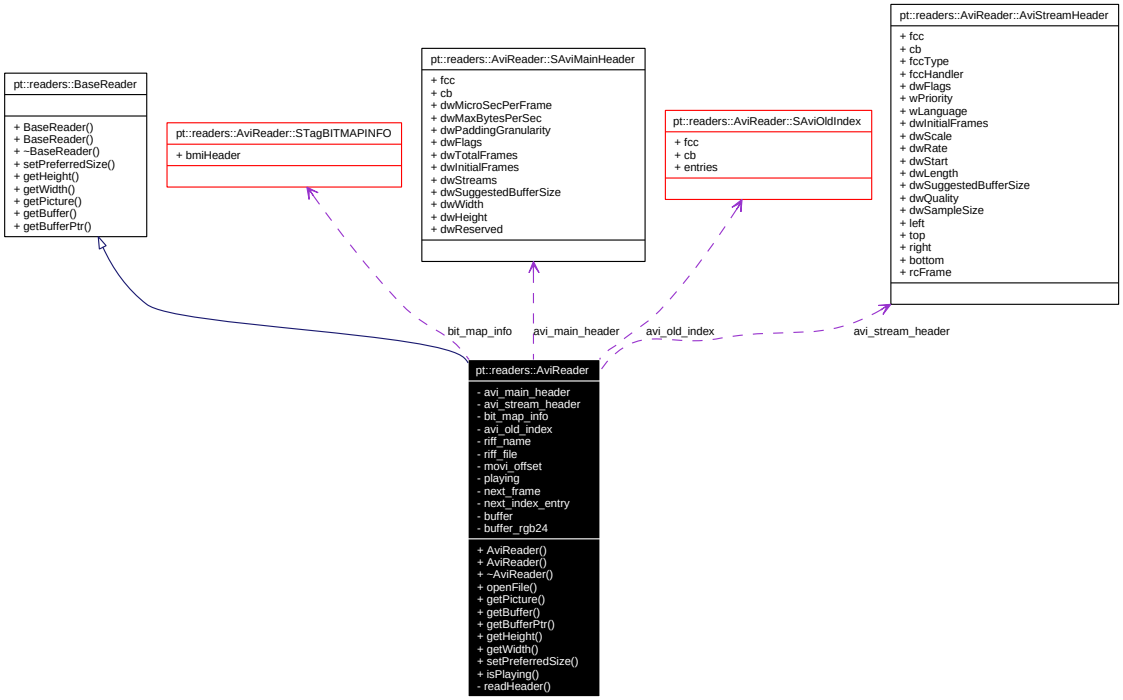
7.2. Documentación de clases

7.2.1. Referencia de la Clase pt::readers::AviReader

`#include <avireader.h>`

Diagrama de herencias de pt::readers::AviReader

Diagrama de colaboración para pt::readers::AviReader:



Métodos públicos

- **AviReader** ()
- **AviReader** (const char ∗file)
- ∼**AviReader** ()
- int **openFile** (const char ∗file)
- int **getPicture** ()
- char ∗ **getBuffer** () const
- const void ∗ **getBufferPtr** () const
- int **getHeight** () const
- int **getWidth** () const
- int **setPreferredSize** (int w, int h)
- bool **isPlaying** () const

Métodos privados

- int **readHeader** ()

Atributos privados

- **pt::readers::AviReader::SAviMainHeader avi_main_header**
- **pt::readers::AviReader::AviStreamHeader avi_stream_header**
- **pt::readers::AviReader::STagBITMAPINFO bit_map_info**
- **pt::readers::AviReader::SAviOldIndex avi_old_index**
- std::string **riff_name**
- FILE * **riff_file**
- int **movi_offset**
- bool **playing**
- int **next_frame**
- int **next_index_entry**
- char * **buffer**
- char * **buffer_rgb24**

Clases

- struct **AviStreamHeader**
- struct **SAviMainHeader**
- struct **SAviOldIndex**
- struct **SAviOldIndexEntry**
- struct **STagBITMAPINFO**
- struct **STagBITMAPINFOHEADER**
- struct **STagRGBQUAD**
- struct **SWaveFormatEx**

7.2.1.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Se encarga de leer secuencias de video de ficheros AVI sin comprimir. Lo ideal sería utilizar ffmpeg para leer cualquier tipo de fichero con cualquier tipo de compresión soportada, pero las salidas de xawtv son sin comprimir, por lo que no es necesario todo lo demás.

El formato avi divide las secuencias de video en "chunks". Un chunk puede ser audio o video y se encuentran intercalados en la zona de datos del fichero para que pueda ser leido de forma secuencial, sin necesidad de saltos. Al final del fichero AVI existe un indice que contiene apuntadores a las zonas de datos donde se encuentran los chunks.

Tareas Pendientes

soportar codecs y distintos tipos de ficheros de video.

7.2.1.2. Documentación del constructor y destructor

AviReader::AviReader ()

Constructor

AviReader::AviReader (const char * *file*)

Constructor con el nombre del fichero a leer

AviReader::~AviReader ()

Destructor

7.2.1.3. Documentación de las funciones miembro

char * AviReader::getBuffer () const `[virtual]`

Devuelve una copia de la imagen leida en RGB24

Implementa **pt::readers::BaseReader** (p. 117).

const void* pt::readers::AviReader::getBufferPtr () const `[inline]`

Devuelve una copia de la imagen leida

int AviReader::getHeight () const `[virtual]`

devuelve el alto de la imagen del fichero leido

Implementa **pt::readers::BaseReader** (p. 117).

int AviReader::getPicture () `[virtual]`

Lee la siguiente imagen del tren.

Implementa **pt::readers::BaseReader** (p. 118).

int AviReader::getWidth () const `[virtual]`

devuelve el ancho de la imagen

Implementa **pt::readers::BaseReader** (p. 118).

bool pt::readers::AviReader::isPlaying () const `[inline]`

devuelve true si se esta realizando la captura del fichero

int AviReader::openFile (const char $*$ ***file*****)**

Abre el fichero especificado

int AviReader::readHeader () `[private]`

Lee la cabecera del fichero

int pt::readers::AviReader::setPreferredSize (int *w*, int *h*) `[inline, virtual]`

Establece el ancho y el alto leido

Implementa **pt::readers::BaseReader** (p. 118).

7.2.1.4. Documentación de los datos miembro

struct pt::readers::AviReader::SAviMainHeader pt::readers::AviReader::avi_main_header `[private]`

Cabecera principal del fichero AVI

struct pt::readers::AviReader::SAviOldIndex pt::readers::AviReader::avi_old_index `[private]`

struct pt::readers::AviReader::AviStreamHeader pt::readers::AviReader::avi_stream_header `[private]`

struct pt::readers::AviReader::STagBITMAPINFO pt::readers::AviReader::bit_map_info `[private]`

Nota:

inexplicablemente los ficheros de xawtv no tienen el campo "STagRGBQUAD bmiColors[ ]"

char* pt::readers::AviReader::buffer [private]

Imagen en blanco y negro

char* pt::readers::AviReader::buffer_rgb24 [private]

Imagen leida

int pt::readers::AviReader::movi_offset [private]

Desplazamiento de la pelicula

int pt::readers::AviReader::next_frame [private]

contiene el siguiente frame que se ha de leer

int pt::readers::AviReader::next_index_entry [private]

La siguiente entrada en el indice (puede ser de video o de audio)

bool pt::readers::AviReader::playing [private]

true si se está leyendo del fichero

FILE* pt::readers::AviReader::riff_file [private]

Descriptor del fichero a cargar

std::string pt::readers::AviReader::riff_name [private]

Nombre del riff

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**avireader.h**
- eyeboard/src/**avireader.cpp**

7.2.2. Referencia de la Clase AviReader

```
#include <singleton.h>
```

7.2.2.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.9

Fecha:

2006

La clase Singleton implementa una estructura singleton, la cual permite que las clases que hereden de ella solo puedan ser instanciadas una vez, y a la vez accedidas desde cualquier otra clase que no sea la instanciadora

La documentación para esta clase fué generada a partir del siguiente archivo:

- eyeboard/src/**singleton.h**

7.2.3. Referencia de la Estructura pt::readers::AviReader::AviStreamHeader

Atributos públicos

- char **fcc** [4]
- DWORD **cb**
- char **fccType** [4]
- char **fccHandler** [4]
- DWORD **dwFlags**
- WORD **wPriority**
- WORD **wLanguage**
- DWORD **dwInitialFrames**
- DWORD **dwScale**
- DWORD **dwRate**
- DWORD **dwStart**
- DWORD **dwLength**
- DWORD **dwSuggestedBufferSize**
- DWORD **dwQuality**
- DWORD **dwSampleSize**

- struct {
  short int **left**
  short int **top**
  short int **right**
  short int **bottom**
  } **rcFrame**

7.2.3.1. Documentación de los datos miembro

short int pt::readers::AviReader::AviStreamHeader::bottom

DWORD pt::readers::AviReader::AviStreamHeader::cb

DWORD pt::readers::AviReader::AviStreamHeader::dwFlags

DWORD pt::readers::AviReader::AviStreamHeader::dwInitialFrames

DWORD pt::readers::AviReader::AviStreamHeader::dwLength

DWORD pt::readers::AviReader::AviStreamHeader::dwQuality

DWORD pt::readers::AviReader::AviStreamHeader::dwRate

DWORD pt::readers::AviReader::AviStreamHeader::dwSampleSize

DWORD pt::readers::AviReader::AviStreamHeader::dwScale

DWORD pt::readers::AviReader::AviStreamHeader::dwStart

DWORD pt::readers::AviReader::AviStreamHeader::dwSuggestedBuffer-Size

char pt::readers::AviReader::AviStreamHeader::fcc[4]

char pt::readers::AviReader::AviStreamHeader::fccHandler[4]

char pt::readers::AviReader::AviStreamHeader::fccType[4]

short int pt::readers::AviReader::AviStreamHeader::left

struct { ... } pt::readers::AviReader::AviStreamHeader::rcFrame

short int pt::readers::AviReader::AviStreamHeader::right

short int pt::readers::AviReader::AviStreamHeader::top

WORD pt::readers::AviReader::AviStreamHeader::wLanguage

WORD pt::readers::AviReader::AviStreamHeader::wPriority

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.4. Referencia de la Estructura pt::readers::AviReader::SAviMainHeader

Atributos públicos

- char **fcc** [4]
- DWORD **cb**
- DWORD **dwMicroSecPerFrame**
- DWORD **dwMaxBytesPerSec**
- DWORD **dwPaddingGranularity**
- DWORD **dwFlags**
- DWORD **dwTotalFrames**
- DWORD **dwInitialFrames**
- DWORD **dwStreams**
- DWORD **dwSuggestedBufferSize**
- DWORD **dwWidth**
- DWORD **dwHeight**
- DWORD **dwReserved** [4]

7.2.4.1. Descripción detallada

Cabecera principal del fichero AVI

7.2.4.2. Documentación de los datos miembro

DWORD pt::readers::AviReader::SAviMainHeader::cb

DWORD pt::readers::AviReader::SAviMainHeader::dwFlags

DWORD pt::readers::AviReader::SAviMainHeader::dwHeight

DWORD pt::readers::AviReader::SAviMainHeader::dwInitialFrames

DWORD pt::readers::AviReader::SAviMainHeader::dwMaxBytesPerSec

DWORD pt::readers::AviReader::SAviMainHeader::dwMicroSecPerFrame

DWORD pt::readers::AviReader::SAviMainHeader::dwPaddingGranularity

DWORD pt::readers::AviReader::SAviMainHeader::dwReserved[4]

DWORD pt::readers::AviReader::SAviMainHeader::dwStreams

DWORD pt::readers::AviReader::SAviMainHeader::dwSuggestedBufferSize

DWORD pt::readers::AviReader::SAviMainHeader::dwTotalFrames

DWORD pt::readers::AviReader::SAviMainHeader::dwWidth

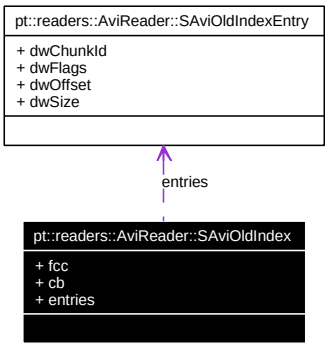
char pt::readers::AviReader::SAviMainHeader::fcc[4]

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.5. Referencia de la Estructura pt::readers::AviReader::SAviOldIndex

Diagrama de colaboración para pt::readers::AviReader::SAviOldIndex:



Atributos públicos

- char **fcc** [4]

  Specifies a FOURCC code. The value must be 'idx1'.

- DWORD **cb**

  Specifies the size of the structure, not including the initial 8 bytes.

- **SAviOldIndexEntry** ∗ **entries**

7.2.5.1. Documentación de los datos miembro

DWORD pt::readers::AviReader::SAviOldIndex::cb

Specifies the size of the structure, not including the initial 8 bytes.

SAviOldIndexEntry∗ **pt::readers::AviReader::SAviOldIndex::entries**

char pt::readers::AviReader::SAviOldIndex::fcc[4]

Specifies a FOURCC code. The value must be 'idx1'.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.6. Referencia de la Estructura pt::readers::AviReader::SAviOldIndexEntry

Atributos públicos

- DWORD **dwChunkId**
- DWORD **dwFlags**
- DWORD **dwOffset**
- DWORD **dwSize**

7.2.6.1. Descripción detallada

Una entrada situada en el índice del fichero (al final de todo)

7.2.6.2. Documentación de los datos miembro

DWORD pt::readers::AviReader::SAviOldIndexEntry::dwChunkId

DWORD pt::readers::AviReader::SAviOldIndexEntry::dwFlags

DWORD pt::readers::AviReader::SAviOldIndexEntry::dwOffset

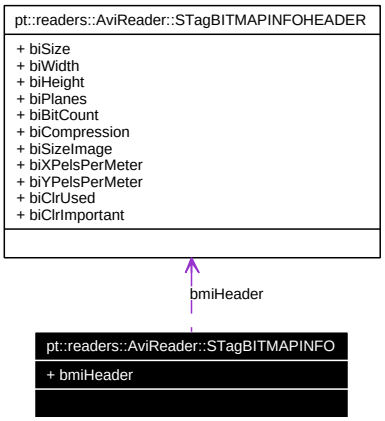
DWORD pt::readers::AviReader::SAviOldIndexEntry::dwSize

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.7. Referencia de la Estructura pt::readers::AviReader::STagBITMAPINFO

Diagrama de colaboración para pt::readers::AviReader::STagBITMAPINFO:



Atributos públicos

- **STagBITMAPINFOHEADER bmiHeader**

7.2.7.1. Descripción detallada

Nota:

inexplicablemente los ficheros de xawtv no tienen el campo "STagRGBQUAD bmiColors[ ]"

7.2.7.2. Documentación de los datos miembro

STagBITMAPINFOHEADER pt::readers::AviReader::STagBITMAPINFO::bmiHeader

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.8. Referencia de la Estructura pt::readers::AviReader::STagBITMAPINFOHEADER

Atributos públicos

- DWORD **biSize**
- LONG **biWidth**
- LONG **biHeight**
- WORD **biPlanes**

- WORD **biBitCount**
- DWORD **biCompression**
- DWORD **biSizeImage**
- LONG **biXPelsPerMeter**
- LONG **biYPelsPerMeter**
- DWORD **biClrUsed**
- DWORD **biClrImportant**

7.2.8.1. Documentación de los datos miembro

WORD pt::readers::AviReader::STagBITMAPINFOHEADER::biBitCount

DWORD pt::readers::AviReader::STagBITMAPINFOHEADER::biClrImportant

DWORD pt::readers::AviReader::STagBITMAPINFOHEADER::biClrUsed

DWORD pt::readers::AviReader::STagBITMAPINFOHEADER::biCompression

LONG pt::readers::AviReader::STagBITMAPINFOHEADER::biHeight

WORD pt::readers::AviReader::STagBITMAPINFOHEADER::biPlanes

DWORD pt::readers::AviReader::STagBITMAPINFOHEADER::biSize

DWORD pt::readers::AviReader::STagBITMAPINFOHEADER::biSizeImage

LONG pt::readers::AviReader::STagBITMAPINFOHEADER::biWidth

LONG pt::readers::AviReader::STagBITMAPINFOHEADER::biXPelsPerMeter

LONG pt::readers::AviReader::STagBITMAPINFOHEADER::biYPelsPerMeter

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.9. Referencia de la Estructura pt::readers::AviReader::STagRGBQUAD

Atributos públicos

- BYTE **rgbBlue**
- BYTE **rgbGreen**
- BYTE **rgbRed**
- BYTE **rgbReserved**

7.2.9.1. Descripción detallada

RGBQUAD

7.2.9.2. Documentación de los datos miembro

BYTE pt::readers::AviReader::STagRGBQUAD::rgbBlue

BYTE pt::readers::AviReader::STagRGBQUAD::rgbGreen

BYTE pt::readers::AviReader::STagRGBQUAD::rgbRed

BYTE pt::readers::AviReader::STagRGBQUAD::rgbReserved

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.10. Referencia de la Estructura pt::readers::AviReader::SWaveFormatEx

Atributos públicos

- WORD **wFormatTag**

- WORD **nChannels**
- WORD **nSamplesPerSec**
- DWORD **nAvgBytesPerSec**
- WORD **nBlockAlign**
- WORD **wBitsPerSample**
- WORD **cbSize**

7.2.10.1. Documentación de los datos miembro

WORD pt::readers::AviReader::SWaveFormatEx::cbSize

DWORD pt::readers::AviReader::SWaveFormatEx::nAvgBytesPerSec

WORD pt::readers::AviReader::SWaveFormatEx::nBlockAlign

WORD pt::readers::AviReader::SWaveFormatEx::nChannels

WORD pt::readers::AviReader::SWaveFormatEx::nSamplesPerSec

WORD pt::readers::AviReader::SWaveFormatEx::wBitsPerSample

WORD pt::readers::AviReader::SWaveFormatEx::wFormatTag

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**avireader.h**

7.2.11. Referencia de la Clase pt::processing::BaseProcessing

`#include <baseprocessing.h>`

Diagrama de herencias de pt::processing::BaseProcessing

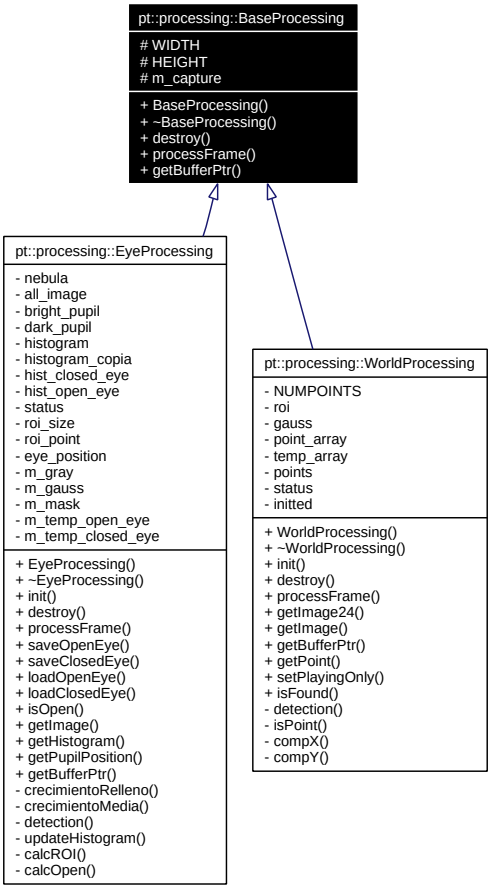
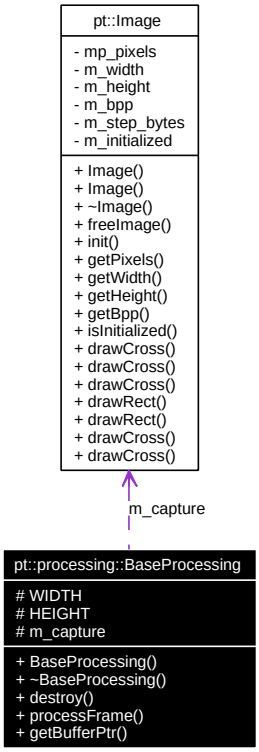


Diagrama de colaboración para pt::processing::BaseProcessing:



Métodos públicos

- **BaseProcessing** ()
- virtual ∼**BaseProcessing** ()
- virtual void **destroy** ()=0
- virtual void **processFrame** (Ipp8u ∗img)=0
- virtual const void ∗ **getBufferPtr** () const =0

Atributos protegidos

- int **WIDTH**
- int **HEIGHT**
- **Image m_capture**

7.2.11.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Es el padre de la jerarquía de procesamiento. Cada clase de procesamiento está asociada a una cámara y se encarga de realizar los procesos necesarios para el correcto tratamiento de la información:

- Filtrados
- Búsqueda de formas
- Obtención de características
- Detección de objetos

7.2.11.2. Documentación del constructor y destructor

pt::processing::BaseProcessing::BaseProcessing ()

Constructor.

pt::processing::BaseProcessing::∼BaseProcessing () `[virtual]`

Destructor

7.2.11.3. Documentación de las funciones miembro

virtual void pt::processing::BaseProcessing::destroy () `[pure virtual]`

Debe destruir la memoria reservada en el proceso. Ha de ser reimplementada en las subclases

Implementado en **pt::processing::EyeProcessing** (p. 148), y **pt::processing::WorldProcessing** (p. 201).

virtual const void∗ pt::processing::BaseProcessing::getBufferPtr () const `[pure virtual]`

Devuelve un puntero a la imagen que está siendo procesada. Ha de ser reimplementada en las subclases

Implementado en **pt::processing::EyeProcessing** (p. 149), y **pt::processing::WorldProcessing** (p. 201).

virtual void pt::processing::BaseProcessing::processFrame (Ipp8u * *img*) `[pure virtual]`

Se encargara de procesar la imagen. Ha de ser reimplementada en las subclases

Implementado en **pt::processing::EyeProcessing** (p. 150), y **pt::processing::WorldProcessing** (p. 202).

7.2.11.4. Documentación de los datos miembro

int pt::processing::BaseProcessing::HEIGHT `[protected]`

Alto de la imagen que se procesa

Image pt::processing::BaseProcessing::m_capture `[protected]`

Imagen que contiene la captura realizada

int pt::processing::BaseProcessing::WIDTH `[protected]`

Ancho de la imagen que se procesa

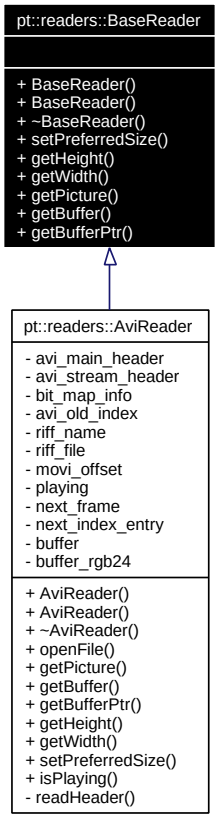
La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**baseprocessing.h**
- eyeboard/src/**baseprocessing.cpp**

7.2.12. Referencia de la Clase pt::readers::BaseReader

`#include <basereader.h>`

Diagrama de herencias de pt::readers::BaseReader



Métodos públicos

- **BaseReader** ()
- **BaseReader** (char ∗file)
- virtual ∼**BaseReader** ()
- virtual int **setPreferredSize** (int w, int h)=0
- virtual int **getHeight** () const =0
- virtual int **getWidth** () const =0
- virtual int **getPicture** ()=0
- virtual char ∗ **getBuffer** () const =0
- virtual const void ∗ **getBufferPtr** ()=0

7.2.12.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Está pensada para ser el padre de la jerarquía de readers, de forma que el usuario que quiera realizar una lectura de un dispositivo o de un fichero solo tenga que instanciarla y ella sola se encargue de todo.

Debería existir una jerarquía de clases y una ReaderFactory que se encargue de crear las instancias para cada reader.

7.2.12.2. Documentación del constructor y destructor

pt::readers::BaseReader::BaseReader ()

Constructor

pt::readers::BaseReader::BaseReader (char ∗ *file*)

Constructor, recibe el nombre del fichero (de video o dispositivo)

pt::readers::BaseReader::∼BaseReader () `[virtual]`

Destructor

7.2.12.3. Documentación de las funciones miembro

virtual char∗ pt::readers::BaseReader::getBuffer () const `[pure virtual]`

Devuelve un puntero a la imagen

Implementado en **pt::readers::AviReader** (p. 99).

virtual const void∗ pt::readers::BaseReader::getBufferPtr () [pure virtual]

Devuelve un puntero a la imagen

virtual int pt::readers::BaseReader::getHeight () const `[pure virtual]`

Devuelve la altura de la imagen

Implementado en **pt::readers::AviReader** (p. 99).

virtual int pt::readers::BaseReader::getPicture () `[pure virtual]`

Realiza la captura de la imagen

Implementado en **pt::readers::AviReader** (p. 99).

virtual int pt::readers::BaseReader::getWidth () const `[pure virtual]`

Devuelve la anchura de la imagen

Implementado en **pt::readers::AviReader** (p. 100).

virtual int pt::readers::BaseReader::setPreferredSize (int *w*, int *h*) `[pure virtual]`

Establece el tamaño que se desea para la imagen, en los dispositivos se puede configurar, pero en los ficheros no, por eso es setPreferredSize y no setSize. El usuario de la clase debe comprobar despues cual es el tamaño establecido

Implementado en **pt::readers::AviReader** (p. 100).

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**basereader.h**
- eyeboard/src/**basereader.cpp**

7.2.13. Referencia de la Clase pt::readers::CamReader

`#include <camreader.h>`

Métodos públicos

- **CamReader** ()
- ∼**CamReader** ()
- int **init** (const char ∗dev, int w, int h, int bpp)
- void **destroy** ()
- const void ∗ **getBufferPtr** () const
- void **getBuffer** (char ∗**buffer**)
- const bool **isRunning** () const

Métodos privados

- int **openDevice** (const char ∗dev)
- int **closeDevice** ()
- void **prepareBuffer** ()
- int **numChannels** () const
- int **setChannel** (int channel)
- int **getCurrentChannel** () const
- int **setPalette** (__u16 depth, __u16 pal)
- int **setPictureOptions** (__u16 brightness, __u16 hue, __u16 colour, __u16 contrast, __u16 whiteness)
- int **getPictureOptions** (struct video_picture ∗vp) const
- int **setPreferredSize** (int w, int h)
- int **getBufferSize** ()
- void **captureLoop** ()

Métodos privados estáticos

- static void ∗ **prePthread** (void ∗)

Atributos privados

- int **desc**
- int **curChannel**
- video_capability **vcaps**
- video_channel **vchannel**
- video_window **vwin**
- video_picture **vpic**
- char ∗ **buffer**
- int **bufferSize**
- pthread_t **thread**
- pthread_attr_t **thread_attr**
- volatile bool **running**
- int **reading_frame_n**
- const int **numBuffers**

7.2.13.1. Descripción detallada

Autor:

Pablo Márquez y Enrique Turégano

Versión:

0.8

Fecha:

2006

Crea un thread interno que captura imágenes de la camara en un buffer circular. El thread se destruye automaticamente cuando se elimina la clase, es ignorado por el usuario de la clase.

La lectura del dispositivo con esta clase es de 5.33 fps y para evitar solapamientos se crea un buffer circular al cual se accede desde fuera.

Utiliza v4l.

Compilar con -lpthread

7.2.13.2. Documentación del constructor y destructor

pt::readers::CamReader::CamReader ()

Constructor

pt::readers::CamReader::∼CamReader ()

Destructor

7.2.13.3. Documentación de las funciones miembro

void pt::readers::CamReader::captureLoop () `[private]`

Bucle de captura

int pt::readers::CamReader::closeDevice () `[private]`

Cierra el dispositivo

void pt::readers::CamReader::destroy ()

Libera memoria y recursos

void pt::readers::CamReader::getBuffer (char * *buffer*)

Realiza una copia del ultimo frame a buffer

Parámetros:

buffer puntero a la zona de memoria donde se quiere copiar FIXME: deberia ser reading_frame_n -1 pero con +7 funciona mejor. (WTF!?)

const void* pt::readers::CamReader::getBufferPtr () const `[inline]`

Devuelve un puntero a la ultima posicion que se leyó del buffer

int pt::readers::CamReader::getBufferSize () `[inline, private]`

Devuelve el tamaño del buffer (ancho * alto * profundidad_de_color)

int pt::readers::CamReader::getCurrentChannel () const `[private]`

Devuelve el canal actual

int pt::readers::CamReader::getPictureOptions (struct video_picture * *vp*) const `[private]`

Devuelve las opciones del picture

int pt::readers::CamReader::init (const char * *dev*, int *w*, int *h*, int *bpp*)

Inicia los parametros y la captura

const bool pt::readers::CamReader::isRunning () const `[inline]`

Devuelve true si se está realizando la captura

int pt::readers::CamReader::numChannels () const `[private]`

Canales disponibles

int pt::readers::CamReader::openDevice (const char * *dev*) `[private]`

Abre el dispositivo

void pt::readers::CamReader::prepareBuffer () `[private]`

Prepara el buffer en funcion del tamaño necesitado

void * pt::readers::CamReader::prePthread (void *) `[static, private]`

Prepara la ejecucion del thread

int pt::readers::CamReader::setChannel (int *channel*) `[private]`

Establece el canal del que se lee

int pt::readers::CamReader::setPalette (__u16 *depth*, __u16 *pal*) `[private]`

Establece la profundidad y el color de la imagen

int pt::readers::CamReader::setPictureOptions (__u16 *brightness*, __u16 *hue*, __u16 *colour*, __u16 *contrast*, __u16 *whiteness*) `[private]`

Establece el brillo, el hue, el color, el contraste y el whitenesss

int pt::readers::CamReader::setPreferredSize (int *w*, int *h*) `[private]`

Establece el tamaño

7.2.13.4. Documentación de los datos miembro

char* pt::readers::CamReader::buffer `[private]`

Contiene las imágenes leidas

int pt::readers::CamReader::bufferSize `[private]`

Tamaño del buffer (una imagen)

int pt::readers::CamReader::curChannel `[private]`

Canal actual

int pt::readers::CamReader::desc `[private]`

Descriptor del dispositivo

const int pt::readers::CamReader::numBuffers `[private]`

Tamaño de la cola circular de captura

int pt::readers::CamReader::reading_frame_n `[private]`

Frame actual de lectura

volatile bool pt::readers::CamReader::running `[private]`

Lectura ejecutandose

pthread_t pt::readers::CamReader::thread `[private]`

Thread de lectura

pthread_attr_t pt::readers::CamReader::thread_attr `[private]`

Atributos del thread

struct video_capability pt::readers::CamReader::vcaps `[private]`

Capabilities

struct video_channel pt::readers::CamReader::vchannel `[private]`

Propiedades del channel

struct video_picture pt::readers::CamReader::vpic `[private]`

Propiedades del picture

struct video_window pt::readers::CamReader::vwin `[private]`

Propiedades del window

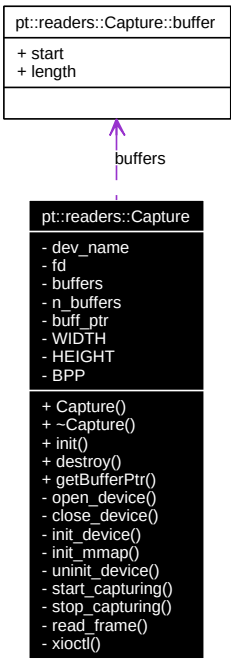
La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**camreader.h**
- eyeboard/src/**camreader.cpp**

7.2.14. Referencia de la Clase pt::readers::Capture

`#include <capture.h>`

Diagrama de colaboración para pt::readers::Capture:



Métodos públicos

- **Capture** ()
- ∼**Capture** ()
- int **init** (std::string dev, int w, int h, int bpp)
- void **destroy** ()
- void ∗ **getBufferPtr** ()

Métodos privados

- int **open_device** ()
- void **close_device** ()
- int **init_device** ()
- int **init_mmap** ()
- int **uninit_device** ()
- int **start_capturing** ()
- int **stop_capturing** ()
- int **read_frame** ()
- int **xioctl** (int **fd**, int request, void ∗arg)

Atributos privados

- std::string **dev_name**
- int **fd**
- **buffer** ∗ **buffers**
- unsigned int **n_buffers**
- char ∗ **buff_ptr**
- int **WIDTH**
- int **HEIGHT**
- int **BPP**

Clases

- struct **buffer**

7.2.14.1. Descripción detallada

Autor:

Enrique Turégano basado en el ejemplo de v4l de Michael H Schimek.

Versión:

0.1

Fecha:

2006

Reliza la captura de video de un dispositivo utilizando v4l2 y v4l. Utiliza mmap para acelerar la captura.

7.2.14.2. Documentación del constructor y destructor

pt::readers::Capture::Capture ()

Constructor

pt::readers::Capture::∼Capture ()

Destructor

7.2.14.3. Documentación de las funciones miembro

void pt::readers::Capture::close_device () `[private]`

Cierra el dispositivo

void pt::readers::Capture::destroy ()

Destruye la clase

void * pt::readers::Capture::getBufferPtr ()

Devuelve un puntero al buffer leido

int pt::readers::Capture::init (std::string *dev*, int *w*, int *h*, int *bpp*)

Inicia la clase.

int pt::readers::Capture::init_device () `[private]`

Inicia el dispositivo

int pt::readers::Capture::init_mmap () `[private]`

Inicia mmap

int pt::readers::Capture::open_device () `[private]`

Abre el dispositivo

int pt::readers::Capture::read_frame () `[private]`

Lee un frame de video

int pt::readers::Capture::start_capturing () `[private]`

Comienza la captura

int pt::readers::Capture::stop_capturing () `[private]`

Detiene la captura

int pt::readers::Capture::uninit_device () `[private]`

Cierra el dispositivo

int pt::readers::Capture::xioctl (int *fd*, int *request*, void * *arg*) `[private]`

Realiza una petición a ioctl

7.2.14.4. Documentación de los datos miembro

int pt::readers::Capture::BPP `[private]`

bits per pixel

char* pt::readers::Capture::buff_ptr `[private]`

Puntero al buffer

struct buffer* pt::readers::Capture::buffers `[private]`

Array con buffers donde se realiza la lectura

std::string pt::readers::Capture::dev_name `[private]`

Nombre del dispositivo

int pt::readers::Capture::fd `[private]`

Descriptor de fichero

int pt::readers::Capture::HEIGHT `[private]`

Alto de la captura

unsigned int pt::readers::Capture::n_buffers `[private]`

Numero de buffers utilizados

int pt::readers::Capture::WIDTH `[private]`

Ancho de la captura

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**capture.h**
- eyeboard/src/**capture.cpp**

7.2.15. Referencia de la Estructura pt::readers::Capture::buffer

Atributos públicos

- void * **start**
- size_t **length**

7.2.15.1. Descripción detallada

Buffer donde se almacena una imagen

7.2.15.2. Documentación de los datos miembro

size_t pt::readers::Capture::buffer::length

void* pt::readers::Capture::buffer::start

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**capture.h**

7.2.16. Referencia de la Clase pt::math::Definitions

`#include <maths.h>`

Atributos públicos estáticos

- static const float **e** = 2.718281828459
- static const float **pi** = 3.141592653589793238

7.2.16.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

7.2.16.2. Documentación de los datos miembro

const float pt::math::Definitions::e = 2.718281828459 `[static]`

Numero e

const float pt::math::Definitions::pi = 3.141592653589793238 `[static]`

Número pi

La documentación para esta clase fué generada a partir del siguiente archivo:

- eyeboard/src/**maths.h**

7.2.17. Referencia de la Clase pt::math::Distribution

`#include <distribution.h>`

Métodos públicos

- **Distribution** (int n=1000)
- ∼**Distribution** ()
- void **addPoint** (Ipp32f x, Ipp32f y)
- void **reset** ()
- **Real2dPoint getCenter** ()
- Ipp32u **getNumber** ()
- Ipp32f **xAt** (unsigned int n)
- Ipp32f **yAt** (unsigned int n)

Atributos privados

- Ipp32f ∗ **x_array**
- Ipp32f ∗ **y_array**
- Ipp32u **count**
- Ipp32u **cur_limit**

7.2.17.1. Documentación del constructor y destructor

pt::math::Distribution::Distribution (int *n* = `1000`**)**

Constructor, por cuestiones de optimización acepta un numero que representa el maximo número de elementos que contendrá el array, en caso de alcanzar ese límite, el array crece automáticamente, transparente al usuario de la clase. Trabaja con Ipp32f que es equivalente al tipo float de 32 bits.

Parámetros:

n tamaño inicial del array que contiene los puntos.

pt::math::Distribution::∼Distribution ()

Destructor

7.2.17.2. Documentación de las funciones miembro

void pt::math::Distribution::addPoint (Ipp32f *x*, Ipp32f *y*)

Añade un punto al array

Real2dPoint pt::math::Distribution::getCenter ()

Devuelve el centro de la distribución de los puntos

Ipp32u pt::math::Distribution::getNumber () `[inline]`

Devuelve el número de puntos que hay en el array

void pt::math::Distribution::reset ()

Vacia los arrays y establece el límite al valor por defecto

Ipp32f pt::math::Distribution::xAt (unsigned int *n*) `[inline]`

Devuelve el valor de la coordenada x pedida

Ipp32f pt::math::Distribution::yAt (unsigned int *n*) `[inline]`

Devuelve el valor de la coordenada y pedida

7.2.17.3. Documentación de los datos miembro

Ipp32u pt::math::Distribution::count `[private]`

Número de elementos en el array

Ipp32u pt::math::Distribution::cur_limit `[private]`

Límite de elementos en el array

Ipp32f* pt::math::Distribution::x_array `[private]`

Array de coordenadas X

Ipp32f* pt::math::Distribution::y_array `[private]`

Array de coordenadas y

La documentación para esta clase fué generada a partir de los siguientes archivos:

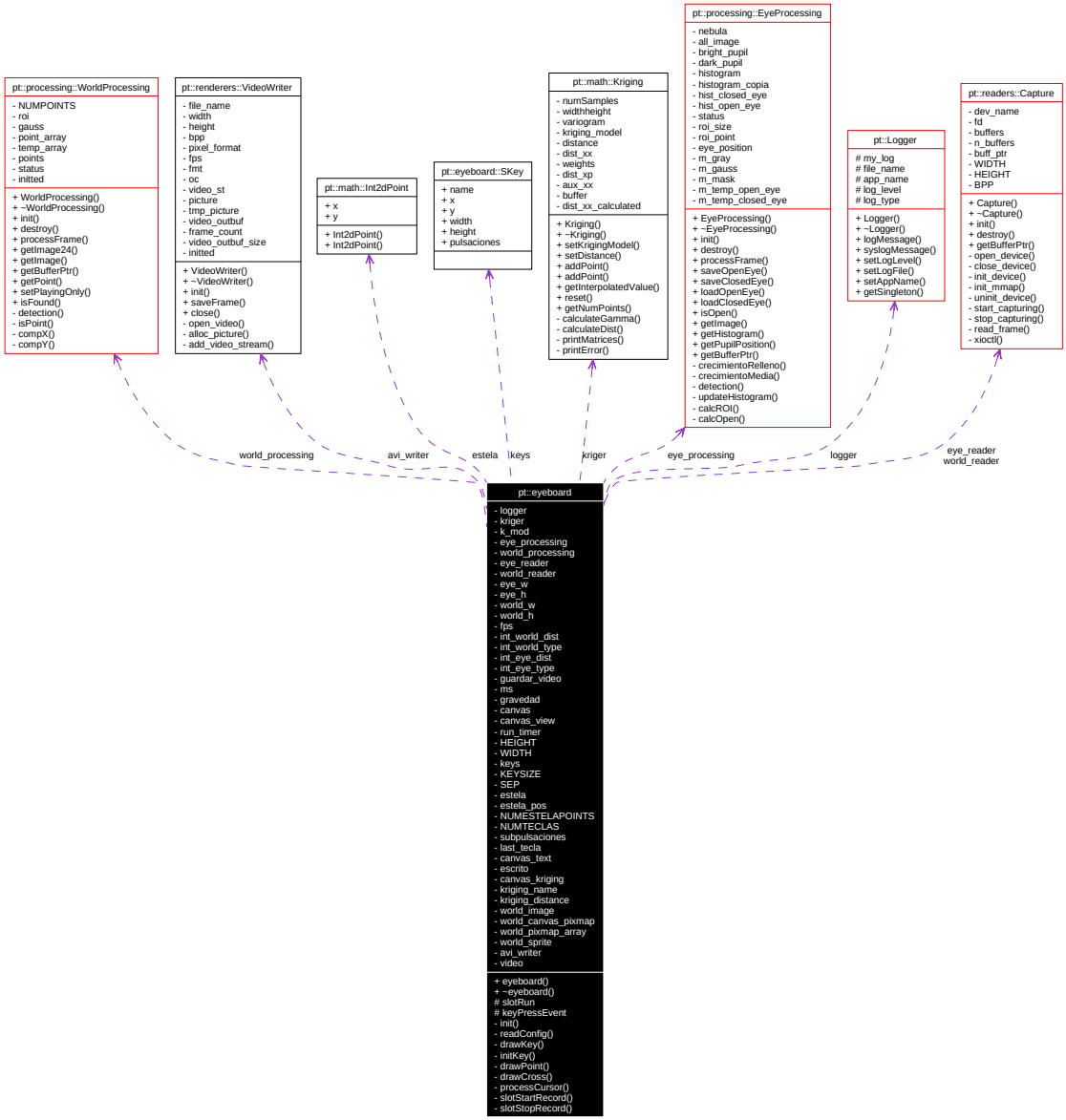
- eyeboard/src/**distribution.h**
- eyeboard/src/**distribution.cpp**

7.2.18. Referencia de la Clase pt::eyeboard

Application Main Window.

`#include <eyeboard.h>`

Diagrama de colaboración para pt::eyeboard:



Métodos públicos

- **eyeboard** ()
- virtual ∼**eyeboard** ()

Slots protegidos

- void **slotRun** ()

- void **keyPressEvent** (QKeyEvent ∗e)

Métodos privados

- void **init** ()
- void **readConfig** ()
- void **drawKey** (int k)
- void **initKey** (int id, int x, int y, int width, int height)
- void **drawPoint** (int x, int y)
- void **drawCross** (int x, int y)
- void **processCursor** (int x, int y)
- void **slotStartRecord** ()
- void **slotStopRecord** ()

Atributos privados

- **Logger logger**
- **math::Kriging** ∗ **kriger**
- **math::Kriging::EKrigingModel k_mod**
- **processing::EyeProcessing** ∗ **eye_processing**
- **processing::WorldProcessing** ∗ **world_processing**
- **readers::Capture** ∗ **eye_reader**
- **readers::Capture** ∗ **world_reader**
- int **eye_w**
- int **eye_h**
- int **world_w**
- int **world_h**
- int **fps**
- int **int_world_dist**
- std::string **int_world_type**
- int **int_eye_dist**
- std::string **int_eye_type**
- bool **guardar_video**
- int **ms**
- bool **gravedad**
- QCanvas ∗ **canvas**
- QCanvasView ∗ **canvas_view**
- QTimer ∗ **run_timer**
- int **HEIGHT**
- int **WIDTH**

- **SKey** ∗ **keys**
- int **KEYSIZE**
- int **SEP**
- **math::Int2dPoint** ∗ **estela**
- int **estela_pos**
- int **NUMESTELAPOINTS**
- int **NUMTECLAS**
- int **subpulsaciones**
- char **last_tecla**
- QCanvasText ∗ **canvas_text**
- QString **escrito**
- QCanvasText ∗ **canvas_kriging**
- QString **kriging_name**
- float **kriging_distance**
- QImage ∗ **world_image**
- QCanvasPixmap ∗ **world_canvas_pixmap**
- QCanvasPixmapArray ∗ **world_pixmap_array**
- QCanvasSprite ∗ **world_sprite**
- **renderers::VideoWriter** ∗ **avi_writer**
- Ipp8u ∗ **video**

Clases

- struct **SKey**

7.2.18.1. Descripción detallada

Application Main Window.

Autor:

enrique <enrique>

Versión:

0.1

7.2.18.2. Documentación del constructor y destructor

pt::eyeboard::eyeboard ()

Constructor

pt::eyeboard::~eyeboard () `[virtual]`

Destructor

7.2.18.3. Documentación de las funciones miembro

void pt::eyeboard::drawCross (int *x*, int *y*) `[private]`

Pinta la cruz roja que se utiliza para escribir

void pt::eyeboard::drawKey (int *k*) `[private]`

Pinta la tecla indexada por k

void pt::eyeboard::drawPoint (int *x*, int *y*) `[private]`

Pinta un punto de detección en el monitor

void pt::eyeboard::init () `[private]`

Inicia la instancia de la clase

void pt::eyeboard::initKey (int *id*, int *x*, int *y*, int *width*, int *height*) `[private]`

Inicia la tecla con esos valores

void pt::eyeboard::keyPressEvent (QKeyEvent * *e*) `[protected, slot]`

Llamado cuando el usuario pulsa una tecla

void pt::eyeboard::processCursor (int *x*, int *y*) `[private]`

Realiza las acciones necesarias ante las coordenadas x e y

void pt::eyeboard::readConfig () `[private]`

Lee la configuración

void pt::eyeboard::slotRun () `[protected, slot]`

Llmado cada 1/fps segundos

void pt::eyeboard::slotStartRecord () `[private]`

Llamado cuando se comienza a grabar el video

void pt::eyeboard::slotStopRecord () `[private]`

Llamado cuando se termina de grabar el video

7.2.18.4. Documentación de los datos miembro

renderers::VideoWriter* pt::eyeboard::avi_writer `[private]`

Se encarga de guardar un video con las secuencias de las cámaras

QCanvas* pt::eyeboard::canvas `[private]`

Canvas donde dibujar

QCanvasText* pt::eyeboard::canvas_kriging `[private]`

canvas donde se muestra el tipo de kriging

QCanvasText* pt::eyeboard::canvas_text `[private]`

es el canvas donde se muestra el texto que el usuario escribe

QCanvasView* pt::eyeboard::canvas_view `[private]`

CanvasView donde dibujar

QString pt::eyeboard::escrito `[private]`

El texto que el usuario ha escrito

math::Int2dPoint* pt::eyeboard::estela `[private]`

Almacena las ultimas posiciones del cursor

int pt::eyeboard::estela_pos `[private]`

Posicion de la estela actual

int pt::eyeboard::eye_h `[private]`

Alto de la imagen del ojo

processing::EyeProcessing* pt::eyeboard::eye_processing [private]

Instancia que se encarga de procesar la imagen de la cámara del ojo

readers::Capture* pt::eyeboard::eye_reader [private]

Se encarga de capturar las imágenes que vienen de la cámara del ojo

int pt::eyeboard::eye_w [private]

Ancho de la imagen del ojo

int pt::eyeboard::fps [private]

fps desired

bool pt::eyeboard::gravedad [private]

bool pt::eyeboard::guardar_video [private]

true si se debe guardar un video

int pt::eyeboard::HEIGHT [private]

Alto de la ventana

int pt::eyeboard::int_eye_dist [private]

valor de distancia de kriging en el proceso de interpolación de la cámara del ojo

std::string pt::eyeboard::int_eye_type [private]

tipo de interpolación del proceso del ojo

int pt::eyeboard::int_world_dist [private]

valor de distancia de kriging en el proceso de interpolación de la cámara del mundo

std::string pt::eyeboard::int_world_type [private]

tipo de interpolación del proceso del mundo

math::Kriging::EKrigingModel pt::eyeboard::k_mod [private]

Modelo de kriging

SKey∗ pt::eyeboard::keys [private]

Array con todas las teclas

int pt::eyeboard::KEYSIZE [private]

Ancho de la tecla a dibujar en pantalla

math::Kriging∗ pt::eyeboard::kriger [private]

Utilizado para realizar la interpolación

float pt::eyeboard::kriging_distance [private]

Distancia de kriging que se está utilizando

QString pt::eyeboard::kriging_name [private]

Cadena de texto con el tipo de kriging que se está haciendo

char pt::eyeboard::last_tecla [private]

Carácter de la última tecla pulasada

Logger pt::eyeboard::logger [private]

Necesario para logear los eventos del sistema

int pt::eyeboard::ms [private]

int pt::eyeboard::NUMESTELAPOINTS [private]

Numero de posiciones del cursor almacenables

int pt::eyeboard::NUMTECLAS [private]

Numero de teclas del teclado

QTimer* pt::eyeboard::run_timer [private]

Timer, desencadena la ejecucion de cada iteracion

int pt::eyeboard::SEP [private]

Separación de los puntos de control de los bordes

int pt::eyeboard::subpulsaciones [private]

Número de iteraciones en que se ha estado pulsando la ultima tecla. En cada segundo se producen fps iteraciones

Ipp8u* pt::eyeboard::video [private]

buffer que contiene la imagen que se guardará en el fichero de vídeo

int pt::eyeboard::WIDTH [private]

Ancho de la ventana

QCanvasPixmap* pt::eyeboard::world_canvas_pixmap [private]

CanvasPixmap donde se muestra la imagen del mundo

int pt::eyeboard::world_h [private]

Alto de la imagen del mundo

QImage* pt::eyeboard::world_image [private]

QImage que contiene la imagen de la cámara del mundo

QCanvasPixmapArray* pt::eyeboard::world_pixmap_array [private]

CanvasPixmapArray donde se muestra la imagen del mundo

processing::WorldProcessing* pt::eyeboard::world_processing [private]

Instancia que se encarga de procesar la imagen de la cámara del mundo

readers::Capture∗ pt::eyeboard::world_reader `[private]`

Se encarga de capturar las imágenes que vienen de la cámara del mundo

QCanvasSprite∗ pt::eyeboard::world_sprite `[private]`

Sprite donde se muestra la imagen del mundo

int pt::eyeboard::world_w `[private]`

Ancho de la imagen del mundo

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**eyeboard.h**
- eyeboard/src/**eyeboard.cpp**

7.2.19. Referencia de la Estructura pt::eyeboard::SKey

Atributos públicos

- char **name**
  Nombre de la tecla.
- int **x**
  coordenada x
- int **y**
  coordenada y
- int **width**
  ancho
- int **height**
  alto
- int **pulsaciones**
  pulsaciones

7.2.19.1. Descripción detallada

Contiene la estructura que representa la tecla que se muestra en el teclado de la pantalla

7.2.19.2. Documentación de los datos miembro

int pt::eyeboard::SKey::height

alto

char pt::eyeboard::SKey::name

Nombre de la tecla.

int pt::eyeboard::SKey::pulsaciones

pulsaciones

int pt::eyeboard::SKey::width

ancho

int pt::eyeboard::SKey::x

coordenada x

int pt::eyeboard::SKey::y

coordenada y

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**eyeboard.h**

7.2.20. Referencia de la Clase pt::processing::EyeProcessing

`#include <eyeprocessing.h>`

Diagrama de herencias de pt::processing::EyeProcessing

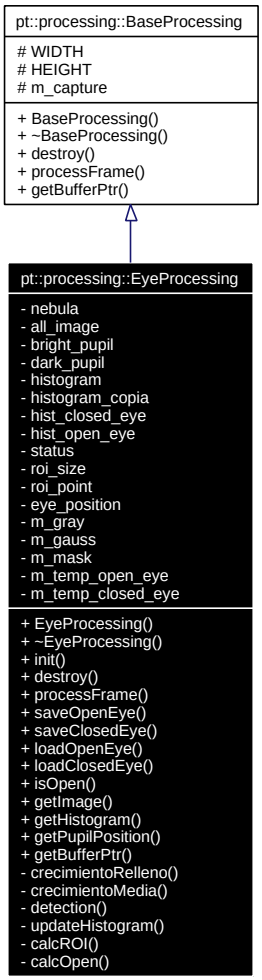
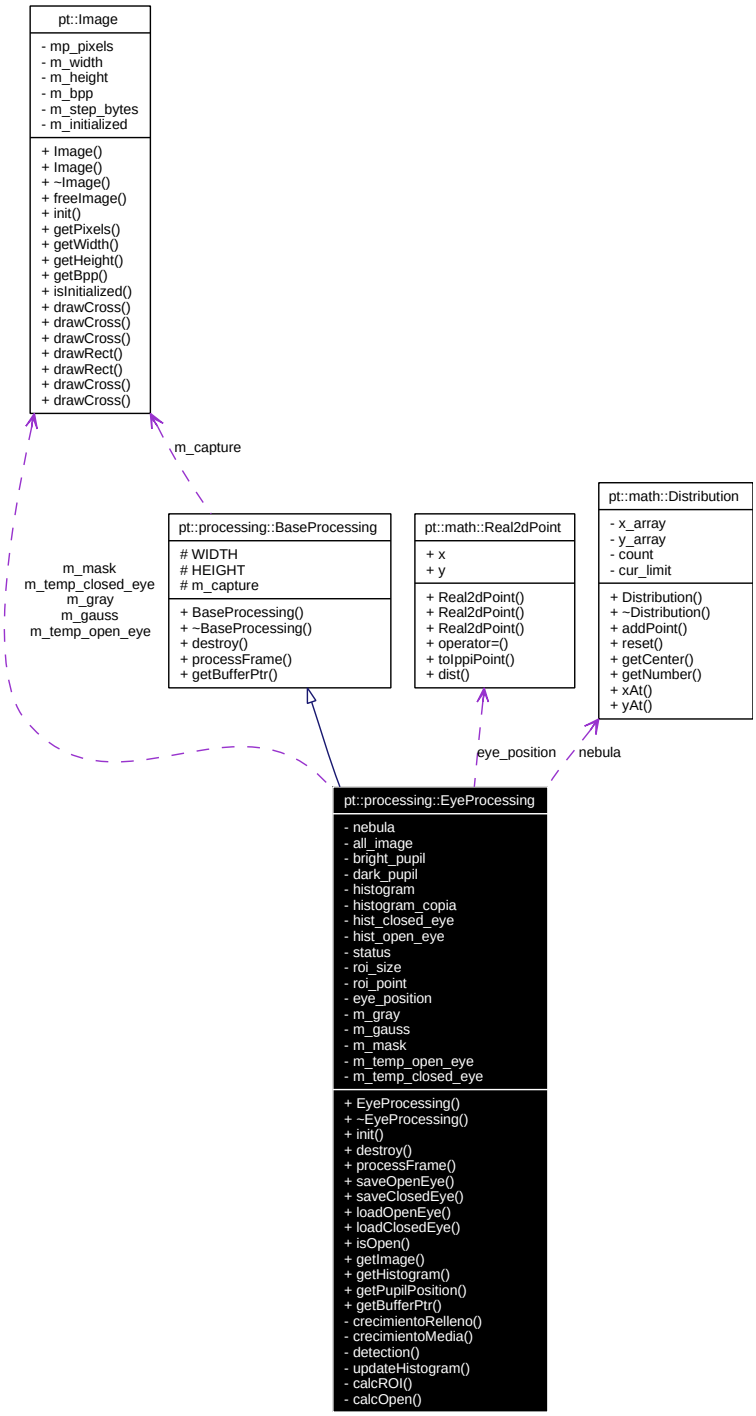


Diagrama de colaboración para pt::processing::EyeProcessing:


pt::Image
- mp_pixels
- m_width
- m_height
- m_bpp
- m_step_bytes
- m_initialized
+ Image()
+ Image()
+ ~Image()
+ freeImage()
+ init()
+ getPixels()
+ getWidth()
+ getHeight()
+ getBpp()
+ isInitialized()
+ drawCross()
+ drawCross()
+ drawCross()
+ drawRect()
+ drawRect()
+ drawCross()
+ drawCross()
m_capture
pt::processing::BaseProcessing
WIDTH
HEIGHT
m_capture
+ BaseProcessing()
+ ~BaseProcessing()
+ destroy()
+ processFrame()
+ getBufferPtr()
pt::math::Real2dPoint
+ x
+ y
+ Real2dPoint()
+ Real2dPoint()
+ Real2dPoint()
+ operator=()
+ toIppiPoint()
+ dist()
pt::math::Distribution
- x_array
- y_array
- count
- cur_limit
+ Distribution()
+ ~Distribution()
+ addPoint()
+ reset()
+ getCenter()
+ getNumber()
+ xAt()
+ yAt()
m_mask
m_temp_closed_eye
m_gray
m_gauss
m_temp_open_eye
eye_position
nebula
pt::processing::EyeProcessing
- nebula
- all_image
- bright_pupil
- dark_pupil
- histogram
- histogram_copia
- hist_closed_eye
- hist_open_eye
- status
- roi_size
- roi_point
- eye_position
- m_gray
- m_gauss
- m_mask
- m_temp_open_eye
- m_temp_closed_eye
+ EyeProcessing()
+ ~EyeProcessing()
+ init()
+ destroy()
+ processFrame()
+ saveOpenEye()
+ saveClosedEye()
+ loadOpenEye()
+ loadClosedEye()
+ isOpen()
+ getImage()
+ getHistogram()
+ getPupilPosition()
+ getBufferPtr()
- crecimientoRelleno()
- crecimientoMedia()
- detection()
- updateHistogram()
- calcROI()
- calcOpen()

Métodos públicos

- **EyeProcessing** ()
  Constructor.
- ∼**EyeProcessing** ()
  Destructor.
- void **init** (int w, int h)
  Inicia las variables.
- void **destroy** ()
  Libera memoria.
- void **processFrame** (Ipp8u ∗img)
  procesa el frame dado
- void **saveOpenEye** ()
  Almacena el histograma y la imagen del ojo abierto.
- void **saveClosedEye** ()
  Almacena el histograma y la imagen del ojo abierto.
- void **loadOpenEye** (char ∗path)
  Almacena el histograma y la imagen del ojo abierto.
- void **loadClosedEye** (char ∗path)
  Almacena el histograma y la imagen del ojo abierto.
- bool **isOpen** ()
  Devuelve la variable que indica si el ojo está abierto.
- void **getImage** (Ipp8u ∗i)
  Devuelve la imagen.
- void **getHistogram** (Ipp32s ∗h)
  Devuelve un puntero al histograma.
- bool **getPupilPosition** (**math::Real2dPoint** &point)
  Devuelve el punto al que se supone que se esta mirando.
- const void ∗ **getBufferPtr** () const

Tipos privados

- enum **EMachineStatus** {
  start, **playing**, **open_eye_saved**, **closed_eye_saved**,
  open_eye, **closed_eye**, **stopped** }

Métodos privados

- void **crecimientoRelleno** (**Image** &img, int x, int y, Ipp8u color, int step)
- void **crecimientoMedia** (**Image** &img, int x, int y, int step)
- void **detection** ()

  Realiza la deteccion de la pupila.

- void **updateHistogram** ()

  Actualiza el histograma.

- void **calcROI** ()

  calcula el area de interes: el area en la que se mueve la pupila

- void **calcOpen** ()

  Realiza la operacion que comprueba si el ojo está abierto y establece la variable.

Atributos privados

- **math::Distribution nebula**
- IppiSize **all_image**
- IppiPoint **bright_pupil**
- IppiPoint **dark_pupil**
- Ipp32s **histogram** [256]

  Histograma de la imagen.

- Ipp32s **histogram_copia** [256]

  Copia del histograma de la imagen: es más eficiente tener una copia que andar con varios mutexes...

- Ipp32s **hist_closed_eye** [256]

  Histograma del ojo cerrado.

- Ipp32s **hist_open_eye** [256]

Histograma del ojo abierto.

- **EMachineStatus status**

  Estado de la máquina.

- IppiSize **roi_size**

  tamaño del área de interés (sólo del ojo)

- IppiPoint **roi_point**

  origen del área de interés (sólo del ojo)

- **math::Real2dPoint eye_position**
- **Image m_gray**

  imagen transformada en gris

- **Image m_gauss**

  imagen sin ruido, en la que se realizan varios procesos

- **Image m_mask**

  Imagen en la que se realiza el crecimiento.

- **Image m_temp_open_eye**

  imagen temporal que almacena el ojo abierto

- **Image m_temp_closed_eye**

  imagen temporal que almacena el ojo cerrado

7.2.20.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Se encarga de procesar el tren de imágenes que viene de la cámara que apunta al ojo. Sus funciones son:

- Detectar la posición del ojo en la imagen
- Averiguar si está abierto o cerrado
- Detectar y calcular el centro de la pupila

necesita IPP

7.2.20.2. Documentación de las enumeraciones miembro de la clase

enum pt::processing::EyeProcessing::EMachineStatus `[private]`

Estados del proceso. el estado indica en que punto de la detección se encuentra **EyeProcessing**(p. 141).

Valores de la enumeración:

start La clase ha sido creada.

playing Se esta reproduciendo la secuencia capturada.

open_eye_saved Se ha guardado el ojo abierto, no el cerrado.

closed_eye_saved Se ha guardado el ojo cerrado, no el abierto.

open_eye El ojo esta abierto.

closed_eye El ojo esta cerrado.

stopped Se ha parado el proceso.

7.2.20.3. Documentación del constructor y destructor

pt::processing::EyeProcessing::EyeProcessing ()

Constructor.

Constructor

pt::processing::EyeProcessing::~EyeProcessing ()

Destructor.

Destructor

7.2.20.4. Documentación de las funciones miembro

void pt::processing::EyeProcessing::calcOpen () `[private]`

Realiza la operacion que comprueba si el ojo está abierto y establece la variable.

Si el ojo abierto y el cerrado han sido memorizados, cambia el estado de la maquina en funcion de la apertura del ojo. Se calcula la distancia manhattan de los histogramas, si está mas cerca del ojo abierto se considera abierto, y si no cerrado.

void pt::processing::EyeProcessing::calcROI () `[private]`

calcula el area de interes: el area en la que se mueve la pupila

Calcula la diferencia entre el ojo abierto y cerrado, en funcion de eso calcula el area de interes, que será el área que se procesará durante la detección. De esta forma se eliminan la mayor parte de los pixeles que no contienen información sobre el ojo en si.

void pt::processing::EyeProcessing::crecimientoMedia (Image & *img*, int *x*, int *y*, int *step*) `[inline, private]`

Recorre la pupila marcada, añadiendo los elementos a un vector. Despues se calculará la media

void pt::processing::EyeProcessing::crecimientoRelleno (Image & *img*, int *x*, int *y*, Ipp8u *color*, int *step*) `[inline, private]`

Marca la zona en la que se encuentra la pupila

void pt::processing::EyeProcessing::destroy () `[virtual]`

Libera memoria.

Reestablece el estado. No hace falta liberar espacio porque no hay nada dinámico.

Implementa **pt::processing::BaseProcessing** (p. 114).

void pt::processing::EyeProcessing::detection () `[private]`

Realiza la deteccion de la pupila.

Realiza el procesado principal de la imagen:

- En primer lugar se calcula si está el ojo abierto. Para ello llama a **calcOpen()**(p. 147) que realiza los cálculos en función del histograma.
- Luego se calcula el valor máximo: el primer brillo purkinje. Para que el ojo se considere abierto tiene que ser detectado como abierto y que ademas el punto más brillante sea mayor a 128, de modo contrario no se busca el centro de la pupila.

- Si el ojo está abierto se busca el punto más oscuro dentro del área de interés, que será considerado interior a la pupila.

- Una vez encontrado un punto interior, se realiza un crecimiento a la imagen, detectando todos los puntos que pertenecen al interior de la pupila.

- Después se eliminan los brillos, que son zonas demasiado claras como para ser consideradas pupilas.

- Luego se recorre la imagen añadiendo todos los puntos a un array y calculando el centro de gravedad de esa distribución.

const void* pt::processing::EyeProcessing::getBufferPtr () const `[inline, virtual]`

Devuelve un puntero a la zona de memoria donde se encuentra la imagen

Implementa **pt::processing::BaseProcessing** (p. 114).

void pt::processing::EyeProcessing::getHistogram (Ipp32s * *h*)

Devuelve un puntero al histograma.

Copia en h el histograma de la imagen actual.

void pt::processing::EyeProcessing::getImage (Ipp8u * *i*)

Devuelve la imagen.

Copia en i la imagen que se está procesando en este momento.

bool pt::processing::EyeProcessing::getPupilPosition (math::Real2dPoint & *point*)

Devuelve el punto al que se supone que se esta mirando.

Devuelve la posición en la que se encuentra el centro de la pupila.

void pt::processing::EyeProcessing::init (int *w*, int *h*)

Inicia las variables.

Inicia todas aquellas variables que deben conocer el tamaño de la imagen para ser iniciadas: las imagenes y all_image.

Parámetros:

w el ancho de la imagen que se le pasará a la clase.

h el alto de la imagen que se le pasará a la clase.

bool pt::processing::EyeProcessing::isOpen () `[inline]`

Devuelve la variable que indica si el ojo está abierto.

void pt::processing::EyeProcessing::loadClosedEye (char * *path*)

Almacena el histograma y la imagen del ojo abierto.

Carga el ojo cerrado de un fichero raw pasado como parametro.

void pt::processing::EyeProcessing::loadOpenEye (char * *path*)

Almacena el histograma y la imagen del ojo abierto.

Carga el ojo abierto de un fichero raw pasado como parametro.

void pt::processing::EyeProcessing::processFrame (Ipp8u * *img*) `[virtual]`

procesa el frame dado

Se ejecuta para cada frame, es llamado por la aplicación recibiendo imágenes en Gray 8. Hace una copia (a la vez que una gaussiana para eliminar ruido) llama a **updateHistogram()**(p. 151) para actualizar los valores del histograma y llama a **detection()**(p. 148) que calcula la posición de la pupila.

Parámetros:

img la imagen que se ha de procesar

Implementa **pt::processing::BaseProcessing** (p. 115).

void pt::processing::EyeProcessing::saveClosedEye ()

Almacena el histograma y la imagen del ojo abierto.

Almacena en variables de la clase el ojo cerrado, tanto la imagen como el histograma. De esta forma podremos calcular si el ojo está abierto o cerrado.

void pt::processing::EyeProcessing::saveOpenEye ()

Almacena el histograma y la imagen del ojo abierto.

Almacena en variables de la clase el ojo abierto, tanto la imagen como el histograma. De esta forma podremos calcular si el ojo está abierto o cerrado.

void pt::processing::EyeProcessing::updateHistogram () `[private]`

Actualiza el histograma.

Calcula el histograma de la imagen gauss. Imagen actual del ojo.

7.2.20.5. Documentación de los datos miembro

IppiSize pt::processing::EyeProcessing::all_image `[private]`

Tamaño de la imagen

IppiPoint pt::processing::EyeProcessing::bright_pupil `[private]`

Lugar en el que se encuentra el brillo de la pupila. Primer purkinge

IppiPoint pt::processing::EyeProcessing::dark_pupil `[private]`

Lugar en el que se encuentra el centro de la pupila

math::Real2dPoint pt::processing::EyeProcessing::eye_position `[private]`

Posición del ojo

Ipp32s pt::processing::EyeProcessing::hist_closed_eye[256] `[private]`

Histograma del ojo cerrado.

Ipp32s pt::processing::EyeProcessing::hist_open_eye[256] `[private]`

Histograma del ojo abierto.

Ipp32s pt::processing::EyeProcessing::histogram[256] `[private]`

Histograma de la imagen.

Ipp32s pt::processing::EyeProcessing::histogram_copia[256] `[private]`

Copia del histograma de la imagen: es más eficiente tener una copia que andar con varios mutexes...

Image pt::processing::EyeProcessing::m_gauss `[private]`

imagen sin ruido, en la que se realizan varios procesos

Image pt::processing::EyeProcessing::m_gray `[private]`

imagen transformada en gris

Image pt::processing::EyeProcessing::m_mask `[private]`

Imagen en la que se realiza el crecimiento.

Image pt::processing::EyeProcessing::m_temp_closed_eye `[private]`

imagen temporal que almacena el ojo cerrado

Image pt::processing::EyeProcessing::m_temp_open_eye `[private]`

imagen temporal que almacena el ojo abierto

math::Distribution pt::processing::EyeProcessing::nebula `[private]`

Array con todos los pixeles que se encuentran en el interior de la pupila

IppiPoint pt::processing::EyeProcessing::roi_point `[private]`

origen del área de interés (sólo del ojo)

IppiSize pt::processing::EyeProcessing::roi_size `[private]`

tamaño del área de interés (sólo del ojo)

EMachineStatus pt::processing::EyeProcessing::status `[private]`

Estado de la máquina.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**eyeprocessing.h**

- eyeboard/src/**eyeprocessing.cpp**

7.2.21. Referencia de la Clase pt::renderers::GLWindow

`#include <glwindow.h>`

Métodos públicos

- **GLWindow** (int w, int h, int **bpp**, std::string title, bool **fs**)
- ∼**GLWindow** ()
- void **setImage** (char ∗)
- int **update** ()

Métodos privados

- void **createGLWindow** ()
- bool **initGLScene** ()
- void **resizeGLScene** ()
- void **swapBuffers** ()
- void **initKeys** ()
- void **drawGLScene** ()
- void **killGLWindow** ()
- int **minPotSup2** (int)

Atributos privados

- Display ∗ **dpy**
- int **screen**
- Window **win**
- GLXContext **ctx**
- XSetWindowAttributes **attr**
- bool **fs**
- XF86VidModeModeInfo **deskMode**
- int **x**
- int **y**
- int **width**
- int **height**
- unsigned int **bpp**
- std::string **window_name**

- char * **image**
- GLuint **texid**
- double **scale_width**
- double **scale_height**
- bool **running**

7.2.21.1. Descripción detallada

Autor:

Enrique Turégano basado en el codigo base de NeHe

Versión:

0.1

Fecha:

2006

Es una clase que sirve para mostrar una imagen (o una secuencia de video) en una ventana del X. En su implementación se ha buscado la sencillez de cara al usuario de la clase, de forma que solo existan dos métodos: uno para establecer la imagen a mostrar (setImage) y otro para actualizar la ventana (update).

Utiliza OpenGL para mostrar la imagen, y acepta imagenes de profundidad 8 (en gris) y 24 en RGB. Selecciona el mejor modo de video disponible. Utiliza doble buffer.

necesita libxxf86vm-dev y libxxf86vm1

Tareas Pendientes

el tamaño de la imagen pasada y el tamaño de la ventana es el mismo, y no deberia serlo, deberia ser posible por ejemplo tener una ventana en modo fullscreen pero con una imagen de 320x240.

7.2.21.2. Documentación del constructor y destructor

pt::renderers::GLWindow::GLWindow (int *w*, int *h*, int *bpp*, std::string *title*, bool *fs*)

Inicia y crea la ventana con los parámetros pasados

pt::renderers::GLWindow::~GLWindow ()

Destruye la ventana

7.2.21.3. Documentación de las funciones miembro

void pt::renderers::GLWindow::createGLWindow () `[private]`

Crea la ventana del X

void pt::renderers::GLWindow::drawGLScene () `[private]`

Pinta la escena

bool pt::renderers::GLWindow::initGLScene () `[private]`

Inicia la escena de OpenGL

void pt::renderers::GLWindow::initKeys () `[private]`

Inicia el vector de teclas

void pt::renderers::GLWindow::killGLWindow () `[private]`

Destruye la ventana

int pt::renderers::GLWindow::minPotSup2 (int) `[private]`

Devuelve la minima potencia de dos superior al numero dado

void pt::renderers::GLWindow::resizeGLScene () `[private]`

Cambia el tamaño de la escena

void pt::renderers::GLWindow::setImage (char $*$)

Establece la imagen a mostrar

void pt::renderers::GLWindow::swapBuffers () `[private]`

Cambia el doble buffer

int pt::renderers::GLWindow::update ()

Actualiza la ventana

7.2.21.4. Documentación de los datos miembro

XSetWindowAttributes pt::renderers::GLWindow::attr `[private]`

Atributos de la ventana del X

unsigned int pt::renderers::GLWindow::bpp `[private]`

Bits por pixel

GLXContext pt::renderers::GLWindow::ctx `[private]`

Contexto de dispositivo para OpenGL

XF86VidModeModeInfo pt::renderers::GLWindow::deskMode `[private]`

Modo de video

Display∗ pt::renderers::GLWindow::dpy `[private]`

Display del X

bool pt::renderers::GLWindow::fs `[private]`

Fullscreen?

int pt::renderers::GLWindow::height `[private]`

Alto de la ventana

char∗ pt::renderers::GLWindow::image `[private]`

Imagen que se muestra

bool pt::renderers::GLWindow::running `[private]`

True si se está ejecutando

double pt::renderers::GLWindow::scale_height `[private]`

escalado de altura de la imagen. Necesario porque la textura ha de ser potencia de 2

double pt::renderers::GLWindow::scale_width `[private]`

escalado de anchura de la imagen. Necesario porque la textura ha de ser potencia de 2

int pt::renderers::GLWindow::screen `[private]`

Pantalla

GLuint pt::renderers::GLWindow::texid `[private]`

Identificador de la textura. La imagen es la textura

int pt::renderers::GLWindow::width `[private]`

Ancho de la ventana

Window pt::renderers::GLWindow::win `[private]`

Ventana

std::string pt::renderers::GLWindow::window_name `[private]`

Nombre que se muestra en la ventana

int pt::renderers::GLWindow::x `[private]`

foo variables

int pt::renderers::GLWindow::y `[private]`

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**glwindow.h**
- eyeboard/src/**glwindow.cpp**

7.2.22. Referencia de la Clase pt::Image

`#include <image.h>`

Métodos públicos

- **Image** ()

- **Image** (int w, int h, int bpp)
- ∼**Image** ()
- void **freeImage** ()

  Libera el espacio reservado.

- void **init** (int w, int h, int bpp=8)

  Inicia la imagen.

- Ipp8u ∗ **getPixels** () const

  Devuelve el puntero a la imagen.

- int **getWidth** () const

  Devuelve el ancho de la imagen.

- int **getHeight** () const

  Devuelve el alto de la imagen.

- int **getBpp** () const

  Devuelve los bits por pixel.

- bool **isInitialized** () const

  Devuelve m_initialized.

- void **drawCross** (int x, int y, int size=10)
- void **drawCross** (IppiPoint center, int size=10)
- void **drawCross** (**math::Real2dPoint** p)
- void **drawRect** (int min_x, int max_x, int min_y, int max_y)
- void **drawRect** (**Image** &img, int min_x, int max_x, int min_y, int max_y)

Métodos públicos estáticos

- static void **drawCross** (**Image** &img, IppiPoint center, int size=10)
- static void **drawCross** (void ∗img, int width, int height, IppiPoint center, int size=10)

Atributos privados

- Ipp8u ∗ **mp_pixels**

  Puntero a la imagen.

- int **m_width**
  Ancho de la imagen.
- int **m_height**
  Alto de la imagen.
- int **m_bpp**
  Bits por pixel.
- int **m_step_bytes**
  Distancia en bytes de una linea a la siguiente.
- bool **m_initialized**
  Si está inicializado.

7.2.22.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.2

Fecha:

2006

Image(p. 157) contiene un buffer para almacenar imagenes.

Tareas Pendientes

arreglar el caos de drawCross necesita IPP

7.2.22.2. Documentación del constructor y destructor

pt::Image::Image ()

pt::Image::Image (int *w*, int *h*, int *bpp*)

pt::Image::∼Image ()

7.2.22.3. Documentación de las funciones miembro

void pt::Image::drawCross (void * *img*, int *width*, int *height*, IppiPoint *center*, int *size* = 10) `[static]`

void pt::Image::drawCross (Image & *img*, IppiPoint *center*, int *size* = 10) `[static]`

Pinta una cruz en el punto dicho de la imagen.

void pt::Image::drawCross (math::Real2dPoint *p*) `[inline]`

void pt::Image::drawCross (IppiPoint *center*, int *size* = 10)

void pt::Image::drawCross (int *x*, int *y*, int *size* = 10)

void pt::Image::drawRect (Image & *img*, int *min_x*, int *max_x*, int *min_y*, int *max_y*)

Pinta un rectangulo en la imagen,

void pt::Image::drawRect (int *min_x*, int *max_x*, int *min_y*, int *max_y*)

void pt::Image::freeImage ()

Libera el espacio reservado.

int pt::Image::getBpp () const `[inline]`

Devuelve los bits por pixel.

int pt::Image::getHeight () const `[inline]`

Devuelve el alto de la imagen.

Ipp8u* pt::Image::getPixels () const `[inline]`

Devuelve el puntero a la imagen.

int pt::Image::getWidth () const `[inline]`

Devuelve el ancho de la imagen.

void pt::Image::init (int *w*, int *h*, int *bpp* = 8)

Inicia la imagen.

por defecto bpp=8

bool pt::Image::isInitialized () const [inline]

Devuelve m_initialized.

7.2.22.4. Documentación de los datos miembro

int pt::Image::m_bpp [private]

Bits por pixel.

int pt::Image::m_height [private]

Alto de la imagen.

bool pt::Image::m_initialized [private]

Si está inicializado.

int pt::Image::m_step_bytes [private]

Distancia en bytes de una linea a la siguiente.

int pt::Image::m_width [private]

Ancho de la imagen.

Ipp8u* pt::Image::mp_pixels [private]

Puntero a la imagen.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**image.h**
- eyeboard/src/**image.cpp**

7.2.23. Referencia de la Clase pt::math::Int2dPoint

`#include <maths.h>`

Métodos públicos

- **Int2dPoint** ()
- **Int2dPoint** (unsigned int _x, unsigned int _y)

Atributos públicos

- unsigned int **x**
- unsigned int **y**

7.2.23.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Contiene las coordenadas enteras de un punto bidimensional.

7.2.23.2. Documentación del constructor y destructor

pt::math::Int2dPoint::Int2dPoint () `[inline]`

Constructor vacío

pt::math::Int2dPoint::Int2dPoint (unsigned int *_x*, unsigned int *_y*) `[inline]`

Constructor parametrizado

Parámetros:

_x contiene la coordenada x

_y contiene la coordenada y

7.2.23.3. Documentación de los datos miembro

unsigned int pt::math::Int2dPoint::x

Contiene la coordenada x

unsigned int pt::math::Int2dPoint::y

Contiene la coordenada y

La documentación para esta clase fué generada a partir del siguiente archivo:

- eyeboard/src/**maths.h**

7.2.24. Referencia de la Clase pt::math::Kriging

`#include <kriging.h>`

Tipos públicos

- enum **EKrigingModel** { **LinearKriging**, **SphericalKriging**, **Exponential-Kriging**, **GaussianKriging** }

Métodos públicos

- **Kriging** (int n)
- ∼**Kriging** ()
- void **setKrigingModel** (**EKrigingModel** model)
- void **setDistance** (float dist)
- void **addPoint** (**Real2dPoint** point, **Real2dPoint** value)
- void **addPoint** (IppiPoint point, IppiPoint value)
- **Real2dPoint getInterpolatedValue** (**Real2dPoint** point)
- void **reset** ()
- unsigned int **getNumPoints** ()

Métodos privados

- float **calculateGamma** (float h)
- float **calculateDist** (**Real2dPoint** a, **Real2dPoint** b)
- void **printMatrices** ()
- void **printError** (IppStatus s)

Atributos privados

- unsigned int **numSamples**
- unsigned int **widthheight**
- std::vector< **SPointValue** > **variogram**

- **EKrigingModel kriging_model**
- float **distance**
- Ipp32f * **dist_xx**
- Ipp32f * **weights**
- Ipp32f * **dist_xp**
- Ipp32f * **aux_xx**
- Ipp32f * **buffer**
- bool **dist_xx_calculated**

Clases

- struct **SPointValue**

7.2.24.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Tareas Pendientes

debería estar creada con templates de tal forma que pudiese realizar interpolaciones entre dos espacios de dimensiones cualesquiera.

Realiza un kriging en funcion de una serie de puntos. **Kriging**(p. 163) es un método de interpolación geoestadística conocido como interpolación óptima. Esta clase recibe puntos en 2d y sus valores asociados como coordenadas 2d. Realiza una interpolación y devuelve el valor 2d asociado a la posicion requerida.

Para realizar un kriging es necesario ajustarse a un modelo, existen cuatro modelos comunes:

- lineal
- esférico
- exponencial

- gaussiano.

Todos los anteriores métodos están implementados.

Es necesario definir una variable distancia que representa la distancia a partir de la cual no se considerarán los valores de los puntos en la realizacion del kriging.

Necesita IPP.

7.2.24.2. Documentación de las enumeraciones miembro de la clase

enum pt::math::Kriging::EKrigingModel

Representa los tipos de kriging posibles.

Valores de la enumeración:

LinearKriging **Kriging**(p. 163) lineal: el peso asociado es la distancia

SphericalKriging **Kriging**(p. 163) esférico: el peso es 1.5∗h - 0.5(h/a)^3

ExponentialKriging **Kriging**(p. 163) exponencial: el peso es 1 - e^(-h/a)

GaussianKriging **Kriging**(p. 163) gausiano: el peso es 1 - e^-(-h/a)^2

7.2.24.3. Documentación del constructor y destructor

pt::math::Kriging::Kriging (int *n*)

Constructor.

Parámetros:

n representa el numero de puntos que tendrá.

pt::math::Kriging::∼Kriging ()

Destructor

7.2.24.4. Documentación de las funciones miembro

void pt::math::Kriging::addPoint (IppiPoint *point*, IppiPoint *value*)

Añade un punto al array de puntos del kriging

void pt::math::Kriging::addPoint (Real2dPoint *point*, Real2dPoint *value*)

Añade un punto al array de puntos del kriging

float pt::math::Kriging::calculateDist (Real2dPoint *a*, Real2dPoint *b*) `[private]`

Calcula la distancia entre dos puntos

float pt::math::Kriging::calculateGamma (float *h*) `[private]`

Calcula el valor gamma de un elemento en función de la distancia

Real2dPoint pt::math::Kriging::getInterpolatedValue (Real2dPoint *point*)

Realiza la interpolación y devuelve el valor estimado

unsigned int pt::math::Kriging::getNumPoints () `[inline]`

Devuelve el numero de puntos en el variograma

void pt::math::Kriging::printError (IppStatus *s*) `[private]`

Devuelve el error asociado a la ultima llamada a IPP

void pt::math::Kriging::printMatrices () `[private]`

Muestra por consola las matrices utilizadas durante el proceso

void pt::math::Kriging::reset () `[inline]`

Vacia el array de puntos

void pt::math::Kriging::setDistance (float *dist*) `[inline]`

Establece la distancia (parametro del kriging)

void pt::math::Kriging::setKrigingModel (EKrigingModel *model*) `[inline]`

Establece el model de kriging a realizar

7.2.24.5. Documentación de los datos miembro

Ipp32f* pt::math::Kriging::aux_xx `[private]`

Es una matriz auxiliar para intercambio de datos

Ipp32f* pt::math::Kriging::buffer [private]

Area de trabajo necesitada por IPP para realizar los calculos

Ipp32f* pt::math::Kriging::dist_xp [private]

Contiene un vector con las distancias del punto a calcular y todas las muestras

Ipp32f* pt::math::Kriging::dist_xx [private]

Almacena una matriz con las distancias de todos los puntos entre si

bool pt::math::Kriging::dist_xx_calculated [private]

true si la matriz dist_xx ya ha sido calculada

float pt::math::Kriging::distance [private]

Almacena la distancia, un parametro necesario para realizar el kriging

EKrigingModel pt::math::Kriging::kriging_model [private]

Almacena el modelo de kriging

unsigned int pt::math::Kriging::numSamples [private]

Numero de muestras del kriging (Ej: matriz de calibrado 3x3= 9 nueve muestras

std::vector<SPointValue> pt::math::Kriging::variogram [private]

Almacena los puntos con sus muestras

Ipp32f* pt::math::Kriging::weights [private]

Almacena un vector con los pesos de la interpolación

unsigned int pt::math::Kriging::widthheight [private]

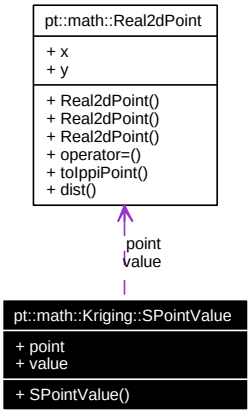
Ancho de las matrices del metodo

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**kriging.h**
- eyeboard/src/**kriging.cpp**

7.2.25. Referencia de la Estructura pt::math::Kriging::SPointValue

Diagrama de colaboración para pt::math::Kriging::SPointValue:



Métodos públicos

- **SPointValue** (**Real2dPoint** p, **Real2dPoint** v)

Atributos públicos

- **Real2dPoint point**
- **Real2dPoint value**

7.2.25.1. Descripción detallada

Almacena un punto y su valor 2d asociado

7.2.25.2. Documentación del constructor y destructor

pt::math::Kriging::SPointValue::SPointValue (Real2dPoint *p*, Real2dPoint *v*) `[inline]`

Constructor

7.2.25.3. Documentación de los datos miembro

Real2dPoint pt::math::Kriging::SPointValue::point

Contiene las coordenadas del punto

Real2dPoint pt::math::Kriging::SPointValue::value

Contiene el valor de la muestra en ese punto

Nota:

La muestra es un espacio bidimensional

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**kriging.h**

7.2.26. Referencia de la Clase pt::Logger

`#include <logger.h>`

Diagrama de herencias de pt::Logger

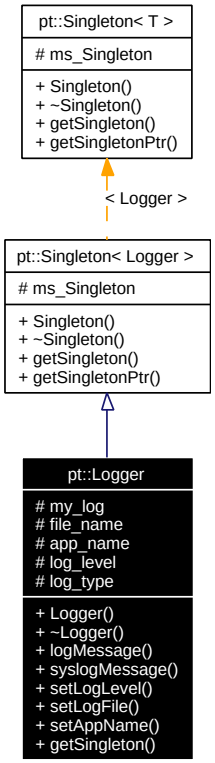
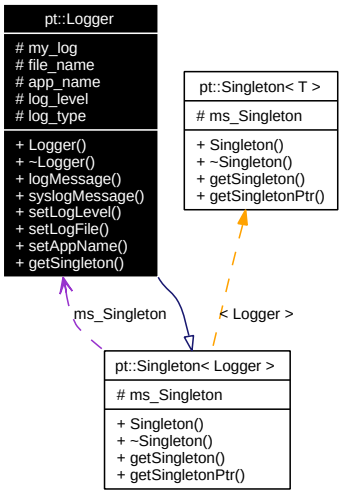


Diagrama de colaboración para pt::Logger:



Métodos públicos

- **Logger** (const std::string &ap_n, const std::string &name="log.txt", int **log_type**=LT_USER, int level=LL_WARNING)
- virtual ~**Logger** ()
- void **logMessage** (const std::string &st, int level=LL_WARNING, bool log_time=true)
- void **syslogMessage** (const std::string &log, int level=LL_WARNING)
- void **setLogLevel** (int new_level)
- void **setLogFile** (const std::string &new_file)
- void **setAppName** (const std::string &ana)

Métodos públicos estáticos

- static **Logger** & **getSingleton** ()

Atributos protegidos

- std::ofstream **my_log**
- std::string **file_name**
- std::string **app_name**
- int **log_level**
- int **log_type**

7.2.26.1. Descripción detallada

Clase creada para loggear todos los eventos de una aplicacion "UNIX-like". Existen dos tipos de logs: aquellos importantes para el sistema (sys log) que se guardan en disco a traves del demonio syslogd, y aquellos particulares de la aplicacion (file log) que se escriben en un fichero especificado.

Se consideran eventos importantes para el sistema:

- "daemon started"
- "critical error"
- ...

Logger(p. 169) puede ser utilizado para escribir en fichero cualquier tipo de informacion util para una determinada aplicacion, siempre que sea en modo log (append).

La distincion final entre el tipo de log (syslog o filelog) a utilizar sera decidida por el usuario de esta clase, aunque se recomienda utilizar syslog solo en los casos en los que contenga informacion relevante para el sistema.

La implementacion de esta clase busca seguridad y velocidad, por ello no abre y cierra en cada evento de log el fichero en el que se guardaran los mensajes, sino que utiliza un "flush" consiguiendo asi que se guarden los cambios en disco.

Esta clase es instanciable una sola vez (singleton)

7.2.26.2. Documentación del constructor y destructor

pt::Logger::Logger (const std::string & *ap_n*, const std::string & *name* = `"log.txt"`, int *log_type* = `LT_USER`, int *level* = `LL_WARNING`)

loggeamos lo que tenga importancia mayor o igual que level

pt::Logger::~Logger () `[virtual]`

Destructor

7.2.26.3. Documentación de las funciones miembro

Logger & pt::Logger::getSingleton () `[static]`

Obtiene el singleton

Reimplementado de **pt::Singleton< Logger >** (p. 183).

void pt::Logger::logMessage (const std::string & *st*, int *level* = `LL_WARNING`, bool *log_time* = `true`)

Funcion para escribir file-logs, por defecto importancia normal

void pt::Logger::setAppName (const std::string & *ana*) `[inline]`

Actualiza el nombre de la aplicacion

void pt::Logger::setLogFile (const std::string & *new_file*)

Actualiza el nombre del fichero en el que se logea

void pt::Logger::setLogLevel (int *new_level*) `[inline]`

Establece el nivel de log

void pt::Logger::syslogMessage (const std::string & *log*, int *level* = `LL_WARNING`)

Metodo para loggear en el sistema

7.2.26.4. Documentación de los datos miembro

std::string pt::Logger::app_name `[protected]`

Nombre de la aplicacion

std::string pt::Logger::file_name `[protected]`

Nombre del fichero de logs

int pt::Logger::log_level `[protected]`

Nivel de log

int pt::Logger::log_type `[protected]`

Tipo de log

std::ofstream pt::Logger::my_log `[protected]`

Ofstream de escritura

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**logger.h**

- eyeboard/src/**logger.cpp**

7.2.27. Referencia de la Clase pt::Profiler

`#include <profiler.h>`

Diagrama de herencias de pt::Profiler

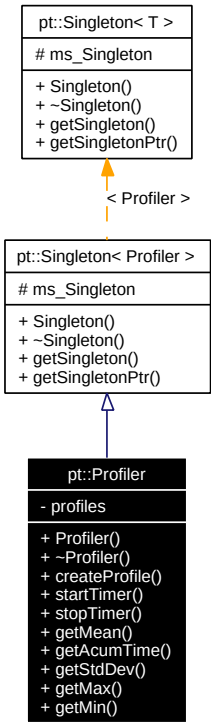
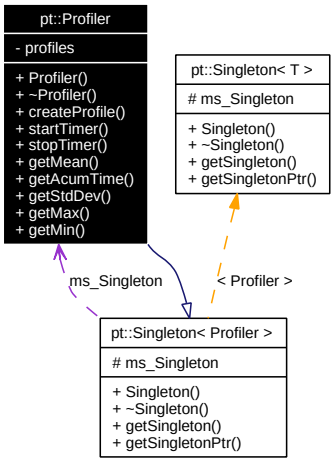


Diagrama de colaboración para pt::Profiler:



Métodos públicos

- **Profiler** ()

  Constructor.

- ∼**Profiler** ()

  Destructor.

- int **createProfile** (std::string name)

  Crea el profile y devuelve el identificador.

- void **startTimer** (int)

  Inicia el contador de tiempo.

- void **stopTimer** (int)

  Para el contador de tiempo y lo almacena modificando la media.

- float **getMean** (int) const

  Devuelve la media de tiempo.

- double **getAcumTime** (int) const

  Devuelve el tiempo total acumulado.

- float **getStdDev** (int) const

  Desviación estándar.

- float **getMax** (int) const

Máximo.

- float **getMin** (int) const

  Mínimo.

Atributos privados

- std::vector< **CProfile** * > **profiles**

  Vector con los profiles.

Clases

- class **CProfile**

7.2.27.1. Descripción detallada

Almacena tiempos de ejecucion de métodos, funciones, bucles... Extrae estadisticas de los tiempos.

Depende directamente de STL e indirectamente (**CProfile**(p. 176)) de IPP

7.2.27.2. Documentación del constructor y destructor

pt::Profiler::Profiler ()

Constructor.

pt::Profiler::~Profiler ()

Destructor.

7.2.27.3. Documentación de las funciones miembro

int pt::Profiler::createProfile (std::string *name*)

Crea el profile y devuelve el identificador.

double pt::Profiler::getAcumTime (int) const

Devuelve el tiempo total acumulado.

float pt::Profiler::getMax (int) const

Máximo.

float pt::Profiler::getMean (int) const

Devuelve la media de tiempo.

float pt::Profiler::getMin (int) const

Mínimo.

float pt::Profiler::getStdDev (int) const

Desviación estándar.

void pt::Profiler::startTimer (int)

Inicia el contador de tiempo.

void pt::Profiler::stopTimer (int)

Para el contador de tiempo y lo almacena modificando la media.

7.2.27.4. Documentación de los datos miembro

std::vector<CProfile*> pt::Profiler::profiles `[private]`

Vector con los profiles.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**profiler.h**
- eyeboard/src/**profiler.cpp**

7.2.28. Referencia de la Clase pt::Profiler::CProfile

Métodos públicos

- **CProfile** (std::string name)

  Crea el profile con el nombre dado.

- ~**CProfile** ()
- void **startTimer** ()

Almacena el tiempo en el que se inicia el cronometro.

- void **stopTimer** ()

  Almacena el tiempo en el que se detiene el cronometro.

- float **getMean** () const

  Devuelve la media de los tiempos.

- double **getAcumTime** () const

  Devuelve el tiempo total acumulado.

- unsigned int **getNumTimes** () const

  Devuelve el numero de veces que se ha llamado.

- std::string **getName** () const

  Devuelve el nombre del profile.

Métodos privados

- void **addTime** (float time)

  Almacena el tiempo que se ha tardado en la muestra.

Atributos privados

- float **cur_val**

  Tiempo de media acumulado.

- double **acum_time**

  Tiempo total acumulado.

- unsigned int **num_times**

  Numero de veces que se ha almacenado.

- std::string **prof_name**

  Nombre del profile.

- vm_tick **time_start**

  Tiempo al inicio.

- vm_tick **time_end**

  Tiempo al final.

- vm_tick **freq**

  Frecuencia.

7.2.28.1. Descripción detallada

Almacena la media del tiempo.

7.2.28.2. Documentación del constructor y destructor

pt::Profiler::CProfile::CProfile (std::string *name*)

Crea el profile con el nombre dado.

pt::Profiler::CProfile::~CProfile ()

7.2.28.3. Documentación de las funciones miembro

void pt::Profiler::CProfile::addTime (float *time*) `[private]`

Almacena el tiempo que se ha tardado en la muestra.

double pt::Profiler::CProfile::getAcumTime () const `[inline]`

Devuelve el tiempo total acumulado.

float pt::Profiler::CProfile::getMean () const `[inline]`

Devuelve la media de los tiempos.

std::string pt::Profiler::CProfile::getName () const `[inline]`

Devuelve el nombre del profile.

unsigned int pt::Profiler::CProfile::getNumTimes () const `[inline]`

Devuelve el numero de veces que se ha llamado.

void pt::Profiler::CProfile::startTimer ()

Almacena el tiempo en el que se inicia el cronometro.

void pt::Profiler::CProfile::stopTimer ()

Almacena el tiempo en el que se detiene el cronometro.

7.2.28.4. Documentación de los datos miembro

double pt::Profiler::CProfile::acum_time `[private]`

Tiempo total acumulado.

float pt::Profiler::CProfile::cur_val `[private]`

Tiempo de media acumulado.

vm_tick pt::Profiler::CProfile::freq `[private]`

Frecuencia.

unsigned int pt::Profiler::CProfile::num_times `[private]`

Numero de veces que se ha almacenado.

std::string pt::Profiler::CProfile::prof_name `[private]`

Nombre del profile.

vm_tick pt::Profiler::CProfile::time_end `[private]`

Tiempo al final.

vm_tick pt::Profiler::CProfile::time_start `[private]`

Tiempo al inicio.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**profiler.h**
- eyeboard/src/**profiler.cpp**

7.2.29. Referencia de la Clase pt::math::Real2dPoint

`#include <maths.h>`

Métodos públicos

- **Real2dPoint** ()
- **Real2dPoint** (IppiPoint p)
- **Real2dPoint** (float _x, float _y)
- void **operator=** (const IppiPoint &p)
- IppiPoint **toIppiPoint** ()

Métodos públicos estáticos

- static float **dist** (**Real2dPoint** a, **Real2dPoint** b)

Atributos públicos

- float **x**
- float **y**

7.2.29.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Contiene las coordenadas reales de un punto bidimensional. Acepta construir por copia de un IppiPoint.

7.2.29.2. Documentación del constructor y destructor

pt::math::Real2dPoint::Real2dPoint () `[inline]`

Constructor vacío

pt::math::Real2dPoint::Real2dPoint (IppiPoint *p*) `[inline]`

Constructor copia de un IppiPoint

pt::math::Real2dPoint::Real2dPoint (float *_x*, float *_y*) `[inline]`

Constructor parametrizado

Parámetros:

- ***_x*** contiene la coordenada x
- ***_y*** contiene la coordenada y

7.2.29.3. Documentación de las funciones miembro

static float pt::math::Real2dPoint::dist (Real2dPoint *a*, Real2dPoint *b*) `[inline, static]`

Calcula la distancia euclídea entre dos puntos

void pt::math::Real2dPoint::operator= (const IppiPoint & *p*) `[inline]`

Obtiene los valores de las coordenadas por un IppiPoint

IppiPoint pt::math::Real2dPoint::toIppiPoint () `[inline]`

Transforma el **Real2dPoint**(p. 180) en un IppiPoint, redondeando el resultado

7.2.29.4. Documentación de los datos miembro

float pt::math::Real2dPoint::x

Contiene la coordenada x

float pt::math::Real2dPoint::y

Contiene la coordenada y

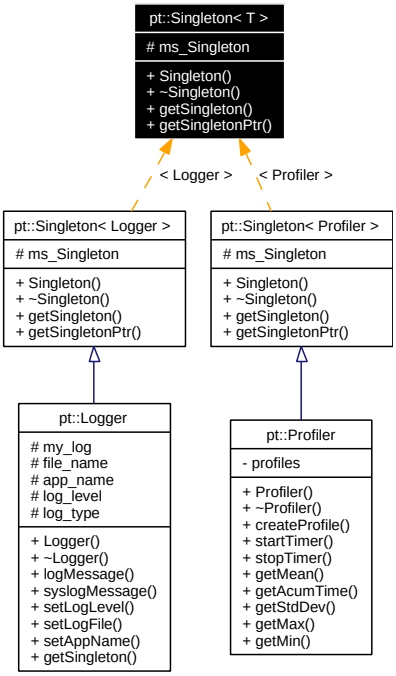
La documentación para esta clase fué generada a partir del siguiente archivo:

- eyeboard/src/**maths.h**

7.2.30. Referencia de la Clase pt::Singleton< T >

`#include <singleton.h>`

Diagrama de herencias de pt::Singleton< T >



Métodos públicos

- **Singleton** ()
- ∼**Singleton** (void)

Métodos públicos estáticos

- static T & **getSingleton** (void)
- static T ∗ **getSingletonPtr** (void)

Atributos protegidos estáticos

- static T ∗ **ms_Singleton** = 0

template<typename T> class pt::Singleton< T >

7.2.30.1. Documentación del constructor y destructor

template<typename T> pt::Singleton< T >::Singleton () `[inline]`

Constructor: assert if esta clase ya esta instanciada, de otra forma, inicia el singleton

template<typename T> pt::Singleton< T >::~Singleton (void) `[inline]`

Desvincula el singleton

7.2.30.2. Documentación de las funciones miembro

template<typename T> static T& pt::Singleton< T >::getSingleton (void) `[inline, static]`

Devuelve una referencia al singleton en cuestion

Reimplementado en **pt::Logger** (p. 171).

template<typename T> static T* pt::Singleton< T >::getSingletonPtr (void) `[inline, static]`

Devuelve un puntero al singleton en cuestion

7.2.30.3. Documentación de los datos miembro

template<typename T> T * pt::Singleton< T >::ms_Singleton = 0 `[static, protected]`

Puntero estatico al singleton

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**singleton.h**
- eyeboard/src/**singleton.cpp**

7.2.31. Referencia de la Estructura pt::SPixel

`#include <eyeboard.h>`

Métodos públicos

- **SPixel** (int _r, int _g, int _b)

Atributos públicos

- uchar **b**
- uchar **g**
- uchar **r**
- uchar **alfa**

7.2.31.1. Descripción detallada

struct **SPixel**(p. 183). Representa el valor en BGR_ALPHA de cada uno de los pixeles de la imagen. Utilizado para acceder directamente a los datos de la imagen mediante uint** jumpTable().

7.2.31.2. Documentación del constructor y destructor

pt::SPixel::SPixel (int *_r*, int *_g*, int *_b*) `[inline]`

7.2.31.3. Documentación de los datos miembro

uchar pt::SPixel::alfa

uchar pt::SPixel::b

uchar pt::SPixel::g

uchar pt::SPixel::r

La documentación para esta estructura fué generada a partir del siguiente archivo:

- eyeboard/src/**eyeboard.h**

7.2.32. Referencia de la Clase pt::renderers::VideoWriter

`#include <videowriter.h>`

Tipos públicos

- enum **EPixelFormat** {

  PIX_FMT_NONE = -1, **PIX_FMT_YUV420P**, **PIX_FMT_YUV422**, **PIX_FMT_RGB24**,

  PIX_FMT_BGR24, **PIX_FMT_YUV422P**, **PIX_FMT_YUV444P**, **PIX_FMT_RGBA32**,

  PIX_FMT_YUV410P, **PIX_FMT_YUV411P**, **PIX_FMT_RGB565**, **PIX_FMT_RGB555**,

  PIX_FMT_GRAY8, **PIX_FMT_MONOWHITE**, **PIX_FMT_MONOBLACK**, **PIX_FMT_PAL8**,

  PIX_FMT_YUVJ420P, **PIX_FMT_YUVJ422P**, **PIX_FMT_YUVJ444P**, **PIX_FMT_XVMC_MPEG2_MC**,

  PIX_FMT_XVMC_MPEG2_IDCT, **PIX_FMT_UYVY422**, **PIX_FMT_UYVY411**, **PIX_FMT_NB** }
- enum **ECodecID** {

  CODEC_ID_NONE, **CODEC_ID_MPEG1VIDEO**, **CODEC_ID_MPEG2VIDEO**, **CODEC_ID_MPEG2VIDEO_XVMC**,

  CODEC_ID_H261, **CODEC_ID_H263**, **CODEC_ID_RV10**, **CODEC_ID_RV20**,

  CODEC_ID_MJPEG, **CODEC_ID_MJPEGB**, **CODEC_ID_LJPEG**, **CODEC_ID_SP5X**,

  CODEC_ID_JPEGLS, **CODEC_ID_MPEG4**, **CODEC_ID_RAWVIDEO**, **CODEC_ID_MSMPEG4V1**,

  CODEC_ID_MSMPEG4V2, **CODEC_ID_MSMPEG4V3**, **CODEC_ID_WMV1**, **CODEC_ID_WMV2**,

  CODEC_ID_H263P, **CODEC_ID_H263I**, **CODEC_ID_FLV1**, **CODEC_ID_SVQ1**,

  CODEC_ID_SVQ3, **CODEC_ID_DVVIDEO**, **CODEC_ID_HUFFYUV**, **CODEC_ID_CYUV**,

  CODEC_ID_H264, **CODEC_ID_INDEO3**, **CODEC_ID_VP3**, **CODEC_ID_THEORA**,

  CODEC_ID_ASV1, **CODEC_ID_ASV2**, **CODEC_ID_FFV1**, **CODEC_ID_4XM**,

  CODEC_ID_VCR1, **CODEC_ID_CLJR**, **CODEC_ID_MDEC**, **CODEC_ID_ROQ**,

  CODEC_ID_INTERPLAY_VIDEO, **CODEC_ID_XAN_WC3**, **CODEC_ID_XAN_WC4**, **CODEC_ID_RPZA**,

CODEC_ID_CINEPAK, **CODEC_ID_WS_VQA**, **CODEC_ID_-MSRLE**, **CODEC_ID_MSVIDEO1**,

CODEC_ID_IDCIN, **CODEC_ID_8BPS**, **CODEC_ID_SMC**, **CODEC_ID_FLIC**,

CODEC_ID_TRUEMOTION1, **CODEC_ID_VMDVIDEO**, **CODEC_-ID_MSZH**, **CODEC_ID_ZLIB**,

CODEC_ID_QTRLE, **CODEC_ID_SNOW**, **CODEC_ID_TSCC**, **CODEC_ID_ULTI**,

CODEC_ID_QDRAW, **CODEC_ID_VIXL**, **CODEC_ID_QPEG**, **CODEC_ID_XVID**,

CODEC_ID_PNG, **CODEC_ID_PPM**, **CODEC_ID_PBM**, **CODEC_-ID_PGM**,

CODEC_ID_PGMYUV, **CODEC_ID_PAM**, **CODEC_ID_FFVHUFF**, **CODEC_ID_RV30**,

CODEC_ID_RV40, **CODEC_ID_VC9**, **CODEC_ID_WMV3**, **CODEC_ID_LOCO**,

CODEC_ID_WNV1, **CODEC_ID_AASC**, **CODEC_ID_INDEO2**, **CODEC_ID_FRAPS**,

CODEC_ID_TRUEMOTION2, **CODEC_ID_BMP**, **CODEC_ID_-CSCD**, **CODEC_ID_MMVIDEO**,

CODEC_ID_ZMBV, **CODEC_ID_AVS**, **CODEC_ID_SMACKVIDEO**, **CODEC_ID_NUV**,

CODEC_ID_PCM_S16LE = 0x10000, **CODEC_ID_PCM_S16BE**, **CODEC_ID_PCM_U16LE**, **CODEC_ID_PCM_U16BE**,

CODEC_ID_PCM_S8, **CODEC_ID_PCM_U8**, **CODEC_ID_PCM_-MULAW**, **CODEC_ID_PCM_ALAW**,

CODEC_ID_PCM_S32LE, **CODEC_ID_PCM_S32BE**, **CODEC_ID_-PCM_U32LE**, **CODEC_ID_PCM_U32BE**,

CODEC_ID_PCM_S24LE, **CODEC_ID_PCM_S24BE**, **CODEC_ID_-PCM_U24LE**, **CODEC_ID_PCM_U24BE**,

CODEC_ID_PCM_S24DAUD, **CODEC_ID_ADPCM_IMA_QT** = 0x11000, **CODEC_ID_ADPCM_IMA_WAV**, **CODEC_ID_ADPCM_-IMA_DK3**,

CODEC_ID_ADPCM_IMA_DK4, **CODEC_ID_ADPCM_IMA_WS**, **CODEC_ID_ADPCM_IMA_SMJPEG**, **CODEC_ID_ADPCM_MS**,

CODEC_ID_ADPCM_4XM, **CODEC_ID_ADPCM_XA**, **CODEC_-ID_ADPCM_ADX**, **CODEC_ID_ADPCM_EA**,

CODEC_ID_ADPCM_G726, **CODEC_ID_ADPCM_CT**, **CODEC_-ID_ADPCM_SWF**, **CODEC_ID_ADPCM_YAMAHA**,

CODEC_ID_ADPCM_SBPRO_4, **CODEC_ID_ADPCM_SBPRO_3**, **CODEC_ID_ADPCM_SBPRO_2**, **CODEC_ID_AMR_NB** = 0x12000,

CODEC_ID_AMR_WB, **CODEC_ID_RA_144** = 0x13000, **CODEC_-ID_RA_288**, **CODEC_ID_ROQ_DPCM** = 0x14000,

CODEC_ID_INTERPLAY_DPCM, **CODEC_ID_XAN_DPCM**, **CODEC_ID_SOL_DPCM**, **CODEC_ID_MP2** = 0x15000,

CODEC_ID_MP3, **CODEC_ID_AAC**, **CODEC_ID_MPEG4AAC**, **CODEC_ID_AC3**,

CODEC_ID_DTS, **CODEC_ID_VORBIS**, **CODEC_ID_DVAUDIO**, **CODEC_ID_WMAV1**,

CODEC_ID_WMAV2, **CODEC_ID_MACE3**, **CODEC_ID_MACE6**, **CODEC_ID_VMDAUDIO**,

CODEC_ID_SONIC, **CODEC_ID_SONIC_LS**, **CODEC_ID_FLAC**, **CODEC_ID_MP3ADU**,

CODEC_ID_MP3ON4, **CODEC_ID_SHORTEN**, **CODEC_ID_ALAC**, **CODEC_ID_WESTWOOD_SND1**,

CODEC_ID_GSM, **CODEC_ID_QDM2**, **CODEC_ID_COOK**, **CODEC_ID_TRUESPEECH**,

CODEC_ID_TTA, **CODEC_ID_SMACKAUDIO**, **CODEC_ID_-OGGTHEORA** = 0x16000, **CODEC_ID_DVD_SUBTITLE** = 0x17000,

CODEC_ID_DVB_SUBTITLE, **CODEC_ID_MPEG2TS** = 0x20000 }

Métodos públicos

- **VideoWriter** ()
- ∼**VideoWriter** ()
- int **init** (std::string fileName, int **width**, int **height**, int **bpp**, int **fps**, **ECodec-ID** cdc=CODEC_ID_NONE)
- void **saveFrame** (char ∗)
- void **close** ()

Métodos privados

- void **open_video** ()
- AVFrame ∗ **alloc_picture** (int pix_fmt)
- AVStream ∗ **add_video_stream** (int codec_id)

Atributos privados

- std::string **file_name**
- int **width**
- int **height**
- int **bpp**
- PixelFormat **pixel_format**
- int **fps**
- AVOutputFormat * **fmt**
- AVFormatContext * **oc**
- AVStream * **video_st**
- AVFrame * **picture**
- AVFrame * **tmp_picture**
- uint8_t * **video_outbuf**
- int **frame_count**
- int **video_outbuf_size**
- bool **initted**

7.2.32.1. Descripción detallada

Autor:

Enrique Turégano basado en el ejemplo de grabación de ffmpeg: `http://cekirdek.pardus.org.tr/~ismail/ffmpeg-docs/output_-_example_8c-source.html`

Versión:

0.1

Fecha:

2006

Se encarga de guardar ficheros de video en cualquier formato y codec soportado. Para ello utiliza la libreria ffmpeg.

Necesita libavformat-dev y libavformat0d

7.2.32.2. Documentación de las enumeraciones miembro de la clase

enum pt::renderers::VideoWriter::ECodecID

Identificador del codec, consultar /usr/include/ffmpeg/avcodec.h

Valores de la enumeración:

CODEC_ID_NONE

CODEC_ID_MPEG1VIDEO

CODEC_ID_MPEG2VIDEO

CODEC_ID_MPEG2VIDEO_XVMC

CODEC_ID_H261

CODEC_ID_H263

CODEC_ID_RV10

CODEC_ID_RV20

CODEC_ID_MJPEG

CODEC_ID_MJPEGB

CODEC_ID_LJPEG

CODEC_ID_SP5X

CODEC_ID_JPEGLS

CODEC_ID_MPEG4

CODEC_ID_RAWVIDEO

CODEC_ID_MSMPEG4V1

CODEC_ID_MSMPEG4V2

CODEC_ID_MSMPEG4V3

CODEC_ID_WMV1

CODEC_ID_WMV2

CODEC_ID_H263P

CODEC_ID_H263I

CODEC_ID_FLV1

CODEC_ID_SVQ1

CODEC_ID_SVQ3

CODEC_ID_DVVIDEO

CODEC_ID_HUFFYUV

CODEC_ID_CYUV

CODEC_ID_H264

CODEC_ID_INDEO3

CODEC_ID_VP3

CODEC_ID_THEORA

CODEC_ID_ASV1

CODEC_ID_ASV2
CODEC_ID_FFV1
CODEC_ID_4XM
CODEC_ID_VCR1
CODEC_ID_CLJR
CODEC_ID_MDEC
CODEC_ID_ROQ
CODEC_ID_INTERPLAY_VIDEO
CODEC_ID_XAN_WC3
CODEC_ID_XAN_WC4
CODEC_ID_RPZA
CODEC_ID_CINEPAK
CODEC_ID_WS_VQA
CODEC_ID_MSRLE
CODEC_ID_MSVIDEO1
CODEC_ID_IDCIN
CODEC_ID_8BPS
CODEC_ID_SMC
CODEC_ID_FLIC
CODEC_ID_TRUEMOTION1
CODEC_ID_VMDVIDEO
CODEC_ID_MSZH
CODEC_ID_ZLIB
CODEC_ID_QTRLE
CODEC_ID_SNOW
CODEC_ID_TSCC
CODEC_ID_ULTI
CODEC_ID_QDRAW
CODEC_ID_VIXL
CODEC_ID_QPEG
CODEC_ID_XVID
CODEC_ID_PNG
CODEC_ID_PPM
CODEC_ID_PBM

CODEC_ID_PGM
CODEC_ID_PGMYUV
CODEC_ID_PAM
CODEC_ID_FFVHUFF
CODEC_ID_RV30
CODEC_ID_RV40
CODEC_ID_VC9
CODEC_ID_WMV3
CODEC_ID_LOCO
CODEC_ID_WNV1
CODEC_ID_AASC
CODEC_ID_INDEO2
CODEC_ID_FRAPS
CODEC_ID_TRUEMOTION2
CODEC_ID_BMP
CODEC_ID_CSCD
CODEC_ID_MMVIDEO
CODEC_ID_ZMBV
CODEC_ID_AVS
CODEC_ID_SMACKVIDEO
CODEC_ID_NUV
CODEC_ID_PCM_S16LE
CODEC_ID_PCM_S16BE
CODEC_ID_PCM_U16LE
CODEC_ID_PCM_U16BE
CODEC_ID_PCM_S8
CODEC_ID_PCM_U8
CODEC_ID_PCM_MULAW
CODEC_ID_PCM_ALAW
CODEC_ID_PCM_S32LE
CODEC_ID_PCM_S32BE
CODEC_ID_PCM_U32LE
CODEC_ID_PCM_U32BE
CODEC_ID_PCM_S24LE

CODEC_ID_PCM_S24BE
CODEC_ID_PCM_U24LE
CODEC_ID_PCM_U24BE
CODEC_ID_PCM_S24DAUD
CODEC_ID_ADPCM_IMA_QT
CODEC_ID_ADPCM_IMA_WAV
CODEC_ID_ADPCM_IMA_DK3
CODEC_ID_ADPCM_IMA_DK4
CODEC_ID_ADPCM_IMA_WS
CODEC_ID_ADPCM_IMA_SMJPEG
CODEC_ID_ADPCM_MS
CODEC_ID_ADPCM_4XM
CODEC_ID_ADPCM_XA
CODEC_ID_ADPCM_ADX
CODEC_ID_ADPCM_EA
CODEC_ID_ADPCM_G726
CODEC_ID_ADPCM_CT
CODEC_ID_ADPCM_SWF
CODEC_ID_ADPCM_YAMAHA
CODEC_ID_ADPCM_SBPRO_4
CODEC_ID_ADPCM_SBPRO_3
CODEC_ID_ADPCM_SBPRO_2
CODEC_ID_AMR_NB
CODEC_ID_AMR_WB
CODEC_ID_RA_144
CODEC_ID_RA_288
CODEC_ID_ROQ_DPCM
CODEC_ID_INTERPLAY_DPCM
CODEC_ID_XAN_DPCM
CODEC_ID_SOL_DPCM
CODEC_ID_MP2
CODEC_ID_MP3
CODEC_ID_AAC
CODEC_ID_MPEG4AAC

CODEC_ID_AC3
CODEC_ID_DTS
CODEC_ID_VORBIS
CODEC_ID_DVAUDIO
CODEC_ID_WMAV1
CODEC_ID_WMAV2
CODEC_ID_MACE3
CODEC_ID_MACE6
CODEC_ID_VMDAUDIO
CODEC_ID_SONIC
CODEC_ID_SONIC_LS
CODEC_ID_FLAC
CODEC_ID_MP3ADU
CODEC_ID_MP3ON4
CODEC_ID_SHORTEN
CODEC_ID_ALAC
CODEC_ID_WESTWOOD_SND1
CODEC_ID_GSM
CODEC_ID_QDM2
CODEC_ID_COOK
CODEC_ID_TRUESPEECH
CODEC_ID_TTA
CODEC_ID_SMACKAUDIO
CODEC_ID_OGGTHEORA
CODEC_ID_DVD_SUBTITLE
CODEC_ID_DVB_SUBTITLE
CODEC_ID_MPEG2TS

enum pt::renderers::VideoWriter::EPixelFormat

Formato de pixel, consultar /usr/include/ffmpeg/avcodec.h

Valores de la enumeración:

PIX_FMT_NONE
PIX_FMT_YUV420P Planar YUV 4:2:0 (1 Cr & Cb sample per 2x2 Y samples).

PIX_FMT_YUV422 Packed pixel, Y0 Cb Y1 Cr.

PIX_FMT_RGB24 Packed pixel, 3 bytes per pixel, RGBRGB...

PIX_FMT_BGR24 Packed pixel, 3 bytes per pixel, BGRBGR...

PIX_FMT_YUV422P Planar YUV 4:2:2 (1 Cr & Cb sample per 2x1 Y samples).

PIX_FMT_YUV444P Planar YUV 4:4:4 (1 Cr & Cb sample per 1x1 Y samples).

PIX_FMT_RGBA32 Packed pixel, 4 bytes per pixel, BGRABGRA..., stored in cpu endianness.

PIX_FMT_YUV410P Planar YUV 4:1:0 (1 Cr & Cb sample per 4x4 Y samples).

PIX_FMT_YUV411P Planar YUV 4:1:1 (1 Cr & Cb sample per 4x1 Y samples).

PIX_FMT_RGB565 always stored in cpu endianness

PIX_FMT_RGB555 always stored in cpu endianness, most significant bit to 1

PIX_FMT_GRAY8

PIX_FMT_MONOWHITE 0 is white

PIX_FMT_MONOBLACK 0 is black

PIX_FMT_PAL8 8 bit with RGBA palette

PIX_FMT_YUVJ420P Planar YUV 4:2:0 full scale (jpeg).

PIX_FMT_YUVJ422P Planar YUV 4:2:2 full scale (jpeg).

PIX_FMT_YUVJ444P Planar YUV 4:4:4 full scale (jpeg).

PIX_FMT_XVMC_MPEG2_MC XVideo Motion Acceleration via common packet passing(xvmc_render.h).

PIX_FMT_XVMC_MPEG2_IDCT

PIX_FMT_UYVY422 Packed pixel, Cb Y0 Cr Y1.

PIX_FMT_UYVY411 Packed pixel, Cb Y0 Y1 Cr Y2 Y3.

PIX_FMT_NB

7.2.32.3. Documentación del constructor y destructor

pt::renderers::VideoWriter::VideoWriter ()

Constructor

pt::renderers::VideoWriter::~VideoWriter ()

Destructor

7.2.32.4. Documentación de las funciones miembro

AVStream * pt::renderers::VideoWriter::add_video_stream (int *codec_id*) `[private]`

Añade un stream de video

AVFrame * pt::renderers::VideoWriter::alloc_picture (int *pix_fmt*) `[private]`

Calcula el espacio necesitado por AVFrame y lo reserva, en funcion de pix_fmt

void pt::renderers::VideoWriter::close ()

Cierra el fichero escribiendo todo lo que falte.

int pt::renderers::VideoWriter::init (std::string *fileName*, int *width*, int *height*, int *bpp*, int *fps*, ECodecID *cdc* = `CODEC_ID_NONE`**)**

Inicia la clase con los parametros requeridos

void pt::renderers::VideoWriter::open_video () `[private]`

Selecciona un codec de video adecuado, lo abre y reserva espacio para la imagen a guardar.

void pt::renderers::VideoWriter::saveFrame (char *)

Guarda un nuevo frame en el fichero

7.2.32.5. Documentación de los datos miembro

int pt::renderers::VideoWriter::bpp `[private]`

bits por pixel

std::string pt::renderers::VideoWriter::file_name `[private]`

Especifica el nombre del fichero en el que se guardara el video

AVOutputFormat* pt::renderers::VideoWriter::fmt `[private]`

int pt::renderers::VideoWriter::fps [private]

Frames por segundo

int pt::renderers::VideoWriter::frame_count [private]

int pt::renderers::VideoWriter::height [private]

Alto del video

bool pt::renderers::VideoWriter::initted [private]

True si se ha iniciado la clase.

AVFormatContext* pt::renderers::VideoWriter::oc [private]

AVFrame* pt::renderers::VideoWriter::picture [private]

PixelFormat pt::renderers::VideoWriter::pixel_format [private]

Formato de pixel, utilizado para los AVFrames

AVFrame * pt::renderers::VideoWriter::tmp_picture [private]

uint8_t* pt::renderers::VideoWriter::video_outbuf [private]

int pt::renderers::VideoWriter::video_outbuf_size [private]

AVStream* pt::renderers::VideoWriter::video_st [private]

int pt::renderers::VideoWriter::width [private]

Ancho del video

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**videowriter.h**
- eyeboard/src/**videowriter.cpp**

7.2.33. Referencia de la Clase pt::processing::WorldProcessing

`#include <worldprocessing.h>`

Diagrama de herencias de pt::processing::WorldProcessing

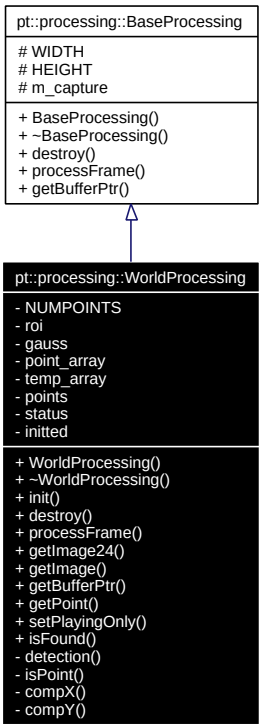
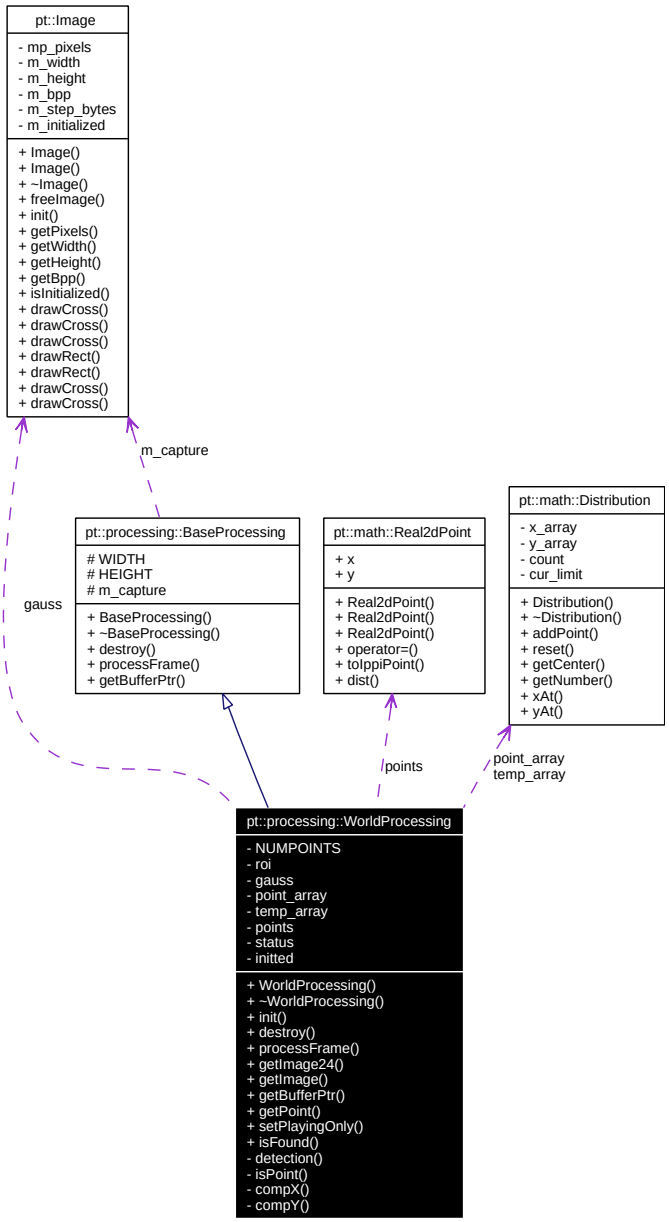


Diagrama de colaboración para pt::processing::WorldProcessing:



Métodos públicos

- **WorldProcessing** ()
- ∼**WorldProcessing** ()
- void **init** (int w, int h, int numpoints=9)
- void **destroy** ()
- void **processFrame** (Ipp8u ∗img)

- void **getImage24** (Ipp8u ∗i)
- void **getImage** (Ipp8u ∗i)
- const void ∗ **getBufferPtr** () const
- bool **getPoint** (int, **math::Real2dPoint** &)
- void **setPlayingOnly** ()
- bool **isFound** ()

Tipos privados

- enum **EStatus** { **finding**, **found**, **playing** }

Métodos privados

- void **detection** ()
- bool **isPoint** (int i, int j, Ipp8u centro)

Métodos privados estáticos

- static int **compX** (const void ∗a, const void ∗b)
- static int **compY** (const void ∗a, const void ∗b)

Atributos privados

- unsigned int **NUMPOINTS**
- IppiSize **roi**
- **Image gauss**
- **math::Distribution point_array**
- **math::Distribution temp_array**
- **math::Real2dPoint** ∗ **points**
- **EStatus status**
- bool **initted**

7.2.33.1. Descripción detallada

Autor:

Enrique Turégano

Versión:

0.1

Fecha:

2006

Se encarga de procesar el tren de imágenes que viene de la cámara que apunta al mundo (al frente del usuario) Su función principal es detectar los puntos de calibración.

necesita IPP

7.2.33.2. Documentación de las enumeraciones miembro de la clase

enum pt::processing::WorldProcessing::EStatus `[private]`

Contiene el estado en el que se encuentra la clase, el estado define las acciones que ha de realizar sobre la imagen.

Valores de la enumeración:

finding Los puntos de calibrado no se han encontrado en el ultimo frame, pero se siguen buscando

found Los puntos de calibrado se encontraron en el último frame

playing Ya no es necesario buscar los puntos de calibrado (Ya se ha calibrado el sistema)

7.2.33.3. Documentación del constructor y destructor

pt::processing::WorldProcessing::WorldProcessing ()

Constructor

pt::processing::WorldProcessing::∼WorldProcessing ()

Destructor

7.2.33.4. Documentación de las funciones miembro

static int pt::processing::WorldProcessing::compX (const void ∗ *a*, const void ∗ *b*) `[inline, static, private]`

Compara la coordenada x de dos Real2dPoint

static int pt::processing::WorldProcessing::compY (const void ∗ *a*, const void ∗ *b*) `[inline, static, private]`

Compara la coordenada y de dos Real2dPoint

void pt::processing::WorldProcessing::destroy () `[virtual]`

Destruye las variables de la clase

Implementa **pt::processing::BaseProcessing** (p. 114).

void pt::processing::WorldProcessing::detection () `[private]`

Realiza la deteccion de los puntos

const void∗ pt::processing::WorldProcessing::getBufferPtr () const `[inline, virtual]`

Devuelve un puntero a la zona de memoria donde se encuentra la imagen procesada.

Implementa **pt::processing::BaseProcessing** (p. 114).

void pt::processing::WorldProcessing::getImage (Ipp8u ∗ *i*)

Devuelve la imagen a 8 bpp

void pt::processing::WorldProcessing::getImage24 (Ipp8u ∗ *i*)

Devuelve la imagen en RGB (aunque de color gris)

bool pt::processing::WorldProcessing::getPoint (int, math::Real2dPoint &)

Devuelve las coordenadas de un punto de calibracion detectado

void pt::processing::WorldProcessing::init (int *w*, int *h*, int *numpoints* = 9)

Inicia la clase en función del ancho y el alto, y el numero de puntos de la maya a detectar

bool pt::processing::WorldProcessing::isFound () `[inline]`

se han encontrado los puntos

bool pt::processing::WorldProcessing::isPoint (int *i*, int *j*, Ipp8u *centro*) `[private]`

Devuelve true si el punto (i, j) es un posible punto de calibración.

void pt::processing::WorldProcessing::processFrame (Ipp8u * *img*) `[virtual]`

Procesa un frame

Implementa **pt::processing::BaseProcessing** (p. 115).

void pt::processing::WorldProcessing::setPlayingOnly () `[inline]`

Establece status como playing (ya no hay que realizar deteccion ninguna)

7.2.33.5. Documentación de los datos miembro

Image pt::processing::WorldProcessing::gauss `[private]`

Contiene la imagen que se está procesando.

bool pt::processing::WorldProcessing::initted `[private]`

indica que la clase se ha iniciado

unsigned int pt::processing::WorldProcessing::NUMPOINTS `[private]`

Contiene el número de puntos que se han de buscar (puntos de calibración)

math::Distribution pt::processing::WorldProcessing::point_array `[private]`

Array de puntos de calibración

math::Real2dPoint* pt::processing::WorldProcessing::points `[private]`

Array final con los puntos detectados

IppiSize pt::processing::WorldProcessing::roi `[private]`

Region of interest: en este caso la region de interés es la imagen entera

EStatus pt::processing::WorldProcessing::status `[private]`

Estado en el que se encuentra la detección

math::Distribution pt::processing::WorldProcessing::temp_array `[private]`

Array temporal de puntos

La documentación para esta clase fué generada a partir de los siguientes archivos:

- eyeboard/src/**worldprocessing.h**
- eyeboard/src/**worldprocessing.cpp**

7.3. Documentación de Archivos

7.3.1. Referencia del Archivo eyeboard/src/avireader.cpp

```
#include "avireader.h"
```

Namespaces

- namespace **pt**
- namespace **pt::readers**

Definiciones

- #define **READ4CC**(x)
- #define **READSIZE**(x)
- #define **READRAW**(x) fread(static_cast<void∗>(&x), sizeof(x), 1, riff_file);

7.3.1.1. Documentación de las definiciones

#define READ4CC(x)

Valor:

```
fread(x,4, 1, riff_file);               \
                                                if(!strcmp(x,#x))       \
                                                        return -1;
```

Realiza la lectura de 4 caracteres

#define READRAW(x) fread(static_cast<void∗>(&x), sizeof(x), 1, riff_file);

Lee raw-mente lo que queramos

#define READSIZE(x)

Valor:

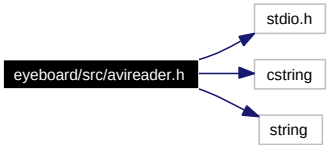
```
fread(static_cast<void*>(&x), 4, 1, riff_file); \
                                                if(x<0)                    \
                                                        return -1;
```

Lee un entero

7.3.2. Referencia del Archivo eyeboard/src/avireader.h

```
#include <stdio.h>
#include <cstring>
#include <string>
#include "basereader.h"
#include "logger.h"
```

Dependencia gráfica adjunta para avireader.h:



Namespaces

- namespace **pt**
- namespace **pt::readers**

Clases

- class **pt::readers::AviReader**
- struct **pt::readers::AviReader::SAviMainHeader**
- struct **pt::readers::AviReader::AviStreamHeader**
- struct **pt::readers::AviReader::STagRGBQUAD**
- struct **pt::readers::AviReader::STagBITMAPINFOHEADER**
- struct **pt::readers::AviReader::STagBITMAPINFO**
- struct **pt::readers::AviReader::SWaveFormatEx**
- struct **pt::readers::AviReader::SAviOldIndexEntry**
- struct **pt::readers::AviReader::SAviOldIndex**

Definiciones

- #define **DWORD** unsigned int
- #define **WORD** short
- #define **BYTE** char
- #define **LONG** long

7.3.2.1. Documentación de las definiciones

#define BYTE char

#define DWORD unsigned int

Por compatibilidad con los tipos de windows

#define LONG long

#define WORD short

7.3.3. Referencia del Archivo eyeboard/src/baseprocessing.cpp

```
#include "baseprocessing.h"
```

Namespaces

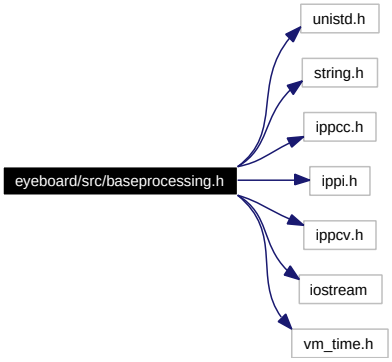
- namespace **pt**
- namespace **pt::processing**

7.3.4. Referencia del Archivo eyeboard/src/baseprocessing.h

```
#include <unistd.h>
#include <string.h>
#include <ippcc.h>
#include <ippi.h>
#include <ippcv.h>
#include <iostream>
#include <vm_time.h>
```

```
#include "image.h"
```

Dependencia gráfica adjunta para baseprocessing.h:



Namespaces

- namespace **pt**
- namespace **pt::processing**

Clases

- class **pt::processing::BaseProcessing**

7.3.5. Referencia del Archivo eyeboard/src/basereader.cpp

```
#include "basereader.h"
```

Namespaces

- namespace **pt**
- namespace **pt::readers**

7.3.6. Referencia del Archivo eyeboard/src/basereader.h

Namespaces

- namespace **pt**
- namespace **pt::readers**

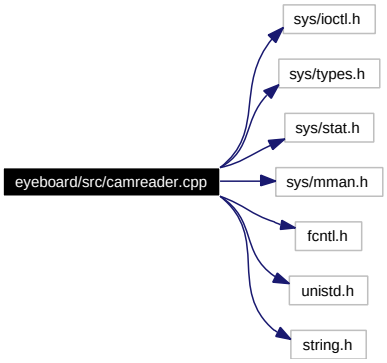
Clases

- class **pt::readers::BaseReader**

7.3.7. Referencia del Archivo eyeboard/src/camreader.cpp

```
#include "camreader.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
```

Dependencia gráfica adjunta para camreader.cpp:



Namespaces

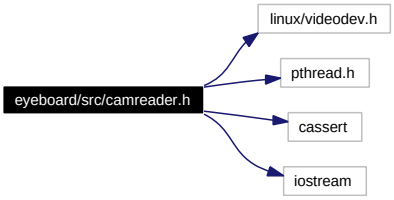
- namespace **pt**
- namespace **pt::readers**

7.3.8. Referencia del Archivo eyeboard/src/camreader.h

```
#include <linux/videodev.h>
#include <pthread.h>
#include <cassert>
```

```
#include <iostream>
```

Dependencia gráfica adjunta para camreader.h:



Namespaces

- namespace **pt**
- namespace **pt::readers**

Clases

- class **pt::readers::CamReader**

Definiciones

- #define **CAMREADER_H**
- #define **__STRICT_ANSI__**

7.3.8.1. Documentación de las definiciones

#define __STRICT_ANSI__

#define CAMREADER_H

7.3.9. Referencia del Archivo eyeboard/src/capture.cpp

```
#include "capture.h"
```

Namespaces

- namespace **pt**
- namespace **pt::readers**
- namespace **std**

Definiciones

- #define **CLEAR**(x) memset (&(x), 0, sizeof (x))

7.3.9.1. Documentación de las definiciones

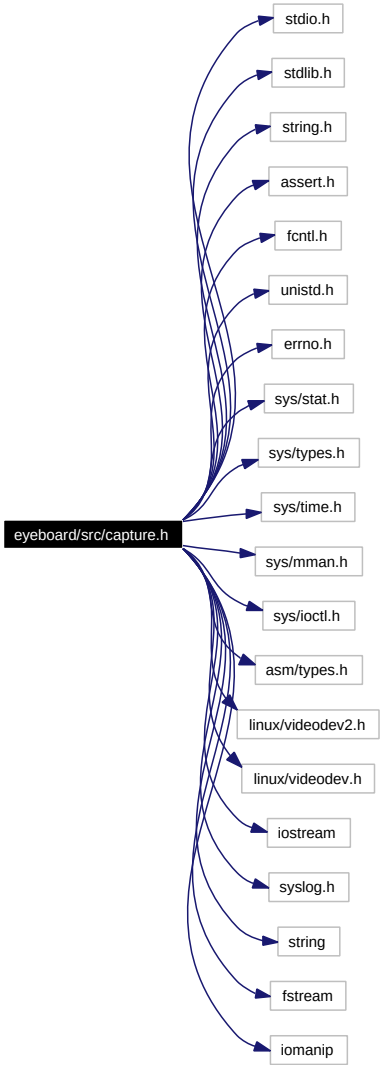
#define CLEAR(x) memset (&(x), 0, sizeof (x))

Pone a cero una zona de la memoria

7.3.10. Referencia del Archivo eyeboard/src/capture.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <asm/types.h>
#include <linux/videodev2.h>
#include <linux/videodev.h>
#include <iostream>
#include <syslog.h>
#include <string>
#include <fstream>
#include <iomanip>
```

Dependencia gráfica adjunta para capture.h:



Namespaces

- namespace **pt**
- namespace **pt::readers**

Clases

- class **pt::readers::Capture**
- struct **pt::readers::Capture::buffer**

Definiciones

- #define **__STRICT_ANSI__**

7.3.10.1. Documentación de las definiciones

#define __STRICT_ANSI__

7.3.11. Referencia del Archivo eyeboard/src/distribution.cpp

```
#include "distribution.h"
```

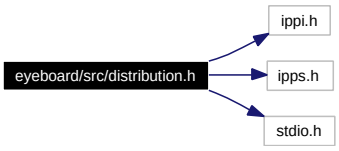
Namespaces

- namespace **pt**
- namespace **pt::math**

7.3.12. Referencia del Archivo eyeboard/src/distribution.h

```
#include <ippi.h>
#include <ipps.h>
#include <stdio.h>
#include "logger.h"
#include "maths.h"
```

Dependencia gráfica adjunta para distribution.h:



Namespaces

- namespace **pt**
- namespace **pt::math**

Clases

- class **pt::math::Distribution**

7.3.13. Referencia del Archivo eyeboard/src/eyeboard.cpp

```
#include "eyeboard.h"
#include "eyeboard.moc"
```

Dependencia gráfica adjunta para eyeboard.cpp:

Namespaces

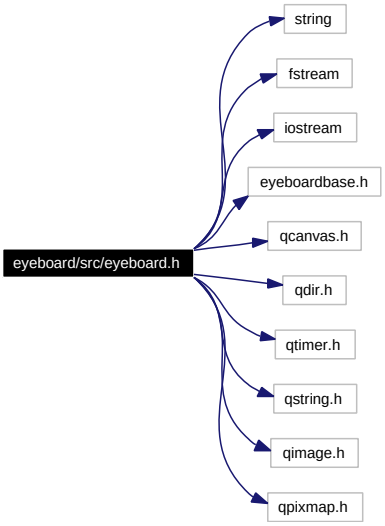
- namespace **pt**

7.3.14. Referencia del Archivo eyeboard/src/eyeboard.h

```
#include <string>
#include <fstream>
#include <iostream>
#include "eyeboardbase.h"
#include <qcanvas.h>
#include <qdir.h>
#include <qtimer.h>
#include <qstring.h>
#include <qimage.h>
#include <qpixmap.h>
#include "capture.h"
#include "worldprocessing.h"
#include "eyeprocessing.h"
#include "maths.h"
#include "kriging.h"
#include "videowriter.h"
```

```
#include "logger.h"
#include "glwindow.h"
```

Dependencia gráfica adjunta para eyeboard.h:



Namespaces

- namespace **pt**

Clases

- struct **pt::SPixel**
- class **pt::eyeboard**

  Application Main Window.

- struct **pt::eyeboard::SKey**

Definiciones

- #define **QT_CLEAN_NAMESPACE**

7.3.14.1. Documentación de las definiciones

#define QT_CLEAN_NAMESPACE

7.3.15. Referencia del Archivo eyeboard/src/eyeprocessing.cpp

```
#include "eyeprocessing.h"
```

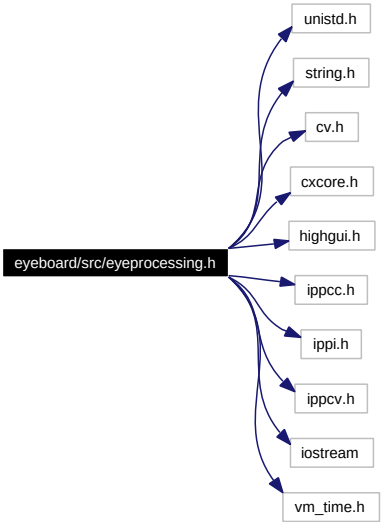
Namespaces

- namespace **pt**
- namespace **pt::processing**

7.3.16. Referencia del Archivo eyeboard/src/eyeprocessing.h

```
#include <unistd.h>
#include <string.h>
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"
#include <ippcc.h>
#include <ippi.h>
#include <ippcv.h>
#include <iostream>
#include <vm_time.h>
#include "distribution.h"
#include "baseprocessing.h"
#include "logger.h"
```

Dependencia gráfica adjunta para eyeprocessing.h:



Namespaces

- namespace **pt**
- namespace **pt::processing**

Clases

- class **pt::processing::EyeProcessing**

7.3.17. Referencia del Archivo eyeboard/src/glwindow.cpp

```
#include <unistd.h>
#include "glwindow.h"
```

Dependencia gráfica adjunta para glwindow.cpp:

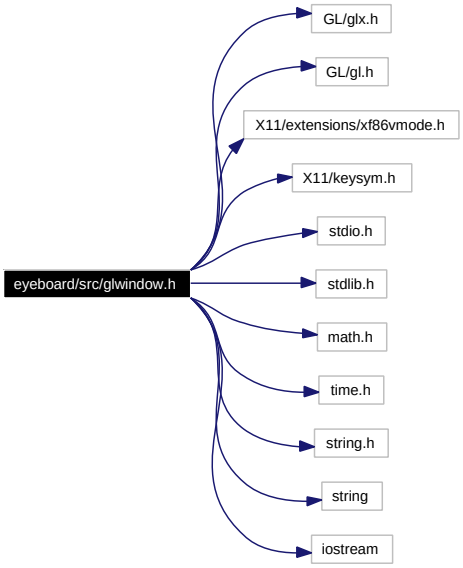
Namespaces

- namespace **pt**
- namespace **pt::renderers**

7.3.18. Referencia del Archivo eyeboard/src/glwindow.h

```
#include <GL/glx.h>
#include <GL/gl.h>
#include <X11/extensions/xf86vmode.h>
#include <X11/keysym.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <string>
#include <iostream>
#include "logger.h"
```

Dependencia gráfica adjunta para glwindow.h:



Namespaces

- namespace **pt**
- namespace **pt::renderers**

Clases

- class **pt::renderers::GLWindow**

7.3.19. Referencia del Archivo eyeboard/src/image.cpp

```
#include "image.h"
```

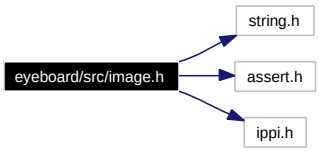
Namespaces

- namespace **pt**

7.3.20. Referencia del Archivo eyeboard/src/image.h

```
#include <string.h>
#include <assert.h>
#include <ippi.h>
#include "maths.h"
#include "logger.h"
```

Dependencia gráfica adjunta para image.h:



Namespaces

- namespace **pt**

Clases

- class **pt::Image**

7.3.21. Referencia del Archivo eyeboard/src/kriging.cpp

```
#include "kriging.h"
```

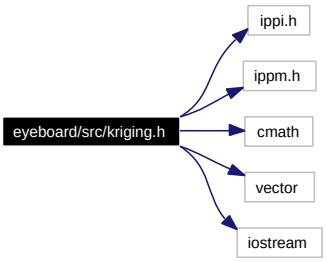
Namespaces

- namespace **pt**
- namespace **pt::math**

7.3.22. Referencia del Archivo eyeboard/src/kriging.h

```
#include <ippi.h>
#include <ippm.h>
#include <cmath>
#include <vector>
#include <iostream>
#include "maths.h"
#include "logger.h"
```

Dependencia gráfica adjunta para kriging.h:



Namespaces

- namespace **pt**
- namespace **pt::math**

Clases

- class **pt::math::Kriging**
- struct **pt::math::Kriging::SPointValue**

7.3.23. Referencia del Archivo eyeboard/src/logger.cpp

```
#include "logger.h"
```

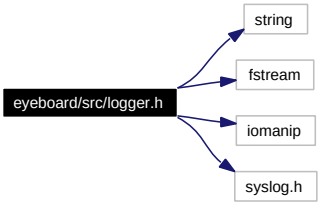
Namespaces

- namespace **pt**

7.3.24. Referencia del Archivo eyeboard/src/logger.h

```
#include <string>
#include <fstream>
#include <iomanip>
#include <syslog.h>
#include "singleton.h"
```

Dependencia gráfica adjunta para logger.h:



Namespaces

- namespace **pt**

Clases

- class **pt::Logger**

Definiciones

- #define **LL_EMERG** 0
- #define **LL_ALERT** 1
- #define **LL_CRIT** 2
- #define **LL_ERR** 3
- #define **LL_WARNING** 4
- #define **LL_NOTICE** 5
- #define **LL_INFO** 6
- #define **LL_DEBUG** 7

- #define **LT_KERN** (0<<3)
- #define **LT_USER** (1<<3)
- #define **LT_MAIL** (2<<3)
- #define **LT_DAEMON** (3<<3)
- #define **LT_AUTH** (4<<3)
- #define **LT_SYSLOG** (5<<3)
- #define **LT_LPR** (6<<3)
- #define **LT_NEWS** (7<<3)
- #define **LT_UUCP** (8<<3)
- #define **LT_CRON** (9<<3)
- #define **LT_AUTHPRIV** (10<<3)
- #define **LT_FTP** (11<<3)

7.3.24.1. Documentación de las definiciones

#define LL_ALERT 1

#define LL_CRIT 2

#define LL_DEBUG 7

#define LL_EMERG 0

Indica la importancia del mensaje a loggear. (Obtenidos de syslog.h) En orden de importancia decreciente:

#define LL_ERR 3

#define LL_INFO 6

#define LL_NOTICE 5

#define LL_WARNING 4

#define LT_AUTH (4<<3)

#define LT_AUTHPRIV (10<<3)

#define LT_CRON (9<<3)

#define LT_DAEMON (3<<3)

#define LT_FTP (11<<3)

#define LT_KERN (0<<3)

Indica el tipo de aplicacion que registra el log. (Obtenidos de syslog.h)

#define LT_LPR (6<<3)

#define LT_MAIL (2<<3)

#define LT_NEWS (7<<3)

#define LT_SYSLOG (5<<3)

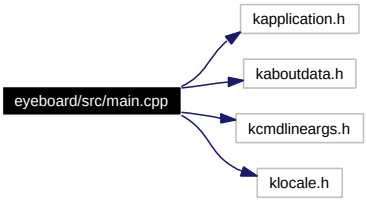
#define LT_USER (1<<3)

#define LT_UUCP (8<<3)

7.3.25. Referencia del Archivo eyeboard/src/main.cpp

```
#include "eyeboard.h"
#include <kapplication.h>
#include <kaboutdata.h>
#include <kcmdlineargs.h>
#include <klocale.h>
```

Dependencia gráfica adjunta para main.cpp:



Funciones

- int **main** (int argc, char **argv)

Variables

- static const char **description** [ ]
- static const char **version** [ ] = "0.1"
- static KCmdLineOptions **options** [ ]

7.3.25.1. Documentación de las funciones

int main (int *argc*, char ** *argv*)

Tareas Pendientes

do something with the command line args here

7.3.25.2. Documentación de las variables

const char description[ ] `[static]`

Valor inicial:

```
I18N_NOOP("A KDE KPart Application")
```

KCmdLineOptions options[ ] `[static]`

Valor inicial:

```
{

        KCmdLineLastOption
}
```

const char version[ ] = "0.1" `[static]`

7.3.26. Referencia del Archivo eyeboard/src/maths.cpp

```
#include "maths.h"
```

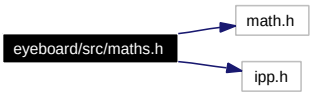
Namespaces

- namespace **pt**
- namespace **pt::math**

7.3.27. Referencia del Archivo eyeboard/src/maths.h

```
#include <math.h>
#include <ipp.h>
```

Dependencia gráfica adjunta para maths.h:



Namespaces

- namespace **pt**
- namespace **pt::math**

Clases

- class **pt::math::Int2dPoint**
- class **pt::math::Real2dPoint**
- class **pt::math::Definitions**

7.3.28. Referencia del Archivo eyeboard/src/profiler.cpp

```
#include "profiler.h"
```

Namespaces

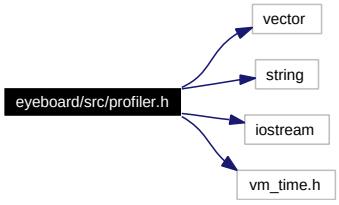
- namespace **pt**

7.3.29. Referencia del Archivo eyeboard/src/profiler.h

```
#include <vector>
#include <string>
```

```
#include <iostream>
#include <vm_time.h>
#include "singleton.h"
#include "logger.h"
```

Dependencia gráfica adjunta para profiler.h:



Namespaces

- namespace **pt**

Clases

- class **pt::Profiler**
- class **pt::Profiler::CProfile**

7.3.30. Referencia del Archivo eyeboard/src/singleton.cpp

```
#include "singleton.h"
```

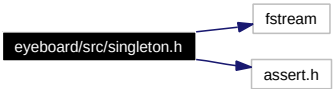
Namespaces

- namespace **pt**

7.3.31. Referencia del Archivo eyeboard/src/singleton.h

```
#include <fstream>
#include <assert.h>
```

Dependencia gráfica adjunta para singleton.h:



Namespaces

- namespace **pt**

Clases

- class **pt::Singleton**< **T** >

7.3.32. Referencia del Archivo eyeboard/src/videowriter.cpp

```
#include "videowriter.h"
```

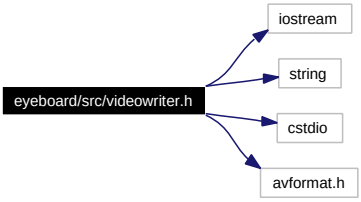
Namespaces

- namespace **pt**
- namespace **pt::renderers**

7.3.33. Referencia del Archivo eyeboard/src/videowriter.h

```
#include <iostream>
#include <string>
#include <cstdio>
#include <avformat.h>
```

Dependencia gráfica adjunta para videowriter.h:



Namespaces

- namespace **pt**
- namespace **pt::renderers**

Clases

- class **pt::renderers::VideoWriter**

7.3.34. Referencia del Archivo eyeboard/src/worldprocessing.cpp

```
#include "worldprocessing.h"
```

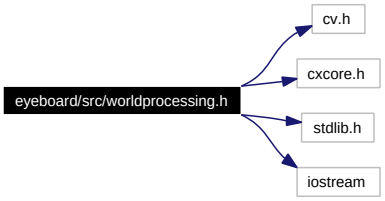
Namespaces

- namespace **pt**
- namespace **pt::processing**

7.3.35. Referencia del Archivo eyeboard/src/worldprocessing.h

```
#include "baseprocessing.h"
#include "logger.h"
#include "distribution.h"
#include "maths.h"
#include "cv.h"
#include "cxcore.h"
#include <stdlib.h>
#include <iostream>
```

Dependencia gráfica adjunta para worldprocessing.h:



Namespaces

- namespace **pt**
- namespace **pt::processing**

Clases

- class **pt::processing::WorldProcessing**

Parte IV

Anexos

Capítulo 8

Posibles ampliaciones y mejoras del sistema

El sistema desarrollado conforma una primera aproximación a un sistema de Eye Tracking. Durante su diseño se han tomado una serie de decisiones "controvertidas", debido a las limitaciones del hardware unas veces y a la complejidad del software otras. Por ello existen diversos aspectos que pueden llegar a ser reconsiderados y rediseñados en versiones futuras.

8.1. Ampliaciones y Mejoras del Hardware

- Calibración con Láser. Los puntos de calibración actuales necesitan estar situados sobre una superficie blanca situada delante del usuario y perpendicular a su vector de visión, pero Babcock et al. [23] utilizan otra solución quizás más cómoda para el usuario, utilizan un láser situado sobre el propio casco, que se encarga de mostrar 9 puntos de color rojo frente al usuario. De esta forma, la calibración podría ser realizada sobre casi cualquier superficie a cualquier distancia (hasta cierto límite). Esto eliminaría en parte el problema del paralaje, aunque probablemente no se pueda utilizar la superficie del monitor como plano de calibración. Además con este método, los puntos estarían situados siempre enfrente del usuario.

- Reconocimiento del Monitor con LEDS: si en lugar de utilizar los puntos de calibración como puntos de detección de monitor utilizásemos LEDS infrarrojos (similares o iguales a los que iluminan el ojo) situados en las esquinas del monitor, la detección de éstos sería inmediata, mucho más rápida e invariante por completo a la luz. El problema es que esto requeriría una instalación hardware extra e impediría reconocer el resto del mundo con esa misma cámara, la imagen se vería negra salvo cuatro puntos.

- Utilizar un Polhemus o un Flock of Birds. Supondría sin duda el mayor cambio que se puede realizar al proyecto. Ambos sistemas son utilizados para obtener la posición y orientación exacta en un mundo tridimensional. Implicaría el cambio del modelo ocular por otro mucho más preciso, para aprovechar la exactitud del posicionamiento tridimensional. El problema es que el coste asociado a este cambio es elevadísimo (equivaldría a 40 sistemas actuales).

- Camaras Wireless. Existen minicámaras espías inalámbricas de tamaño y coste similar a las cámaras usadas que permiten capturar imágenes sin necesidad de estar conectadas por cable al ordenador y al alimentador. Han de llevar una batería de 9 voltios para alimentarse. Las imágenes se transmiten por radio a un receptor conectado a la capturadora.

Figura 8.1: Cámara inalámbrica

  Esto incrementaría enormemente la movilidad del usuario dado que sustituyendo la alimentación de ambas cámaras y de los LEDS por una batería portatil, se podría mover sin estar "atado" a ningún lugar. La única limitación son los 50-200 metros de alcance del sistema de transmisión.

- Tracker binocular. El sistema propuesto está formado por una cámara que apunta a un solo ojo del usuario, lo cual supone la pérdida de profundidad de la escena por parte del programa. La distancia al objeto apuntado ha de ser inferida a partir de una imagen bidimensional. Un sistema que disponga de una cámara para cada ojo podría calcular la distancia al objeto en cuestión y podría, en función de ello, eliminar el tan odiado defecto del paralaje. Este nuevo sistema permitiría registrar los movimientos de vergencia también.

- Portátil en lugar de PC de sobremesa. Otra solución para incrementar la libertad del usuario es utilizar un ordenador portátil en lugar de un PC de sobremesa. Para ello sería necesario utilzar una tarjeta capturadora PCMCIA con doble entrada de video RCA. El usuario podría moverse con plena

libertad limitado únicamente por la duración de la batería del portátil. Esta solución es especialmente útil a la hora de saber el punto de mirada del usuario sin importar la interacción con el ordenador. Válida para realizar estudios psicológicos.

8.2. Ampliaciones y Mejoras del Software

Algunos de los cambios en el software estan subordinados a cambios en el hardware. Tal es el caso de la incorporación de una segunda cámara, que implicaría realizar otra captura, un procesamiento asociado a ésta; también sería necesario hacer un "metaprocesamiento" de las dos imágenes de las cámaras. Esto implicaría realizar una calibración en profundidad también.

- Resolución. Se podría mejorar la resolución del tracker mejorando el método de interpolación de forma que los resultados obtenidos sean más fieles a la realidad. El método actual de interpolación es un Kriging , llamado también estimación óptima. Kriging funciona bien para interpolar valores, pero su comportamiento dista mucho de ser óptimo para valores extrapolados.

- Diseño. Existen muchos aspectos en el diseño que se pueden reconsiderar. Debido en gran parte al carácter experimental de las aplicaciones desarrolladas, la estructura de clases y el diseño global de las mismas no han sido previstas desde un principio. Quizás con un conocimiento previo de los objetivos del proyecto se hubiese mejorado el diseño. Se podría crear por ejemplo una clase factory que se encarguase de proporcionar instancias de clases preparadas para capturar del dispositivo requerido.

- Filtro de Kalman: Esta basado en álgebra lineal y en los modelos ocultos de Markov. Es una herramienta recursiva muy importante en visión artificial que estima el estado de un sistema dinámico en función de una serie de medidas incompletas y ruidosas. Puede proporcionar información continuamente actualizada sobre la posición y velocidad de ciertos objetos de los que solo se tienen unas pocas medidas. En el tracking se realizan dos tipos de mediciones sujetas a error.

    - La primera consiste en averiguar la posición del centro de la pupila. El problema de intentar aplicar Kalman a la detección de la pupila es que los movimientos principales que rigen la posición de la misma (sacádicos) son de unos 1000° por segundo y suelen moverse unos 45°, debido a esto se necesitaría una velocidad de captura de unos 70 fps para empezar a considerar si merece la pena aplicar un filtro de Kalman. Por lo tanto Kalman no es aplicable en las condiciones actuales.

- La segunda medición consiste en detectar los puntos de calibración. En este caso si se puede aplicar filtrado de Kalman dado que la variación de la posición de dichos puntos en la imagen depende únicamente del movimiento de la cabeza del usuario. En este caso se mejoraría el proceso de detección prediciendo las futuras posiciones de los puntos y añadiría robustez al algoritmo.

Capítulo 9

Aplicaciones del sistema

La aplicación desarrollada permite escribir con sólo mirar un teclado mostrado en la pantalla. Sin embargo, para su realización, se ha necesitado calcular el punto de mirada, es decir, el lugar exacto al que el usuario está mirando. El framework principal con el que se ha creado la aplicación permite registrar secuencias de vídeo con el contenido de las imágenes de ambas cámaras (imagen 9.1). Debido a estas características, se puede observar el comportamiento humano ante diversos estímulos.

Las secuencias de vídeo permiten monitorizar los movimientos de los ojos, lo cual supone un método muy valioso para entender la percepción visual y la cognición. La habilidad de monitorizar los movimientos de los ojos de los observadores abre nuevos campos de investigación y nuevas aplicaciones:

- Utilización de ordenadores. El uso principal en el que nos hemos centrado es facilitar la tarea de escritura a personas con discapacidad, sin embargo, los usuarios sin problemas de movilidad se pueden ver beneficiados de este sistema:
  - Moviendo ventanas con ayuda del otro ojo. De esta forma el ojo derecho se utilizaría como "botón" de un ratón.
  - Se podría jugar a juegos de ordenador utilizando el sistema construido como ratón, lo cual incrementaría enormemente la velocidad con la que normalmente movemos el ratón. Podría suponer el fin de la diversión de los juegos en FPS.
- Monitorizar y corregir comportamientos en el deporte. Prácticamente cualquier deportista de cualquier disciplina deportiva se vería beneficiado del entrenamiento con un eyetracker. Gracias a esto se puede detectar comportamientos incorrectos, defectos, acciones innecesarias... En [28] Fairchild et al. estudian analíticamente el comportamiento de golfistas profesionales.

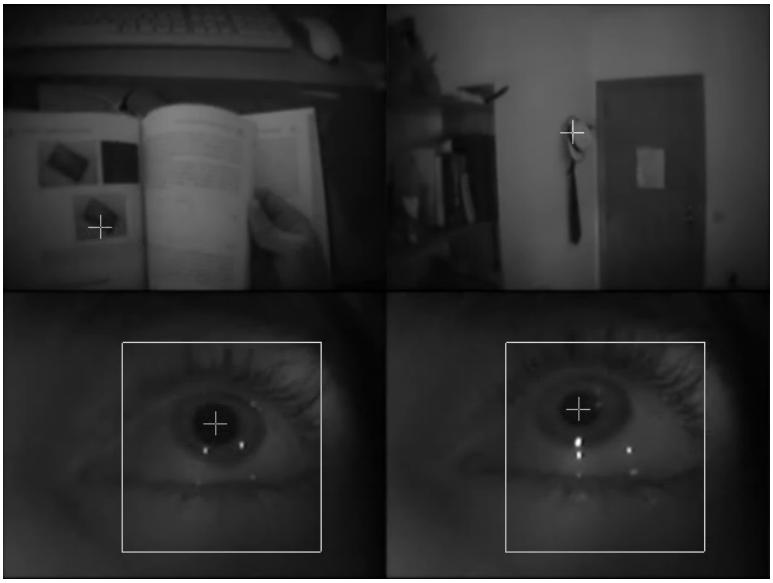


Figura 9.1: Ejemplo del cálculo del punto de mirada

Este sistema sería realmente útil en deportes como fórmula uno, rallies, tennis...

- Corregir comportamientos al aprender a conducir. Se puede observar el comportamiento de los conductores en prácticas con un eyetracker. De esta forma se pueden corregir comportamientos erróneos. También se puede observar el comportamiento de conductores profesionales para aprender de ellos en las autoescuelas.

- Estudios de marketing. Probalemente el campo más rentable de utilización de eyetrackers. Los diseñadores web podrían comprobar que partes son más atractivas de sus sitios, que partes son de especial interés, qué estaban leyendo los usuarios cuando abandonaron la página... El comportamiento de las personas ante anuncios de televisión resulta especialmente útil, dónde centran la atención las personas, qué anuncios gustan más... La forma de construir escaparates, las prendas de ropa que más gustan, la posición que ocupen los productos en las tiendas... todo esto podría verse beneficiado del uso de eyetrackers.

- Investigación en comportamiento. Para los psicólogos resulta de especial importancia estudiar y analizar las diferentes formas de comportamiento de las personas en función de su edad, de su situación social, de sus características. Se pueden realizar estudios sobre la percepción y cognición humanas.

- Investigación médica. Se pueden llegar a detectar síntomas de enfermedades,

como pérdidas de atención, problemas de visión...

Bibliografía

[1] Applied science laboratories - eye-tracking expertise. http://www.a-s-l.com/.

[2] Debian gnu/linux. http://www.debian.org/.

[3] Edición de vídeo. http://www.videoedicion.org/.

[4] Gesturepad de fingerworks. http://www.fingerworks.com/.

[5] Gnome on-screen keyboard. http://www.gok.ca/.

[6] Ipp: Intel performance primitives. http://www.intel.com/cd/software/products/asmo-na/eng/perflib/ipp/index.htm.

[7] Kde accessibility project. http://accessibility.kde.org/.

[8] Nuevos periféricos utilizados como ayudas técnicas para personas con discapacidad. Joaquín Fonoll Salvador.

[9] Opencv library en sourceforge. http://sourceforge.net/projects/opencvlibrary/.

[10] openeyes: the open source eyetracker. http://hcvl.hci.iastate.edu/cgi-bin/openEyes.cgi.

[11] Stl reference. http://www.sgi.com/tech/stl/.

[12] Thinking in c++. http://mindview.net/Books/TICPP/ThinkingInCPP2e.html.

[13] Wikipedia en castellano. http://es.wikipedia.org/.

[14] Wikipedia en inglés. http://en.wikipedia.org/.

[15] Navid Qaragozlou Ali Ajdari Rad, Karim Faez. Fast circle detection using gradient pair vectors, 2003.

[16] Gautam Barua. Optimal estimations of random fields using kriging. In *College on Soil Physics*, March 2003.

[17] R.H.S. Carpenter. *Movements of the Eyes*. Pion Limited, sencond edition, 1988.

[18] E. R. Daves. *Machine Vision Theory Algorithms Practicalities*. Elsevier, 2005 edition, third.

[19] Jason Babcock Dongheng Li and Derrick J. Parkhurst. openeyes: a low-cost head-mounted eye-tracking solution. In *The Human Computer Interaction Program*, 2006.

[20] Ruqin Zhang Dongheng Li. Eye typing using a low-cost desktop eye tracker. Final Report for ComS/HCI 575x.

[21] Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer, first edition, 2003.

[22] Jeff B. Pelz Jason S. Babcock. Building a lightweight eyetracker.

[23] Jeff B. Pelz Jason S. Babcock. The wearable eyetracker: A tool for the study of high-level visual tasks, February 2003.

[24] Jeff B. Pelz Jason S. Babcock, Marianne Lipps. How people look at pictures before, during and after scene capture: Buswell revisted.

[25] Jason Babcock Jeff B. Pelz, Roxanne Canosa. Extended tasks elicit complex eye movement patterns. In *Eye Tracking Research & Applications Symposium*, 2000.

[26] Roxanne Canosa Jeff B. Pelz. Oculomotor behavior and perceptual strategies in complex tasks. *Vision Research*, (41):3587–3596, 2001.

[27] Winfield D. Parkhurst D. J. Li, D. Starburst: A hybrid algorithm for video-based eye tracking combining feature-based and model-based approaches. In *IEEE Vision for Human-Computer Interaction Workshop*, 2006.

[28] Jason Babcock Mark D. Fairchild, Garrett M. Johnson and Jeff B. Pelz. Is your eye on the ball?: Eye tracking golfers while putting.

[29] Ulises Gamero Rodríguez. Astronomía asistida por ordenador. Master's thesis, Escuela Politécnica, Universidad de Extremadura, Junio 2002.

[30] Saúl Sánchez Vacas. Snakes: Reconstrucción 3d. Master's thesis, Escuela Politécnica, Universidad de Extremadura, Septiembre 2003.

[31] Li D. Babcock J. Parkhurst D. J. Winfield, D. Towards an open-hardware open-software toolkit for robust low-cost eye tracking in hci applications. Technical report, IEEE Vision for Human-Computer Interaction Workshop, 2006.

[32] Leung Yip R.K.K., Tam P.K.S. Modification of the hough transform for circles an ellipses detection using a 2-dimensional array. *Pattern recognition*, (25):1007–1022, 1992.

Índice alfabético

__STRICT_ANSI__
  camreader.h, 208
  capture.h, 211
~AviReader
  pt::readers::AviReader, 99
~BaseProcessing
  pt::processing::BaseProcessing, 114
~BaseReader
  pt::readers::BaseReader, 117
~CProfile
  pt::Profiler::CProfile, 178
~CamReader
  pt::readers::CamReader, 120
~Capture
  pt::readers::Capture, 125
~Distribution
  pt::math::Distribution, 130
~EyeProcessing
  pt::processing::EyeProcessing, 147
~GLWindow
  pt::renderers::GLWindow, 154
~Image
  pt::Image, 159
~Kriging
  pt::math::Kriging, 165
~Logger
  pt::Logger, 171
~Profiler
  pt::Profiler, 175
~Singleton
  pt::Singleton, 183
~VideoWriter
  pt::renderers::VideoWriter, 194
~WorldProcessing
  pt::processing::WorldProcessing, 200
~eyeboard
  pt::eyeboard, 134

aberraciones, 13
acum_time
  pt::Profiler::CProfile, 179
add_video_stream
  pt::renderers::VideoWriter, 195
addPoint
  pt::math::Distribution, 130
  pt::math::Kriging, 165
addTime
  pt::Profiler::CProfile, 178
Alexander Stepanov, 77
alfa
  pt::SPixel, 184
all_image
  pt::processing::EyeProcessing, 151
alloc_picture
  pt::renderers::VideoWriter, 195
app_name
  pt::Logger, 172
ARB, 77
atención, 7
  atención visual, 8
attr
  pt::renderers::GLWindow, 156
aux_xx
  pt::math::Kriging, 166
avi_main_header
  pt::readers::AviReader, 100
avi_old_index
  pt::readers::AviReader, 100
avi_stream_header

pt::readers::AviReader, 100
avi_writer
pt::eyeboard, 136
AviReader, 101
pt::readers::AviReader, 99
avireader.cpp
READ4CC, 203
READRAW, 203
READSIZE, 203
avireader.h
BYTE, 205
DWORD, 205
LONG, 205
WORD, 205

b
pt::SPixel, 184
babcock, 60
BaseProcessing
pt::processing::BaseProcessing, 114
BaseReader
pt::readers::BaseReader, 117
biBitCount
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biClrImportant
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biClrUsed
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biCompression
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biHeight
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biPlanes
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biSize
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biSizeImage
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
bit_map_info
pt::readers::AviReader, 100
biWidth
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biXPelsPerMeter
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
biYPelsPerMeter
pt::readers::AviReader::STag-BITMAPINFOHEADER, 109
bmiHeader
pt::readers::AviReader::STag-BITMAPINFO, 108
bottom
pt::readers::AviReader::Avi-StreamHeader, 103
BPP
pt::readers::Capture, 127
bpp
pt::renderers::GLWindow, 156
pt::renderers::VideoWriter, 195
bright_pupil
pt::processing::EyeProcessing, 151
Broadbent, 9
bttv, 83
buff_ptr
pt::readers::Capture, 127
buffer
pt::math::Kriging, 166

pt::readers::AviReader, 101
pt::readers::CamReader, 122
buffer_rgb24
pt::readers::AviReader, 101
buffers
pt::readers::Capture, 127
bufferSize
pt::readers::CamReader, 122
BYTE
avireader.h, 205

córnea, 14
cables RCA, 81
calcOpen
pt::processing::EyeProcessing, 147
calcROI
pt::processing::EyeProcessing, 148
calculateDist
pt::math::Kriging, 165
calculateGamma
pt::math::Kriging, 166
calibración, 88
caminos visuales, 12
campo visual, 19
CamReader
pt::readers::CamReader, 120
camreader.h
__STRICT_ANSI__, 208
CAMREADER_H, 208
CAMREADER_H
camreader.h, 208
canvas
pt::eyeboard, 136
canvas_kriging
pt::eyeboard, 136
canvas_text
pt::eyeboard, 136
canvas_view
pt::eyeboard, 136
Capture
pt::readers::Capture, 125
capture.cpp
CLEAR, 209
capture.h
__STRICT_ANSI__, 211
captureLoop
pt::readers::CamReader, 120
cb
pt::readers::AviReader::AviStreamHeader, 103
pt::readers::AviReader::SAviMainHeader, 105
pt::readers::AviReader::SAviOldIndex, 106
cbSize
pt::readers::AviReader::SWaveFormatEx, 111
CLEAR
capture.cpp, 209
close
pt::renderers::VideoWriter, 195
close_device
pt::readers::Capture, 126
closed_eye
pt::processing::EyeProcessing, 147
closed_eye_saved
pt::processing::EyeProcessing, 147
closeDevice
pt::readers::CamReader, 120
CODEC_ID_4XM
pt::renderers::VideoWriter, 190
CODEC_ID_8BPS
pt::renderers::VideoWriter, 190
CODEC_ID_AAC
pt::renderers::VideoWriter, 192
CODEC_ID_AASC
pt::renderers::VideoWriter, 191
CODEC_ID_AC3
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_4XM
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_ADX
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_CT
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_EA
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_G726

pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_DK3
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_DK4
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_QT
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_SMJPEG
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_WAV
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_IMA_WS
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_MS
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_SBPRO_2
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_SBPRO_3
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_SBPRO_4
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_SWF
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_XA
pt::renderers::VideoWriter, 192
CODEC_ID_ADPCM_YAMAHA
pt::renderers::VideoWriter, 192
CODEC_ID_ALAC
pt::renderers::VideoWriter, 193
CODEC_ID_AMR_NB
pt::renderers::VideoWriter, 192
CODEC_ID_AMR_WB
pt::renderers::VideoWriter, 192
CODEC_ID_ASV1
pt::renderers::VideoWriter, 189
CODEC_ID_ASV2
pt::renderers::VideoWriter, 189
CODEC_ID_AVS
pt::renderers::VideoWriter, 191
CODEC_ID_BMP
pt::renderers::VideoWriter, 191
CODEC_ID_CINEPAK
pt::renderers::VideoWriter, 190
CODEC_ID_CLJR
pt::renderers::VideoWriter, 190
CODEC_ID_COOK
pt::renderers::VideoWriter, 193
CODEC_ID_CSCD
pt::renderers::VideoWriter, 191
CODEC_ID_CYUV
pt::renderers::VideoWriter, 189
CODEC_ID_DTS
pt::renderers::VideoWriter, 193
CODEC_ID_DVAUDIO
pt::renderers::VideoWriter, 193
CODEC_ID_DVB_SUBTITLE
pt::renderers::VideoWriter, 193
CODEC_ID_DVD_SUBTITLE
pt::renderers::VideoWriter, 193
CODEC_ID_DVVIDEO
pt::renderers::VideoWriter, 189
CODEC_ID_FFV1
pt::renderers::VideoWriter, 190
CODEC_ID_FFVHUFF
pt::renderers::VideoWriter, 191
CODEC_ID_FLAC
pt::renderers::VideoWriter, 193
CODEC_ID_FLIC
pt::renderers::VideoWriter, 190
CODEC_ID_FLV1
pt::renderers::VideoWriter, 189
CODEC_ID_FRAPS
pt::renderers::VideoWriter, 191
CODEC_ID_GSM
pt::renderers::VideoWriter, 193
CODEC_ID_H261
pt::renderers::VideoWriter, 189
CODEC_ID_H263
pt::renderers::VideoWriter, 189
CODEC_ID_H263I
pt::renderers::VideoWriter, 189
CODEC_ID_H263P
pt::renderers::VideoWriter, 189
CODEC_ID_H264
pt::renderers::VideoWriter, 189
CODEC_ID_HUFFYUV
pt::renderers::VideoWriter, 189
CODEC_ID_IDCIN

pt::renderers::VideoWriter, 190
CODEC_ID_INDEO2
pt::renderers::VideoWriter, 191
CODEC_ID_INDEO3
pt::renderers::VideoWriter, 189
CODEC_ID_INTERPLAY_DPCM
pt::renderers::VideoWriter, 192
CODEC_ID_INTERPLAY_VIDEO
pt::renderers::VideoWriter, 190
CODEC_ID_JPEGLS
pt::renderers::VideoWriter, 189
CODEC_ID_LJPEG
pt::renderers::VideoWriter, 189
CODEC_ID_LOCO
pt::renderers::VideoWriter, 191
CODEC_ID_MACE3
pt::renderers::VideoWriter, 193
CODEC_ID_MACE6
pt::renderers::VideoWriter, 193
CODEC_ID_MDEC
pt::renderers::VideoWriter, 190
CODEC_ID_MJPEG
pt::renderers::VideoWriter, 189
CODEC_ID_MJPEGB
pt::renderers::VideoWriter, 189
CODEC_ID_MMVIDEO
pt::renderers::VideoWriter, 191
CODEC_ID_MP2
pt::renderers::VideoWriter, 192
CODEC_ID_MP3
pt::renderers::VideoWriter, 192
CODEC_ID_MP3ADU
pt::renderers::VideoWriter, 193
CODEC_ID_MP3ON4
pt::renderers::VideoWriter, 193
CODEC_ID_MPEG1VIDEO
pt::renderers::VideoWriter, 189
CODEC_ID_MPEG2TS
pt::renderers::VideoWriter, 193
CODEC_ID_MPEG2VIDEO
pt::renderers::VideoWriter, 189
CODEC_ID_MPEG2VIDEO_XVMC
pt::renderers::VideoWriter, 189
CODEC_ID_MPEG4
pt::renderers::VideoWriter, 189
CODEC_ID_MPEG4AAC
pt::renderers::VideoWriter, 192
CODEC_ID_MSMPEG4V1
pt::renderers::VideoWriter, 189
CODEC_ID_MSMPEG4V2
pt::renderers::VideoWriter, 189
CODEC_ID_MSMPEG4V3
pt::renderers::VideoWriter, 189
CODEC_ID_MSRLE
pt::renderers::VideoWriter, 190
CODEC_ID_MSVIDEO1
pt::renderers::VideoWriter, 190
CODEC_ID_MSZH
pt::renderers::VideoWriter, 190
CODEC_ID_NONE
pt::renderers::VideoWriter, 189
CODEC_ID_NUV
pt::renderers::VideoWriter, 191
CODEC_ID_OGGTHEORA
pt::renderers::VideoWriter, 193
CODEC_ID_PAM
pt::renderers::VideoWriter, 191
CODEC_ID_PBM
pt::renderers::VideoWriter, 190
CODEC_ID_PCM_ALAW
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_MULAW
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S16BE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S16LE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S24BE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S24DAUD
pt::renderers::VideoWriter, 192
CODEC_ID_PCM_S24LE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S32BE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S32LE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_S8

pt::renderers::VideoWriter, 191
CODEC_ID_PCM_U16BE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_U16LE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_U24BE
pt::renderers::VideoWriter, 192
CODEC_ID_PCM_U24LE
pt::renderers::VideoWriter, 192
CODEC_ID_PCM_U32BE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_U32LE
pt::renderers::VideoWriter, 191
CODEC_ID_PCM_U8
pt::renderers::VideoWriter, 191
CODEC_ID_PGM
pt::renderers::VideoWriter, 190
CODEC_ID_PGMYUV
pt::renderers::VideoWriter, 191
CODEC_ID_PNG
pt::renderers::VideoWriter, 190
CODEC_ID_PPM
pt::renderers::VideoWriter, 190
CODEC_ID_QDM2
pt::renderers::VideoWriter, 193
CODEC_ID_QDRAW
pt::renderers::VideoWriter, 190
CODEC_ID_QPEG
pt::renderers::VideoWriter, 190
CODEC_ID_QTRLE
pt::renderers::VideoWriter, 190
CODEC_ID_RA_144
pt::renderers::VideoWriter, 192
CODEC_ID_RA_288
pt::renderers::VideoWriter, 192
CODEC_ID_RAWVIDEO
pt::renderers::VideoWriter, 189
CODEC_ID_ROQ
pt::renderers::VideoWriter, 190
CODEC_ID_ROQ_DPCM
pt::renderers::VideoWriter, 192
CODEC_ID_RPZA
pt::renderers::VideoWriter, 190
CODEC_ID_RV10
pt::renderers::VideoWriter, 189
CODEC_ID_RV20
pt::renderers::VideoWriter, 189
CODEC_ID_RV30
pt::renderers::VideoWriter, 191
CODEC_ID_RV40
pt::renderers::VideoWriter, 191
CODEC_ID_SHORTEN
pt::renderers::VideoWriter, 193
CODEC_ID_SMACKAUDIO
pt::renderers::VideoWriter, 193
CODEC_ID_SMACKVIDEO
pt::renderers::VideoWriter, 191
CODEC_ID_SMC
pt::renderers::VideoWriter, 190
CODEC_ID_SNOW
pt::renderers::VideoWriter, 190
CODEC_ID_SOL_DPCM
pt::renderers::VideoWriter, 192
CODEC_ID_SONIC
pt::renderers::VideoWriter, 193
CODEC_ID_SONIC_LS
pt::renderers::VideoWriter, 193
CODEC_ID_SP5X
pt::renderers::VideoWriter, 189
CODEC_ID_SVQ1
pt::renderers::VideoWriter, 189
CODEC_ID_SVQ3
pt::renderers::VideoWriter, 189
CODEC_ID_THEORA
pt::renderers::VideoWriter, 189
CODEC_ID_TRUEMOTION1
pt::renderers::VideoWriter, 190
CODEC_ID_TRUEMOTION2
pt::renderers::VideoWriter, 191
CODEC_ID_TRUESPEECH
pt::renderers::VideoWriter, 193
CODEC_ID_TSCC
pt::renderers::VideoWriter, 190
CODEC_ID_TTA
pt::renderers::VideoWriter, 193
CODEC_ID_ULTI
pt::renderers::VideoWriter, 190
CODEC_ID_VC9

pt::renderers::VideoWriter, 191
CODEC_ID_VCR1
pt::renderers::VideoWriter, 190
CODEC_ID_VIXL
pt::renderers::VideoWriter, 190
CODEC_ID_VMDAUDIO
pt::renderers::VideoWriter, 193
CODEC_ID_VMDVIDEO
pt::renderers::VideoWriter, 190
CODEC_ID_VORBIS
pt::renderers::VideoWriter, 193
CODEC_ID_VP3
pt::renderers::VideoWriter, 189
CODEC_ID_WESTWOOD_SND1
pt::renderers::VideoWriter, 193
CODEC_ID_WMAV1
pt::renderers::VideoWriter, 193
CODEC_ID_WMAV2
pt::renderers::VideoWriter, 193
CODEC_ID_WMV1
pt::renderers::VideoWriter, 189
CODEC_ID_WMV2
pt::renderers::VideoWriter, 189
CODEC_ID_WMV3
pt::renderers::VideoWriter, 191
CODEC_ID_WNV1
pt::renderers::VideoWriter, 191
CODEC_ID_WS_VQA
pt::renderers::VideoWriter, 190
CODEC_ID_XAN_DPCM
pt::renderers::VideoWriter, 192
CODEC_ID_XAN_WC3
pt::renderers::VideoWriter, 190
CODEC_ID_XAN_WC4
pt::renderers::VideoWriter, 190
CODEC_ID_XVID
pt::renderers::VideoWriter, 190
CODEC_ID_ZLIB
pt::renderers::VideoWriter, 190
CODEC_ID_ZMBV
pt::renderers::VideoWriter, 191
compX
pt::processing::WorldProcessing, 200
compY
pt::processing::WorldProcessing, 200
congnición, 91
correlación, 67
count
pt::math::Distribution, 130
CProfile
pt::Profiler::CProfile, 178
createGLWindow
pt::renderers::GLWindow, 155
createProfile
pt::Profiler, 175
crecimientoMedia
pt::processing::EyeProcessing, 148
crecimientoRelleno
pt::processing::EyeProcessing, 148
cristalino, 16
ctx
pt::renderers::GLWindow, 156
cur_limit
pt::math::Distribution, 130
cur_val
pt::Profiler::CProfile, 179
curChannel
pt::readers::CamReader, 122

dark_pupil
pt::processing::EyeProcessing, 151
debian, 71
desc
pt::readers::CamReader, 122
description
main.cpp, 222
deskMode
pt::renderers::GLWindow, 156
destroy
pt::processing::BaseProcessing, 114
pt::processing::EyeProcessing, 148
pt::processing::WorldProcessing, 200
pt::readers::CamReader, 120
pt::readers::Capture, 126

detección pupila, 48
detection
pt::processing::EyeProcessing, 148
pt::processing::WorldProcessing, 201
Deutsh y Deutsch, 9
dev_name
pt::readers::Capture, 127
DFD's, 39
Diagrama de Contexto, 39
Dimitri van Heesch, 73
dist
pt::math::Real2dPoint, 181
dist_xp
pt::math::Kriging, 167
dist_xx
pt::math::Kriging, 167
dist_xx_calculated
pt::math::Kriging, 167
distance
pt::math::Kriging, 167
Distribution
pt::math::Distribution, 129
Doxygen, 73
dpy
pt::renderers::GLWindow, 156
drawCross
pt::eyeboard, 135
pt::Image, 160
drawGLScene
pt::renderers::GLWindow, 155
drawKey
pt::eyeboard, 135
drawPoint
pt::eyeboard, 135
drawRect
pt::Image, 160
dwChunkId
pt::readers::AviReader::SAviOldIndexEntry, 107
dwFlags
pt::readers::AviReader::AviStreamHeader, 103
pt::readers::AviReader::SAviMainHeader, 105
pt::readers::AviReader::SAviOldIndexEntry, 107
dwHeight
pt::readers::AviReader::SAviMainHeader, 105
dwInitialFrames
pt::readers::AviReader::AviStreamHeader, 103
pt::readers::AviReader::SAviMainHeader, 105
dwLength
pt::readers::AviReader::AviStreamHeader, 103
dwMaxBytesPerSec
pt::readers::AviReader::SAviMainHeader, 105
dwMicroSecPerFrame
pt::readers::AviReader::SAviMainHeader, 105
dwOffset
pt::readers::AviReader::SAviOldIndexEntry, 107
DWORD
avireader.h, 205
dwPaddingGranularity
pt::readers::AviReader::SAviMainHeader, 105
dwQuality
pt::readers::AviReader::AviStreamHeader, 103
dwRate
pt::readers::AviReader::AviStreamHeader, 103
dwReserved
pt::readers::AviReader::SAviMainHeader, 105
dwSampleSize
pt::readers::AviReader::AviStreamHeader, 103
dwScale
pt::readers::AviReader::AviStreamHeader, 103

dwSize
pt::readers::AviReader::SAviOldIndexEntry, 107
dwStart
pt::readers::AviReader::AviStreamHeader, 103
dwStreams
pt::readers::AviReader::SAviMainHeader, 105
dwSuggestedBufferSize
pt::readers::AviReader::AviStreamHeader, 103
pt::readers::AviReader::SAviMainHeader, 105
dwTotalFrames
pt::readers::AviReader::SAviMainHeader, 105
dwWidth
pt::readers::AviReader::SAviMainHeader, 105

e
pt::math::Definitions, 129
EagleEyes, 36
ECodecID
pt::renderers::VideoWriter, 188
EHT, 56
EKrigingModel
pt::math::Kriging, 165
electro-oculografía, 25
EMachineStatus
pt::processing::EyeProcessing, 147
entries
pt::readers::AviReader::SAviOldIndex, 106
EOG, 25
EPixelFormat
pt::renderers::VideoWriter, 193
esclerótica, 16, 60
escrito
pt::eyeboard, 136
EStatus
pt::processing::WorldProcessing, 200
estela
pt::eyeboard, 136
estela_pos
pt::eyeboard, 136
ExponentialKriging
pt::math::Kriging, 165
eye tracker, 25
eye_h
pt::eyeboard, 136
eye_position
pt::processing::EyeProcessing, 151
eye_processing
pt::eyeboard, 136
eye_reader
pt::eyeboard, 137
eye_w
pt::eyeboard, 137
EyeBoard, 89
eyeboard
pt::eyeboard, 134
eyeboard.h
QT_CLEAN_NAMESPACE, 213
eyeboard/src/avireader.cpp, 203
eyeboard/src/avireader.h, 204
eyeboard/src/baseprocessing.cpp, 205
eyeboard/src/baseprocessing.h, 205
eyeboard/src/basereader.cpp, 206
eyeboard/src/basereader.h, 206
eyeboard/src/camreader.cpp, 207
eyeboard/src/camreader.h, 207
eyeboard/src/capture.cpp, 208
eyeboard/src/capture.h, 209
eyeboard/src/distribution.cpp, 211
eyeboard/src/distribution.h, 211
eyeboard/src/eyeboard.cpp, 212
eyeboard/src/eyeboard.h, 212
eyeboard/src/eyeprocessing.cpp, 214
eyeboard/src/eyeprocessing.h, 214
eyeboard/src/glwindow.cpp, 215
eyeboard/src/glwindow.h, 216
eyeboard/src/image.cpp, 217
eyeboard/src/image.h, 217
eyeboard/src/kriging.cpp, 217
eyeboard/src/kriging.h, 218

eyeboard/src/logger.cpp, 218
eyeboard/src/logger.h, 219
eyeboard/src/main.cpp, 221
eyeboard/src/maths.cpp, 222
eyeboard/src/maths.h, 223
eyeboard/src/profiler.cpp, 223
eyeboard/src/profiler.h, 223
eyeboard/src/singleton.cpp, 224
eyeboard/src/singleton.h, 224
eyeboard/src/videowriter.cpp, 225
eyeboard/src/videowriter.h, 225
eyeboard/src/worldprocessing.cpp, 226
eyeboard/src/worldprocessing.h, 226
eyeboardconfig, 84
EyeProcessing
    pt::processing::EyeProcessing, 147
EyeToy, 36

fóvea, 18
fcc
    pt::readers::AviReader::AviStreamHeader, 103
    pt::readers::AviReader::SAviMainHeader, 105
    pt::readers::AviReader::SAviOldIndex, 106
fccHandler
    pt::readers::AviReader::AviStreamHeader, 103
fccType
    pt::readers::AviReader::AviStreamHeader, 103
fd
    pt::readers::Capture, 127
ffmpeg, 77
fijaciones, 22
file_name
    pt::Logger, 172
    pt::renderers::VideoWriter, 195
finding
    pt::processing::WorldProcessing, 200
fmt
    pt::renderers::VideoWriter, 195
foto-oculografía, 26
found
    pt::processing::WorldProcessing, 200
fps
    pt::eyeboard, 137
    pt::renderers::VideoWriter, 195
frame_count
    pt::renderers::VideoWriter, 196
freeImage
    pt::Image, 160
freq
    pt::Profiler::CProfile, 179
fs
    pt::renderers::GLWindow, 156
FSF, 73

g
    pt::SPixel, 184
gauss
    pt::processing::WorldProcessing, 202
GaussianKriging
    pt::math::Kriging, 165
GCC, 72, 73
GDB, 72, 73
getAcumTime
    pt::Profiler, 175
    pt::Profiler::CProfile, 178
getBpp
    pt::Image, 160
getBuffer
    pt::readers::AviReader, 99
    pt::readers::BaseReader, 117
    pt::readers::CamReader, 120
getBufferPtr
    pt::processing::BaseProcessing, 114
    pt::processing::EyeProcessing, 149
    pt::processing::WorldProcessing, 201
    pt::readers::AviReader, 99
    pt::readers::BaseReader, 117
    pt::readers::CamReader, 121

pt::readers::Capture, 126
getBufferSize
pt::readers::CamReader, 121
getCenter
pt::math::Distribution, 130
getCurrentChannel
pt::readers::CamReader, 121
getHeight
pt::Image, 160
pt::readers::AviReader, 99
pt::readers::BaseReader, 117
getHistogram
pt::processing::EyeProcessing, 149
getImage
pt::processing::EyeProcessing, 149
pt::processing::WorldProcessing, 201
getImage24
pt::processing::WorldProcessing, 201
getInterpolatedValue
pt::math::Kriging, 166
getMax
pt::Profiler, 175
getMean
pt::Profiler, 176
pt::Profiler::CProfile, 178
getMin
pt::Profiler, 176
getName
pt::Profiler::CProfile, 178
getNumber
pt::math::Distribution, 130
getNumPoints
pt::math::Kriging, 166
getNumTimes
pt::Profiler::CProfile, 178
getPicture
pt::readers::AviReader, 99
pt::readers::BaseReader, 118
getPictureOptions
pt::readers::CamReader, 121
getPixels
pt::Image, 160
getPoint
pt::processing::WorldProcessing, 201
getPupilPosition
pt::processing::EyeProcessing, 149
getSingleton
pt::Logger, 171
pt::Singleton, 183
getSingletonPtr
pt::Singleton, 183
getStdDev
pt::Profiler, 176
getWidth
pt::Image, 160
pt::readers::AviReader, 99
pt::readers::BaseReader, 118
Gibson, 8
Gimp, 73
GLWindow
pt::renderers::GLWindow, 154
GNU/Linux, 71
GPL, 72, 73
gravedad, 89
gravedad
pt::eyeboard, 137
guardar_video
pt::eyeboard, 137

HEIGHT
pt::eyeboard, 137
pt::processing::BaseProcessing, 115
pt::readers::Capture, 127
height
pt::eyeboard::SKey, 141
pt::renderers::GLWindow, 156
pt::renderers::VideoWriter, 196
hist_closed_eye
pt::processing::EyeProcessing, 151
hist_open_eye
pt::processing::EyeProcessing, 151
histogram
pt::processing::EyeProcessing, 151
histogram_copia

pt::processing::EyeProcessing, 151
Hough circular, 50
Hough para elipses, 56
humor acuoso, 17
humor vítreo, 17

Image
pt::Image, 159
image
pt::renderers::GLWindow, 156
imperfecciones ópticas, 13
infrared, 89
init
pt::eyeboard, 135
pt::Image, 160
pt::processing::EyeProcessing, 149
pt::processing::WorldProcessing, 201
pt::readers::CamReader, 121
pt::readers::Capture, 126
pt::renderers::VideoWriter, 195
init_device
pt::readers::Capture, 126
init_mmap
pt::readers::Capture, 126
initGLScene
pt::renderers::GLWindow, 155
initKey
pt::eyeboard, 135
initKeys
pt::renderers::GLWindow, 155
initted
pt::processing::WorldProcessing, 202
pt::renderers::VideoWriter, 196
Int2dPoint
pt::math::Int2dPoint, 162
int_eye_dist
pt::eyeboard, 137
int_eye_type
pt::eyeboard, 137
int_world_dist
pt::eyeboard, 137
int_world_type
pt::eyeboard, 137
Intel®, 76, 77
IPP, 76, 83
iris, 17
isFound
pt::processing::WorldProcessing, 201
isInitialized
pt::Image, 161
isOpen
pt::processing::EyeProcessing, 150
isPlaying
pt::readers::AviReader, 100
isPoint
pt::processing::WorldProcessing, 201
isRunning
pt::readers::CamReader, 121
itune, 86

Jam, 8
joystick, 33

k_mod
pt::eyeboard, 137
kalman, 233
Kanizsa, 9
KDevelop, 72
keyPressEvent
pt::eyeboard, 135
keys
pt::eyeboard, 138
KEYSIZE
pt::eyeboard, 138
Kile, 74
killGLWindow
pt::renderers::GLWindow, 155
Kimage, 74
Kosslyn, 10
kriger
pt::eyeboard, 138
Kriging
pt::math::Kriging, 165
kriging, 233

kriging_distance
pt::eyeboard, 138
kriging_model
pt::math::Kriging, 167
kriging_name
pt::eyeboard, 138

láser, 231
last_tecla
pt::eyeboard, 138
left
pt::readers::AviReader::AviStreamHeader, 104
length
pt::readers::Capture::buffer, 128
Lentes de contacto, 25
librerías, 83
limbo, 16
LinearKriging
pt::math::Kriging, 165
LL_ALERT
logger.h, 220
LL_CRIT
logger.h, 220
LL_DEBUG
logger.h, 220
LL_EMERG
logger.h, 220
LL_ERR
logger.h, 220
LL_INFO
logger.h, 220
LL_NOTICE
logger.h, 220
LL_WARNING
logger.h, 220
loadClosedEye
pt::processing::EyeProcessing, 150
loadOpenEye
pt::processing::EyeProcessing, 150
log_level
pt::Logger, 172
log_type
pt::Logger, 172
Logger
pt::Logger, 171
logger
pt::eyeboard, 138
logger.h
LL_ALERT, 220
LL_CRIT, 220
LL_DEBUG, 220
LL_EMERG, 220
LL_ERR, 220
LL_INFO, 220
LL_NOTICE, 220
LL_WARNING, 220
LT_AUTH, 220
LT_AUTHPRIV, 220
LT_CRON, 220
LT_DAEMON, 221
LT_FTP, 221
LT_KERN, 221
LT_LPR, 221
LT_MAIL, 221
LT_NEWS, 221
LT_SYSLOG, 221
LT_USER, 221
LT_UUCP, 221
logMessage
pt::Logger, 171
LONG
avireader.h, 205
LT_AUTH
logger.h, 220
LT_AUTHPRIV
logger.h, 220
LT_CRON
logger.h, 220
LT_DAEMON
logger.h, 221
LT_FTP
logger.h, 221
LT_KERN
logger.h, 221
LT_LPR
logger.h, 221
LT_MAIL

logger.h, 221
LT_NEWS
logger.h, 221
LT_SYSLOG
logger.h, 221
LT_USER
logger.h, 221
LT_UUCP
logger.h, 221

mácula, 19
músculos oculares, 21
m_bpp
pt::Image, 161
m_capture
pt::processing::BaseProcessing, 115
m_gauss
pt::processing::EyeProcessing, 152
m_gray
pt::processing::EyeProcessing, 152
m_height
pt::Image, 161
m_initialized
pt::Image, 161
m_mask
pt::processing::EyeProcessing, 152
m_step_bytes
pt::Image, 161
m_temp_closed_eye
pt::processing::EyeProcessing, 152
m_temp_open_eye
pt::processing::EyeProcessing, 152
m_width
pt::Image, 161
main
main.cpp, 222
main.cpp
description, 222
main, 222
options, 222
version, 222
Mesa, 77
minPotSup2
pt::renderers::GLWindow, 155
MMX, 76
modelo ocular, 59
movi_offset
pt::readers::AviReader, 101
mp_pixels
pt::Image, 161
ms
pt::eyeboard, 138
ms_Singleton
pt::Singleton, 183
Multiplicador de Lagrange, 63
my_log
pt::Logger, 172

n_buffers
pt::readers::Capture, 127
name
pt::eyeboard::SKey, 141
nAvgBytesPerSec
pt::readers::AviReader::SWaveFormatEx, 111
nBlockAlign
pt::readers::AviReader::SWaveFormatEx, 111
nChannels
pt::readers::AviReader::SWaveFormatEx, 111
nebula
pt::processing::EyeProcessing, 152
next_frame
pt::readers::AviReader, 101
next_index_entry
pt::readers::AviReader, 101
nistagmos, 23
nSamplesPerSec
pt::readers::AviReader::SWaveFormatEx, 111
num_times
pt::Profiler::CProfile, 179
numBuffers
pt::readers::CamReader, 122
numChannels
pt::readers::CamReader, 121

NUMESTELAPOINTS
pt::eyeboard, 138
NUMPOINTS
pt::processing::WorldProcessing, 202
numSamples
pt::math::Kriging, 167
NUMTECLAS
pt::eyeboard, 138

oc
pt::renderers::VideoWriter, 196
ojo humano, 11
open_device
pt::readers::Capture, 126
open_eye
pt::processing::EyeProcessing, 147
open_eye_saved
pt::processing::EyeProcessing, 147
open_video
pt::renderers::VideoWriter, 195
OpenCV, 77
openDevice
pt::readers::CamReader, 121
openEyes, 59
openFile
pt::readers::AviReader, 100
OpenGL, 77
operator=
pt::math::Real2dPoint, 181
options
main.cpp, 222

pantalla táctil, 34
percepción, 91
persecuciones lentas, 22
persistencia de la visión, 19
phi (fenómeno), 19
pi
pt::math::Definitions, 129
picture
pt::renderers::VideoWriter, 196
PIX_FMT_BGR24
pt::renderers::VideoWriter, 194
PIX_FMT_GRAY8
pt::renderers::VideoWriter, 194
PIX_FMT_MONOBLACK
pt::renderers::VideoWriter, 194
PIX_FMT_MONOWHITE
pt::renderers::VideoWriter, 194
PIX_FMT_NB
pt::renderers::VideoWriter, 194
PIX_FMT_NONE
pt::renderers::VideoWriter, 193
PIX_FMT_PAL8
pt::renderers::VideoWriter, 194
PIX_FMT_RGB24
pt::renderers::VideoWriter, 194
PIX_FMT_RGB555
pt::renderers::VideoWriter, 194
PIX_FMT_RGB565
pt::renderers::VideoWriter, 194
PIX_FMT_RGBA32
pt::renderers::VideoWriter, 194
PIX_FMT_UYVY411
pt::renderers::VideoWriter, 194
PIX_FMT_UYVY422
pt::renderers::VideoWriter, 194
PIX_FMT_XVMC_MPEG2_IDCT
pt::renderers::VideoWriter, 194
PIX_FMT_XVMC_MPEG2_MC
pt::renderers::VideoWriter, 194
PIX_FMT_YUV410P
pt::renderers::VideoWriter, 194
PIX_FMT_YUV411P
pt::renderers::VideoWriter, 194
PIX_FMT_YUV420P
pt::renderers::VideoWriter, 193
PIX_FMT_YUV422
pt::renderers::VideoWriter, 193
PIX_FMT_YUV422P
pt::renderers::VideoWriter, 194
PIX_FMT_YUV444P
pt::renderers::VideoWriter, 194
PIX_FMT_YUVJ420P
pt::renderers::VideoWriter, 194
PIX_FMT_YUVJ422P
pt::renderers::VideoWriter, 194

PIX_FMT_YUVJ444P
pt::renderers::VideoWriter, 194
pixel_format
pt::renderers::VideoWriter, 196
playing
pt::processing::EyeProcessing, 147
pt::processing::WorldProcessing, 200
pt::readers::AviReader, 101
PlayStation, 36
point
pt::math::Kriging::SPointValue, 168
point_array
pt::processing::WorldProcessing, 202
points
pt::processing::WorldProcessing, 202
Posner, 10
prepareBuffer
pt::readers::CamReader, 121
prePthread
pt::readers::CamReader, 121
printError
pt::math::Kriging, 166
printMatrices
pt::math::Kriging, 166
processCursor
pt::eyeboard, 135
processFrame
pt::processing::BaseProcessing, 114
pt::processing::EyeProcessing, 150
pt::processing::WorldProcessing, 201
prof_name
pt::Profiler::CProfile, 179
Profiler
pt::Profiler, 175
profiles
pt::Profiler, 176
psicología, 91
pt::eyeboard, 131
∼eyeboard, 134
avi_writer, 136
canvas, 136
canvas_kriging, 136
canvas_text, 136
canvas_view, 136
drawCross, 135
drawKey, 135
drawPoint, 135
escrito, 136
estela, 136
estela_pos, 136
eye_h, 136
eye_processing, 136
eye_reader, 137
eye_w, 137
eyeboard, 134
fps, 137
gravedad, 137
guardar_video, 137
HEIGHT, 137
init, 135
initKey, 135
int_eye_dist, 137
int_eye_type, 137
int_world_dist, 137
int_world_type, 137
k_mod, 137
keyPressEvent, 135
keys, 138
KEYSIZE, 138
kriger, 138
kriging_distance, 138
kriging_name, 138
last_tecla, 138
logger, 138
ms, 138
NUMESTELAPOINTS, 138
NUMTECLAS, 138
processCursor, 135
readConfig, 135
run_timer, 138
SEP, 139
slotRun, 135

slotStartRecord, 135
slotStopRecord, 136
subpulsaciones, 139
video, 139
WIDTH, 139
world_canvas_pixmap, 139
world_h, 139
world_image, 139
world_pixmap_array, 139
world_processing, 139
world_reader, 139
world_sprite, 140
world_w, 140
pt::eyeboard::SKey, 140
height, 141
name, 141
pulsaciones, 141
width, 141
x, 141
y, 141
pt::Image, 157
∼Image, 159
drawCross, 160
drawRect, 160
freeImage, 160
getBpp, 160
getHeight, 160
getPixels, 160
getWidth, 160
Image, 159
init, 160
isInitialized, 161
m_bpp, 161
m_height, 161
m_initialized, 161
m_step_bytes, 161
m_width, 161
mp_pixels, 161
pt::Logger, 169
∼Logger, 171
app_name, 172
file_name, 172
getSingleton, 171
log_level, 172
log_type, 172
Logger, 171
logMessage, 171
my_log, 172
setAppName, 172
setLogFile, 172
setLogLevel, 172
syslogMessage, 172
pt::math::Definitions, 128
e, 129
pi, 129
pt::math::Distribution, 129
∼Distribution, 130
addPoint, 130
count, 130
cur_limit, 130
Distribution, 129
getCenter, 130
getNumber, 130
reset, 130
x_array, 130
xAt, 130
y_array, 131
yAt, 130
pt::math::Int2dPoint, 161
pt::math::Int2dPoint
Int2dPoint, 162
x, 162
y, 162
pt::math::Kriging, 163
∼Kriging, 165
addPoint, 165
aux_xx, 166
buffer, 166
calculateDist, 165
calculateGamma, 166
dist_xp, 167
dist_xx, 167
dist_xx_calculated, 167
distance, 167
EKrigingModel, 165
ExponentialKriging, 165
GaussianKriging, 165
getInterpolatedValue, 166

getNumPoints, 166
Kriging, 165
kriging_model, 167
LinearKriging, 165
numSamples, 167
printError, 166
printMatrices, 166
reset, 166
setDistance, 166
setKrigingModel, 166
SphericalKriging, 165
variogram, 167
weights, 167
widthheight, 167
pt::math::Kriging::SPointValue, 168
pt::math::Kriging::SPointValue
point, 168
SPointValue, 168
value, 168
pt::math::Real2dPoint, 180
pt::math::Real2dPoint
dist, 181
operator=, 181
Real2dPoint, 180, 181
toIppiPoint, 181
x, 181
y, 181
pt::processing::BaseProcessing, 111
pt::processing::BaseProcessing
∼BaseProcessing, 114
BaseProcessing, 114
destroy, 114
getBufferPtr, 114
HEIGHT, 115
m_capture, 115
processFrame, 114
WIDTH, 115
pt::processing::EyeProcessing, 141
closed_eye, 147
closed_eye_saved, 147
open_eye, 147
open_eye_saved, 147
playing, 147
start, 147
stopped, 147
pt::processing::EyeProcessing
∼EyeProcessing, 147
all_image, 151
bright_pupil, 151
calcOpen, 147
calcROI, 148
crecimientoMedia, 148
crecimientoRelleno, 148
dark_pupil, 151
destroy, 148
detection, 148
EMachineStatus, 147
eye_position, 151
EyeProcessing, 147
getBufferPtr, 149
getHistogram, 149
getImage, 149
getPupilPosition, 149
hist_closed_eye, 151
hist_open_eye, 151
histogram, 151
histogram_copia, 151
init, 149
isOpen, 150
loadClosedEye, 150
loadOpenEye, 150
m_gauss, 152
m_gray, 152
m_mask, 152
m_temp_closed_eye, 152
m_temp_open_eye, 152
nebula, 152
processFrame, 150
roi_point, 152
roi_size, 152
saveClosedEye, 150
saveOpenEye, 150
status, 152
updateHistogram, 151
pt::processing::WorldProcessing, 197
finding, 200
found, 200
playing, 200

pt::processing::WorldProcessing
~WorldProcessing, 200
compX, 200
compY, 200
destroy, 200
detection, 201
EStatus, 200
gauss, 202
getBufferPtr, 201
getImage, 201
getImage24, 201
getPoint, 201
init, 201
initted, 202
isFound, 201
isPoint, 201
NUMPOINTS, 202
point_array, 202
points, 202
processFrame, 201
roi, 202
setPlayingOnly, 202
status, 202
temp_array, 202
WorldProcessing, 200
pt::Profiler, 173
~Profiler, 175
createProfile, 175
getAcumTime, 175
getMax, 175
getMean, 176
getMin, 176
getStdDev, 176
Profiler, 175
profiles, 176
startTimer, 176
stopTimer, 176
pt::Profiler::CProfile, 176
~CProfile, 178
acum_time, 179
addTime, 178
CProfile, 178
cur_val, 179
freq, 179
getAcumTime, 178
getMean, 178
getName, 178
getNumTimes, 178
num_times, 179
prof_name, 179
startTimer, 178
stopTimer, 179
time_end, 179
time_start, 179
pt::readers::AviReader, 95
pt::readers::AviReader
~AviReader, 99
avi_main_header, 100
avi_old_index, 100
avi_stream_header, 100
AviReader, 99
bit_map_info, 100
buffer, 101
buffer_rgb24, 101
getBuffer, 99
getBufferPtr, 99
getHeight, 99
getPicture, 99
getWidth, 99
isPlaying, 100
movi_offset, 101
next_frame, 101
next_index_entry, 101
openFile, 100
playing, 101
readHeader, 100
riff_file, 101
riff_name, 101
setPreferredSize, 100
pt::readers::AviReader::AviStreamHeader, 102
pt::readers::AviReader::AviStream-Header
bottom, 103
cb, 103
dwFlags, 103
dwInitialFrames, 103
dwLength, 103

dwQuality, 103
dwRate, 103
dwSampleSize, 103
dwScale, 103
dwStart, 103
dwSuggestedBufferSize, 103
fcc, 103
fccHandler, 103
fccType, 103
left, 104
rcFrame, 104
right, 104
top, 104
wLanguage, 104
wPriority, 104
pt::readers::AviReader::SAviMainHeader, 104
pt::readers::AviReader::SAviMainHeader
cb, 105
dwFlags, 105
dwHeight, 105
dwInitialFrames, 105
dwMaxBytesPerSec, 105
dwMicroSecPerFrame, 105
dwPaddingGranularity, 105
dwReserved, 105
dwStreams, 105
dwSuggestedBufferSize, 105
dwTotalFrames, 105
dwWidth, 105
fcc, 105
pt::readers::AviReader::SAviOldIndex, 106
pt::readers::AviReader::SAviOldIndex
cb, 106
entries, 106
fcc, 106
pt::readers::AviReader::SAviOldIndexEntry, 107
pt::readers::AviReader::SAviOldIndexEntry
dwChunkId, 107
dwFlags, 107
dwOffset, 107
dwSize, 107
pt::readers::AviReader::STagBITMAPINFO, 107
pt::readers::AviReader::STagBITMAPINFO
bmiHeader, 108
pt::readers::AviReader::STagBITMAPINFOHEADER, 108
pt::readers::AviReader::STagBITMAPINFOHEADER
biBitCount, 109
biClrImportant, 109
biClrUsed, 109
biCompression, 109
biHeight, 109
biPlanes, 109
biSize, 109
biSizeImage, 109
biWidth, 109
biXPelsPerMeter, 109
biYPelsPerMeter, 109
pt::readers::AviReader::STagRGBQUAD, 110
pt::readers::AviReader::STagRGBQUAD
rgbBlue, 110
rgbGreen, 110
rgbRed, 110
rgbReserved, 110
pt::readers::AviReader::SWaveFormatEx, 110
pt::readers::AviReader::SWaveFormatEx
cbSize, 111
nAvgBytesPerSec, 111
nBlockAlign, 111
nChannels, 111
nSamplesPerSec, 111
wBitsPerSample, 111
wFormatTag, 111
pt::readers::BaseReader, 115
pt::readers::BaseReader
∼BaseReader, 117

BaseReader, 117
getBuffer, 117
getBufferPtr, 117
getHeight, 117
getPicture, 118
getWidth, 118
setPreferredSize, 118
pt::readers::CamReader, 118
pt::readers::CamReader
∼CamReader, 120
buffer, 122
bufferSize, 122
CamReader, 120
captureLoop, 120
closeDevice, 120
curChannel, 122
desc, 122
destroy, 120
getBuffer, 120
getBufferPtr, 121
getBufferSize, 121
getCurrentChannel, 121
getPictureOptions, 121
init, 121
isRunning, 121
numBuffers, 122
numChannels, 121
openDevice, 121
prepareBuffer, 121
prePthread, 121
reading_frame_n, 122
running, 123
setChannel, 122
setPalette, 122
setPictureOptions, 122
setPreferredSize, 122
thread, 123
thread_attr, 123
vcaps, 123
vchannel, 123
vpic, 123
vwin, 123
pt::readers::Capture, 123
∼Capture, 125
BPP, 127
buff_ptr, 127
buffers, 127
Capture, 125
close_device, 126
destroy, 126
dev_name, 127
fd, 127
getBufferPtr, 126
HEIGHT, 127
init, 126
init_device, 126
init_mmap, 126
n_buffers, 127
open_device, 126
read_frame, 126
start_capturing, 126
stop_capturing, 126
uninit_device, 126
WIDTH, 127
xioctl, 126
pt::readers::Capture::buffer, 128
length, 128
start, 128
pt::renderers::GLWindow, 153
∼GLWindow, 154
attr, 156
bpp, 156
createGLWindow, 155
ctx, 156
deskMode, 156
dpy, 156
drawGLScene, 155
fs, 156
GLWindow, 154
height, 156
image, 156
initGLScene, 155
initKeys, 155
killGLWindow, 155
minPotSup2, 155
resizeGLScene, 155
running, 156
scale_height, 156

scale_width, 156
screen, 157
setImage, 155
swapBuffers, 155
texid, 157
update, 155
width, 157
win, 157
window_name, 157
x, 157
y, 157
pt::renderers::VideoWriter, 184
CODEC_ID_4XM, 190
CODEC_ID_8BPS, 190
CODEC_ID_AAC, 192
CODEC_ID_AASC, 191
CODEC_ID_AC3, 192
CODEC_ID_ADPCM_4XM, 192
CODEC_ID_ADPCM_ADX, 192
CODEC_ID_ADPCM_CT, 192
CODEC_ID_ADPCM_EA, 192
CODEC_ID_ADPCM_G726, 192
CODEC_ID_ADPCM_IMA_DK3, 192
CODEC_ID_ADPCM_IMA_DK4, 192
CODEC_ID_ADPCM_IMA_QT, 192
CODEC_ID_ADPCM_IMA_SMJPEG, 192
CODEC_ID_ADPCM_IMA_WAV, 192
CODEC_ID_ADPCM_IMA_WS, 192
CODEC_ID_ADPCM_MS, 192
CODEC_ID_ADPCM_SBPRO_2, 192
CODEC_ID_ADPCM_SBPRO_3, 192
CODEC_ID_ADPCM_SBPRO_4, 192
CODEC_ID_ADPCM_SWF, 192
CODEC_ID_ADPCM_XA, 192
CODEC_ID_ADPCM_YAMAHA, 192
CODEC_ID_ALAC, 193
CODEC_ID_AMR_NB, 192
CODEC_ID_AMR_WB, 192
CODEC_ID_ASV1, 189
CODEC_ID_ASV2, 189
CODEC_ID_AVS, 191
CODEC_ID_BMP, 191
CODEC_ID_CINEPAK, 190
CODEC_ID_CLJR, 190
CODEC_ID_COOK, 193
CODEC_ID_CSCD, 191
CODEC_ID_CYUV, 189
CODEC_ID_DTS, 193
CODEC_ID_DVAUDIO, 193
CODEC_ID_DVB_SUBTITLE, 193
CODEC_ID_DVD_SUBTITLE, 193
CODEC_ID_DVVIDEO, 189
CODEC_ID_FFV1, 190
CODEC_ID_FFVHUFF, 191
CODEC_ID_FLAC, 193
CODEC_ID_FLIC, 190
CODEC_ID_FLV1, 189
CODEC_ID_FRAPS, 191
CODEC_ID_GSM, 193
CODEC_ID_H261, 189
CODEC_ID_H263, 189
CODEC_ID_H263I, 189
CODEC_ID_H263P, 189
CODEC_ID_H264, 189
CODEC_ID_HUFFYUV, 189
CODEC_ID_IDCIN, 190
CODEC_ID_INDEO2, 191
CODEC_ID_INDEO3, 189
CODEC_ID_INTERPLAY_DPCM, 192
CODEC_ID_INTERPLAY_VIDEO, 190
CODEC_ID_JPEGLS, 189
CODEC_ID_LJPEG, 189
CODEC_ID_LOCO, 191

CODEC_ID_MACE3, 193
CODEC_ID_MACE6, 193
CODEC_ID_MDEC, 190
CODEC_ID_MJPEG, 189
CODEC_ID_MJPEGB, 189
CODEC_ID_MMVIDEO, 191
CODEC_ID_MP2, 192
CODEC_ID_MP3, 192
CODEC_ID_MP3ADU, 193
CODEC_ID_MP3ON4, 193
CODEC_ID_MPEG1VIDEO, 189
CODEC_ID_MPEG2TS, 193
CODEC_ID_MPEG2VIDEO, 189
CODEC_ID_MPEG2VIDEO_-
XVMC, 189
CODEC_ID_MPEG4, 189
CODEC_ID_MPEG4AAC, 192
CODEC_ID_MSMPEG4V1, 189
CODEC_ID_MSMPEG4V2, 189
CODEC_ID_MSMPEG4V3, 189
CODEC_ID_MSRLE, 190
CODEC_ID_MSVIDEO1, 190
CODEC_ID_MSZH, 190
CODEC_ID_NONE, 189
CODEC_ID_NUV, 191
CODEC_ID_OGGTHEORA, 193
CODEC_ID_PAM, 191
CODEC_ID_PBM, 190
CODEC_ID_PCM_ALAW, 191
CODEC_ID_PCM_MULAW, 191
CODEC_ID_PCM_S16BE, 191
CODEC_ID_PCM_S16LE, 191
CODEC_ID_PCM_S24BE, 191
CODEC_ID_PCM_S24DAUD,
192
CODEC_ID_PCM_S24LE, 191
CODEC_ID_PCM_S32BE, 191
CODEC_ID_PCM_S32LE, 191
CODEC_ID_PCM_S8, 191
CODEC_ID_PCM_U16BE, 191
CODEC_ID_PCM_U16LE, 191
CODEC_ID_PCM_U24BE, 192
CODEC_ID_PCM_U24LE, 192
CODEC_ID_PCM_U32BE, 191
CODEC_ID_PCM_U32LE, 191
CODEC_ID_PCM_U8, 191
CODEC_ID_PGM, 190
CODEC_ID_PGMYUV, 191
CODEC_ID_PNG, 190
CODEC_ID_PPM, 190
CODEC_ID_QDM2, 193
CODEC_ID_QDRAW, 190
CODEC_ID_QPEG, 190
CODEC_ID_QTRLE, 190
CODEC_ID_RA_144, 192
CODEC_ID_RA_288, 192
CODEC_ID_RAWVIDEO, 189
CODEC_ID_ROQ, 190
CODEC_ID_ROQ_DPCM, 192
CODEC_ID_RPZA, 190
CODEC_ID_RV10, 189
CODEC_ID_RV20, 189
CODEC_ID_RV30, 191
CODEC_ID_RV40, 191
CODEC_ID_SHORTEN, 193
CODEC_ID_SMACKAUDIO, 193
CODEC_ID_SMACKVIDEO, 191
CODEC_ID_SMC, 190
CODEC_ID_SNOW, 190
CODEC_ID_SOL_DPCM, 192
CODEC_ID_SONIC, 193
CODEC_ID_SONIC_LS, 193
CODEC_ID_SP5X, 189
CODEC_ID_SVQ1, 189
CODEC_ID_SVQ3, 189
CODEC_ID_THEORA, 189
CODEC_ID_TRUEMOTION1,
190
CODEC_ID_TRUEMOTION2,
191
CODEC_ID_TRUESPEECH, 193
CODEC_ID_TSCC, 190
CODEC_ID_TTA, 193
CODEC_ID_ULTI, 190
CODEC_ID_VC9, 191
CODEC_ID_VCR1, 190
CODEC_ID_VIXL, 190
CODEC_ID_VMDAUDIO, 193

CODEC_ID_VMDVIDEO, 190
CODEC_ID_VORBIS, 193
CODEC_ID_VP3, 189
CODEC_ID_WESTWOOD_-
SND1, 193
CODEC_ID_WMAV1, 193
CODEC_ID_WMAV2, 193
CODEC_ID_WMV1, 189
CODEC_ID_WMV2, 189
CODEC_ID_WMV3, 191
CODEC_ID_WNV1, 191
CODEC_ID_WS_VQA, 190
CODEC_ID_XAN_DPCM, 192
CODEC_ID_XAN_WC3, 190
CODEC_ID_XAN_WC4, 190
CODEC_ID_XVID, 190
CODEC_ID_ZLIB, 190
CODEC_ID_ZMBV, 191
PIX_FMT_BGR24, 194
PIX_FMT_GRAY8, 194
PIX_FMT_MONOBLACK, 194
PIX_FMT_MONOWHITE, 194
PIX_FMT_NB, 194
PIX_FMT_NONE, 193
PIX_FMT_PAL8, 194
PIX_FMT_RGB24, 194
PIX_FMT_RGB555, 194
PIX_FMT_RGB565, 194
PIX_FMT_RGBA32, 194
PIX_FMT_UYVY411, 194
PIX_FMT_UYVY422, 194
PIX_FMT_XVMC_MPEG2_-
IDCT, 194
PIX_FMT_XVMC_MPEG2_MC,
194
PIX_FMT_YUV410P, 194
PIX_FMT_YUV411P, 194
PIX_FMT_YUV420P, 193
PIX_FMT_YUV422, 193
PIX_FMT_YUV422P, 194
PIX_FMT_YUV444P, 194
PIX_FMT_YUVJ420P, 194
PIX_FMT_YUVJ422P, 194
PIX_FMT_YUVJ444P, 194
pt::renderers::VideoWriter
∼VideoWriter, 194
add_video_stream, 195
alloc_picture, 195
bpp, 195
close, 195
ECodecID, 188
EPixelFormat, 193
file_name, 195
fmt, 195
fps, 195
frame_count, 196
height, 196
init, 195
initted, 196
oc, 196
open_video, 195
picture, 196
pixel_format, 196
saveFrame, 195
tmp_picture, 196
video_outbuf, 196
video_outbuf_size, 196
video_st, 196
VideoWriter, 194
width, 196
pt::Singleton, 182
∼Singleton, 183
getSingleton, 183
getSingletonPtr, 183
ms_Singleton, 183
Singleton, 183
pt::SPixel, 183
alfa, 184
b, 184
g, 184
r, 184
SPixel, 184
pulsaciones
pt::eyeboard::SKey, 141
pupila, 16
Purkinje, 59

Qt, 75

QT_CLEAN_NAMESPACE
eyeboard.h, 213

r
pt::SPixel, 184
RANSAC, 58
ratón, 32
rcFrame
pt::readers::AviReader::AviStreamHeader, 104
READ4CC
avireader.cpp, 203
read_frame
pt::readers::Capture, 126
readConfig
pt::eyeboard, 135
readHeader
pt::readers::AviReader, 100
reading_frame_n
pt::readers::CamReader, 122
READRAW
avireader.cpp, 203
READSIZE
avireader.cpp, 203
Real2dPoint
pt::math::Real2dPoint, 180, 181
reconocimiento de voz, 35
reset
pt::math::Distribution, 130
pt::math::Kriging, 166
resizeGLScene
pt::renderers::GLWindow, 155
retina, 18
rgbBlue
pt::readers::AviReader::STagRGBQUAD, 110
rgbGreen
pt::readers::AviReader::STagRGBQUAD, 110
rgbRed
pt::readers::AviReader::STagRGBQUAD, 110
rgbReserved
pt::readers::AviReader::STagRGBQUAD, 110
riff_file
pt::readers::AviReader, 101
riff_name
pt::readers::AviReader, 101
right
pt::readers::AviReader::AviStreamHeader, 104
roi
pt::processing::WorldProcessing, 202
roi_point
pt::processing::EyeProcessing, 152
roi_size
pt::processing::EyeProcessing, 152
run_timer
pt::eyeboard, 138
running
pt::readers::CamReader, 123
pt::renderers::GLWindow, 156

sacádicos, 22, 233
saveClosedEye
pt::processing::EyeProcessing, 150
saveFrame
pt::renderers::VideoWriter, 195
saveOpenEye
pt::processing::EyeProcessing, 150
scale_height
pt::renderers::GLWindow, 156
scale_width
pt::renderers::GLWindow, 156
screen
pt::renderers::GLWindow, 157
SEP
pt::eyeboard, 139
setAppName
pt::Logger, 172
setChannel
pt::readers::CamReader, 122
setDistance
pt::math::Kriging, 166
setImage

pt::renderers::GLWindow, 155
setKrigingModel
pt::math::Kriging, 166
setLogFile
pt::Logger, 172
setLogLevel
pt::Logger, 172
setPalette
pt::readers::CamReader, 122
setPictureOptions
pt::readers::CamReader, 122
setPlayingOnly
pt::processing::WorldProcessing, 202
setPreferredSize
pt::readers::AviReader, 100
pt::readers::BaseReader, 118
pt::readers::CamReader, 122
SIMD, 76
Singleton
pt::Singleton, 183
slotRun
pt::eyeboard, 135
slotStartRecord
pt::eyeboard, 135
slotStopRecord
pt::eyeboard, 136
smooth pursuits, 22
SphericalKriging
pt::math::Kriging, 165
SPixel
pt::SPixel, 184
SPointValue
pt::math::Kriging::SPointValue, 168
Stallman, Richard, 73
starburst, 59
start
pt::processing::EyeProcessing, 147
pt::readers::Capture::buffer, 128
start_capturing
pt::readers::Capture, 126
startTimer
pt::Profiler, 176
pt::Profiler::CProfile, 178
status
pt::processing::EyeProcessing, 152
pt::processing::WorldProcessing, 202
STL, 77
stop_capturing
pt::readers::Capture, 126
stopped
pt::processing::EyeProcessing, 147
stopTimer
pt::Profiler, 176
pt::Profiler::CProfile, 179
subpulsaciones
pt::eyeboard, 139
swapBuffers
pt::renderers::GLWindow, 155
syslogMessage
pt::Logger, 172

tableta digitalizadora, 34
tarball, 84
tarjeta capturadora, 81
teclado, 32
teclado virtual, 89
temp_array
pt::processing::WorldProcessing, 202
texid
pt::renderers::GLWindow, 157
thread
pt::readers::CamReader, 123
thread_attr
pt::readers::CamReader, 123
time_end
pt::Profiler::CProfile, 179
time_start
pt::Profiler::CProfile, 179
tmp_picture
pt::renderers::VideoWriter, 196
toIppiPoint
pt::math::Real2dPoint, 181
top

pt::readers::AviReader::AviStreamHeader, 104
trackball, 33
transformada de Hough, 51
Treisman, 10
Turégano, Enrique, 74
tvtime, 83

uninit_device
pt::readers::Capture, 126
Universidad de Extremadura, 74
update
pt::renderers::GLWindow, 155
updateHistogram
pt::processing::EyeProcessing, 151

value
pt::math::Kriging::SPointValue, 168
variogram
pt::math::Kriging, 167
vcaps
pt::readers::CamReader, 123
vchannel
pt::readers::CamReader, 123
vergencias, 22
version
main.cpp, 222
vestibulares, 23
video
pt::eyeboard, 139
video-oculografía, 26
video_outbuf
pt::renderers::VideoWriter, 196
video_outbuf_size
pt::renderers::VideoWriter, 196
video_st
pt::renderers::VideoWriter, 196
VideoWriter
pt::renderers::VideoWriter, 194
Visio, 75
Von Helmholtz, 8
vpic
pt::readers::CamReader, 123
vwin
pt::readers::CamReader, 123

wBitsPerSample
pt::readers::AviReader::SWaveFormatEx, 111
weights
pt::math::Kriging, 167
wFormatTag
pt::readers::AviReader::SWaveFormatEx, 111
WIDTH
pt::eyeboard, 139
pt::processing::BaseProcessing, 115
pt::readers::Capture, 127
width
pt::eyeboard::SKey, 141
pt::renderers::GLWindow, 157
pt::renderers::VideoWriter, 196
widthheight
pt::math::Kriging, 167
win
pt::renderers::GLWindow, 157
window_name
pt::renderers::GLWindow, 157
wLanguage
pt::readers::AviReader::AviStreamHeader, 104
WORD
avireader.h, 205
world_canvas_pixmap
pt::eyeboard, 139
world_h
pt::eyeboard, 139
world_image
pt::eyeboard, 139
world_pixmap_array
pt::eyeboard, 139
world_processing
pt::eyeboard, 139
world_reader
pt::eyeboard, 139
world_sprite

pt::eyeboard, 140
world_w
pt::eyeboard, 140
WorldProcessing
pt::processing::WorldProcessing, 200
wPriority
pt::readers::AviReader::AviStreamHeader, 104

x
pt::eyeboard::SKey, 141
pt::math::Int2dPoint, 162
pt::math::Real2dPoint, 181
pt::renderers::GLWindow, 157
x_array
pt::math::Distribution, 130
xAt
pt::math::Distribution, 130
xawtv, 83
xioctl
pt::readers::Capture, 126

y
pt::eyeboard::SKey, 141
pt::math::Int2dPoint, 162
pt::math::Real2dPoint, 181
pt::renderers::GLWindow, 157
y_array
pt::math::Distribution, 131
Yarbus y Noton, 10
yAt
pt::math::Distribution, 130