TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Interacción con una cámara desde Processing

TITULACIÓN: Ingeniería Técnica de Telecomunicaciones, especialidad en Sistemas de Telecomunicación.

AUTOR: Sergi Martínez Ortega

DIRECTOR: Eulalia Barriere Figueroa

FECHA: 13 de julio de 2009

Título: Interacción con una cámara desde Processing

Autor: Sergi Martínez Ortega

Director: Eulalia Barriere Figueroa

Fecha: 13 de julio de 2009

Resumen

Este documento contiene el trabajo de fin de carrera “Interacción con una cámara desde Processing”, en el que se exploran varios aspectos sobre el proceso de interacción persona-ordenador mediante una cámara conectada con USB, sobre un sketch de Processing. Processing es un lenguaje creado para artistas con gran soporte para contenidos multimedia, incorpora muchas librerías que hacen relativamente fácil la creación de una aplicación multimedia. En este documento se empieza primero describiendo algunos conceptos sobre la interacción persona-ordenador así como algunas técnicas básicas de acondicionamiento de imagen y de visión por ordenador que pueden ser necesarias para interactuar con el ordenador a través de una cámara.
En el siguiente capítulo se habla de Processing y se analizan las librerías necesarias para interactuar con una cámara vía Processing, para escoger la más conveniente para usar. También se detallan algunas características de Processing y su utilización en el entorno Eclipse. Finalmente se muestra el proceso de creación de un sketch programado en Eclipse usando librerías de Processing, que interactúa con una webcam, a modo de demostración de los conceptos explicados anteriormente.

Se adjunta un CD con los ejemplos comentados en este documento, todos ellos programados con Processing para verificar y entender los métodos explicados.

Title: Webcam interaction using Processing

Author Sergi Martínez Ortega

Director: Eulalia Barriere Figueroa

Date: July, 13th 2009

Overview

This document contains the final project **"**Webcam interaction using Processing", where it is explored some human-computer interaction process aspects through a USB connected camera, in a Processing Sketch. Processing is a programming language, created for artists, with big support for multimedia content; it incorporates lots of libraries that make relatively easier multimedia application creation. This document starts describing some human-computer interaction concepts as well as some basic image conditioning techniques and some computer vision techniques that can be useful for interacting with computer via camera.

Next chapter talks about Processing, and the necessary libraries for interacting with a camera and Processing are analyzed for choosing best library for being used by us. Some Processing features and Eclipse utilization are explained too.

Finally it is shown the process for creating a Processing Sketch with Eclipse platform. This sketch interacts with a camera, for show the learned concepts.

This Project has a CD with all the examples shown in this document, all of them made with Processing for verify and understand the explained methods.

Índice

INTRODUCCIÓN

Cada vez más, las nuevas tecnologías nos permiten una interacción con el ordenador más adaptada al ser humano, haciendo más intuitivo el manejo de cualquier aplicación y permitiendo que el usuario no tenga que llevar a cabo ningún proceso de aprendizaje, o un proceso muy pequeño, para usarla. El desarrollo cada vez mayor de librerías y entornos de programación dedicados a finalidades artísticas nos permite, cada vez más, interaccionar vía micrófonos, con nuevos dispositivos hardware o cámaras. En este proyecto exploraremos la interacción por medio del uso de una cámara conectada al ordenador, de donde extraeremos la información necesaria para interactuar con nuestra aplicación. Para que el ordenador “vea” esta información necesaria para interaccionar, deberemos aplicar técnicas de visión por ordenador, por tanto también exploraremos algunas técnicas de visión por ordenador, así como el acondicionamiento de la imagen necesario para llevar a cabo estas técnicas.

Para programar esta interacción a través de una cámara, usaremos un lenguaje y entorno de programación opensource desarrollado en Java, creado para programar aplicaciones multimedia, llamado Processing. El proyecto Processing incorpora muchas librerías para todo tipo de finalidades, entre ellas nos proporciona varias para la captura de imagen desde una cámara y para llevar a cabo procesos de visión. Además también incorpora muchos métodos para la representación gráfica de formas, objetos 3D, colores etc. Por eso no nos será extremadamente difícil crear una aplicación con alto contenido visual con un objetivo creativo. Para crear esta aplicación, valoraremos las diferentes librerías de las que disponemos y escogeremos la más adecuada para nuestro objetivo.

Ambientalització: Al tratarse de un proyecto de programación, este trabajo no provocado ningún impacto medioambiental.

CAPÍTULO 1. VISION POR ORDENADOR

1.1 Interacción Persona Ordenador

La interacción persona-ordenador (IPO o HCI) es la ciencia que estudia la interacción entre personas (usuarios) y ordenadores. Principalmente su propósito es el de hacer cada vez más pequeña la barrera entre la manera que tiene el usuario para decir lo que quiere hacer a la máquina y la manera que tiene la máquina en entender esa información. Como la interacción persona-ordenador estudia el ser humano y el ordenador como un conjunto, ésta, está vinculada a varias disciplinas, en el lado del ordenador podemos encontrar temas relacionados con gráficos por ordenador, sistemas operativos o lenguajes de programación mientras que en el lado humano podemos encontrar temas como la psicología, ciencias cognitivas o teoría de la comunicación. La interacción persona-ordenador tiene lugar en la interfaz de usuario, que puede ser presentada al usuario de distintas maneras y mediante diferentes dispositivos hardware, así como el usuario también puede comunicarse con el ordenador mediante diferentes métodos y con distinto hardware. En nuestro caso abordaremos la interacción persona-ordenador mediante un dispositivo de captura de vídeo, siempre usando una interfaz gráfica (GUI).

En el sistema persona-ordenador podemos diferenciar 3 elementos principales:

- Usuario

Hay que tener en cuenta que el ser humano tiene una capacidad limitada de procesar información; lo cual es muy importante considerar al hacer el diseño. Nos podemos comunicar a través de cuatro canales de entrada/salida: visión, audición, tacto y movimiento. La información recibida se almacena en la memoria sensorial, la memoria a corto plazo y la memoria a largo plazo. Una vez recibimos la información, ésta es procesada a través del razonamiento y de habilidades adquiridas, como por ejemplo el hecho de poder resolver problemas o el detectar errores. A todo este proceso afectará al estado emocional del usuario, dado que influye directamente sobre las capacidades de una persona. Además, todos los usuarios tendrán habilidades comunes, pero habrá otras que variarán según la persona.

- Ordenador

El sistema utilizado puede afectar de diferentes formas al usuario. Los dispositivos de entrada permiten introducir información, como sería el caso del teclado del ordenador, el teclado de un teléfono, el habla o selecciones por pantalla, con el ratón o una cámara. Como dispositivos de salida contaríamos con diversos tipos de monitores, proyectores, altavoces, etc. También pueden

ser útiles otro tipo de dispositivos, por ejemplo sensores de temperatura, movimiento, etc. Respecto a la memoria, cuentan con la RAM a corto plazo y discos magnéticos y ópticos a largo plazo. El último rasgo característico es el procesamiento. El ordenador tendrá un límite de velocidad en el procesamiento, que puede ser mucho mayor que el de un ser humano.

- Interfaz de usuario

Sería el elemento que se encuentra entre los dos elementos que acabamos de describir. Es importante que haya una buena comunicación entre usuario y ordenador, por este motivo la interfaz tiene que estar diseñada pensando en las necesidades del usuario. Es de vital importancia este buen entendimiento entre ambas partes dado que sino la interacción no será posible.

En nuestro caso abordaremos la interacción persona-ordenador mediante un dispositivo de captura de vídeo, siempre usando una interfaz gráfica (GUI). Por tanto des del punto de vista del ordenador, éste se comunicará con el usuario de una manera visual mostrando por pantalla la información a transmitir. Desde el punto de vista del usuario, nuestra manera para comunicarnos con el ordenador será mediante movimiento, o posicionamiento en una escena, capturado por el ordenador a través de alguna cámara. Dado que en los seres humanos la tarea de ver es algo trivial, ya que es nuestro sentido más utilizado, el usuario solamente tendrá que aprender a comunicarse con el ordenador (saber que movimientos tiene que hacer y para qué). Por otro lado el ordenador tendrá que mandar información gráfica a la interfaz y también tendrá que poder reconocer la información que el usuario le está mandando a través de la cámara. Como esto no es tarea fácil, necesitaremos técnicas de visión por ordenador.

1.2 Visión por ordenador

La visión por ordenador, también denominada visión artificial, puede definirse como el proceso de extracción de información del mundo físico mediante imágenes utilizando para ello un ordenador.

La visión es nuestro sentido más poderoso y más complejo, en un principio se pensó que crear un sistema de visión artificial no sería demasiado difícil, ya que se pensaba que, por ejemplo, si para los humanos, resolver una ecuación diferencial es una tarea difícil mientras que para un ordenador no lo es tanto, entonces una tarea que para los humanos es trivial, como la visión, para un ordenador, lo sería más. Rápidamente se llegó a la conclusión de que la visión no tenía por qué ser tan fácil para un ordenador, ya que, mientras que los humanos somos conscientes del razonamiento y etapas necesarios para solucionar una ecuación diferencial, desconocemos las etapas y procesado que se lleva a cabo en el proceso de la visión humana, ya que ésta se lleva a cabo de una manera inconsciente.

En un sistema de visión por ordenador podemos distinguir varias etapas, captura, preprocesado y detección, que se describen en los siguientes apartados.

1.2.1 Captura

La primera etapa de todas, será la de adquirir la imagen, digitalizarla y transmitirla a nuestro ordenador, donde será procesada, para ello necesitaremos una cámara, existen varios tipos de cámaras con diferentes estándares de salida de vídeo, cada uno con sus características.

Podemos encontrar cámaras con salida USB, actualmente el formato más usado de forma doméstica para entrada/salida de datos desde un PC. La transmisión es en serie y su principal ventaja es la compatibilidad con cualquier PC, aunque puede ser lento para según qué aplicación.

Por otro lado podemos comunicar cámara y ordenador vía firewire. Es un estándar multiplataforma para entrada/salida de datos en serie a gran velocidad y fue desarrollado por Apple. Es mucho más rápido que USB aunque necesitaremos una cámara que lo soporte.

En nuestro caso, la mejor cámara que hemos podido conseguir es una Samsung digital modelo VP-D453, tiene salida USB y vídeo compuesto, que es una señal de video analógico muy común en equipos domésticos y tiene dos componentes.

1.2.2 Preprocesado

En un sistema de visión artificial, la etapa de preprocesamiento tiene como finalidad producir otra imagen de mayor calidad que simplifique y facilite posteriores etapas. Por tanto el objetivo no es extraer información de la imagen, sino actuar sobre ella de manera que podamos compensar los defectos de la iluminación y efectos de ruido de todo tipo.

Para ello necesitaremos conocer algún concepto, como qué es un histograma.

Histograma: Es una representación gráfica de la frecuencia con la que los niveles de gris, o los niveles de cada componente de color, aparecen en una imagen. El eje de abscisas nos indica los distintos niveles discretos de grises y en el de ordenadas la frecuencia o el número de píxeles con ese nivel de gris en la imagen. En la siguiente figura tenemos un ejemplo de varios tipos de histogramas.

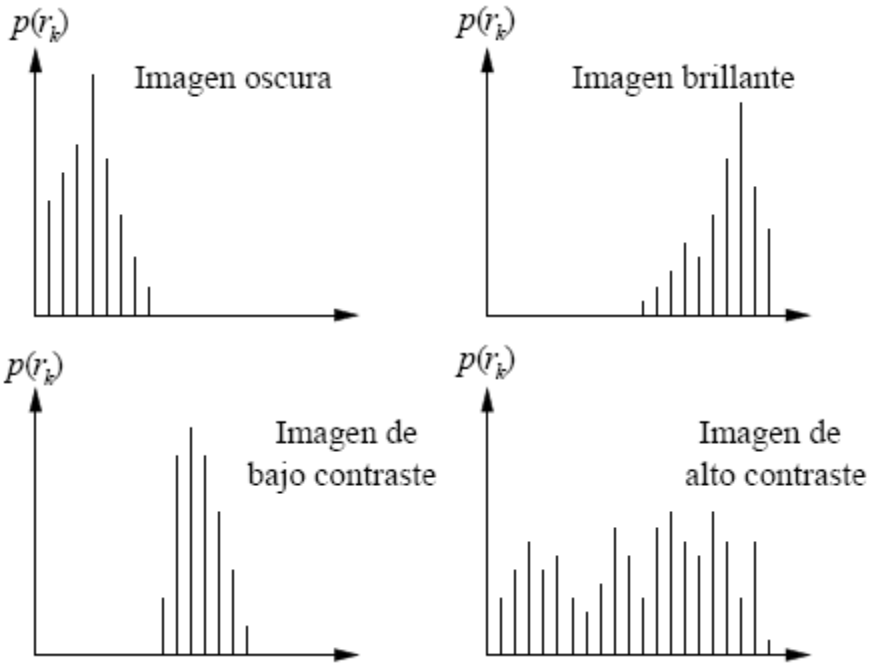

Fig. 1.1. Ejemplo de diferentes histogramas

El histograma nos proporciona información de cómo están distribuidos los distintos niveles de gris en la imagen, esto nos será útil para ver si la imagen tiene un buen brillo, contraste, decidir un valor de umbralización, etc.

Teniendo el histograma de una imagen, podremos hacer una serie de transformaciones en la imagen para que ésta quede más practicable para la siguiente etapa, a continuación explicamos algunos de estos métodos.

1.2.2.1 Igualación del histograma

Este método normalmente incrementa el contraste global de la imagen, especialmente si la imagen está usando pocos valores de gris y muy cercanos entre ellos, este ajuste el valor de los niveles de gris podrá ser distribuido mejor a lo largo del histograma. Se define como:

$$p(r_k) = \frac{n_k}{n}$$

Donde:

r_k es el k-ésimo nivel de gris.
n_k es el número de píxeles con tal nivel de gris.
n es el número total de píxeles.

Este método irá sumando la probabilidad,$p(r_k)$, acumulada de cada valor de gris y asignando su valor correspondiente según los niveles de gris en el sistema.

Ver ejemplo 1 del CD adjunto.

Fig. 1.2. Izquierda: Imagen Original. Derecha: Imagen Resultante de la transformación.

1.2.2.2 Transformaciones punto a punto

Se denominan transformaciones punto a punto a todos aquellos métodos destinados a mejorar la imagen en los que el valor del píxel resultante sea función del píxel original.

Normalmente esta transformación viene dada por una función discreta definida sobre los números naturales comprendidos entre cero y el número de niveles de gris que tengamos. Define una transformación píxel a píxel para la imagen a procesar. Esta función nos será útil para, por ejemplo, aprovechar al máximo nuestros niveles de gris, arreglar malas iluminaciones, etc.

Al aplicar éste método me he encontrado con que este método presenta el problema que al usarlo en imágenes de color no podemos predecir con exactitud qué color será el resultante, ya que al aplicar la transformación a tres capas distintas de color, el color final variará considerablemente. Por tanto, reservaremos este método solo para imágenes sobre tonos de gris.

Ver ejemplo 2 del CD adjunto.

Fig. 1.3. Izquierda: Imagen Original. Derecha: Imagen con una transformación de x/2

1.2.2.3 Operaciones espaciales

En estas transformaciones el valor del píxel resultante de la transformación depende del píxel original así como también de los píxeles vecinos de dicho píxel. Para ello se define una máscara, que es una matriz de coeficientes obtenidos de diferentes formas, dependiendo del resultado que se quiera conseguir, capacidad de computación etc. Esta máscara se convoluciona con la matriz de píxeles para obtener una nueva imagen.

En este apartado veremos dos implementaciones de este tipo de transformaciones, que detallamos a continuación.

Filtro pasa bajas

El filtro pasa bajas es un filtro de promediado, por eso los coeficientes de la máscara serán todos positivos. Depende de la implementación que hagamos, podemos dar más o menos importancia al píxel central respecto a los vecinos. Es normal que los coeficientes de la matriz se normalicen de tal manera que su suma sea igual a uno, así la ganancia del filtro a frecuencias nulas (por ejemplo, zonas homogéneas) será uno y la componente continua no se verá afectada.

El objetivo de este filtro es el de reducir posible ruido de adquisición, dado que puede eliminar píxeles sueltos que varíen mucho respecto a sus vecinos. El inconveniente de este filtro es que suaviza los cambios más bruscos de color, ya que elimina las componentes espectrales de mayor frecuencia, de forma que el aspecto final de la imagen se percibirá ligeramente desenfocada, más desenfocada contra mayor sea la máscara utilizada.

En el ejemplo de la figura 1.4 se ha utilizado un filtro pasa bajas con la siguiente máscara.

$$1/16 \times \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Como entrada del filtro usamos la salida de una cámara conectada al ordenador mediante USB, a la que añadimos algo de ruido “de nieve”. Tras aplicar el filtro comprobamos que éste suaviza mucho los bordes y empieza a eliminar los píxeles erróneos. Para eliminarlos del todo deberíamos usar una máscara mayor, aunque esto nos suavizaría aun más los bordes. Otra opción puede ser el filtro de mediana, en el que en vez de hacer una media de los píxeles vecinos, se hace una mediana, en este caso eliminamos efectos de ruido sin suavizar tanto los bordes.

Ver ejemplo 3 del CD adjunto.

Fig. 1.4. Entrada del filtro pasa bajas (izquierda), salida del filtro pasa bajas (derecha)

Filtro pasa altas

Los filtros pasa altas acentúan las transiciones, algo muy útil para la visión por ordenador, porque de esta manera podremos detectar bordes de objetos, cambios bruscos de color, etc.

En este tipo de filtro, la máscara contendrá coeficientes positivos y negativos, y normalmente su suma será nula, de forma que la respuesta del filtro para zonas uniformes (frecuencias nulas) será nula. Esta máscara puede ser obtenida de diferentes maneras: gaussiana, primera o segunda derivada, etc. Pueden ser direccionales, si queremos realzar especialmente los bordes en una dirección.

En el siguiente ejemplo hemos usado también la salida de una cámara para pasarla por el filtro pasa altas, en este caso hemos usado la siguiente máscara:

1/8 x

-1	-1	-1
-1	8	-1
-1	-1	-1

Esta máscara corresponde, a la aproximación discreta de la laplaciana de una función escalar. Es por tanto un filtro de segunda derivada. En la figura 1.5 podemos apreciar los efectos de este filtro, como puede verse, las zonas de color homogéneo son filtradas (su valor es cero) de modo que resultan resaltados las transiciones de color.

Ver ejemplo 4 del CD adjunto.

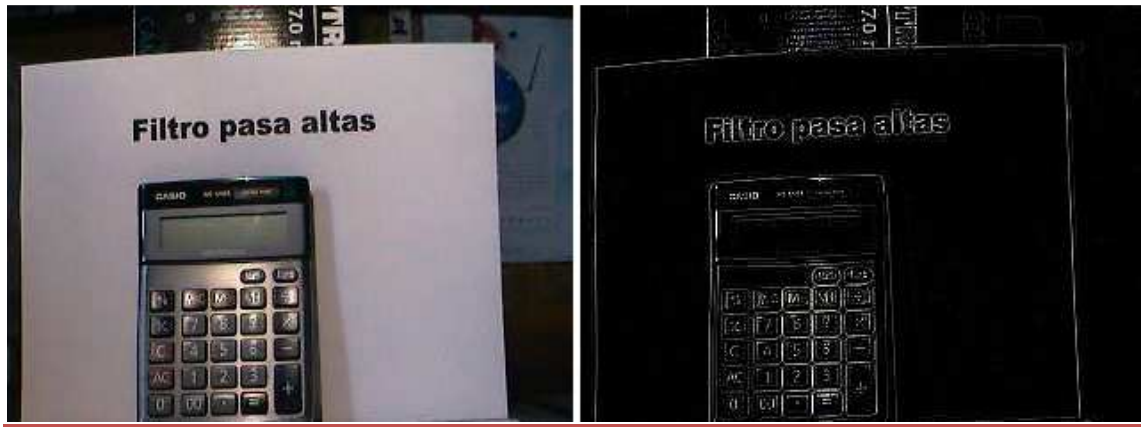

Fig.1.5. Izquierda: entrada del filtro, derecha: salida del filtro, podemos ver como enfatiza los bordes.

El siguiente filtro, sin embargo, es más sencillo, se trata de un filtro pasa altas direccional. Éste solo enfatiza los bordes en una dirección, como podemos ver en la figura 1.6. Su máscara es la siguiente:

-1	*0*	*1*
-1	*0*	*1*
-1	*0*	*1*

Podemos comprobar la diferencia con el filtro anterior, éste solo remarca las transiciones de izquierda a derecha. Podemos combinar una serie de filtros de este tipo, uno para cada dirección, para obtener una detección de todos los bordes.

Ver ejemplo 5 del CD adjunto.

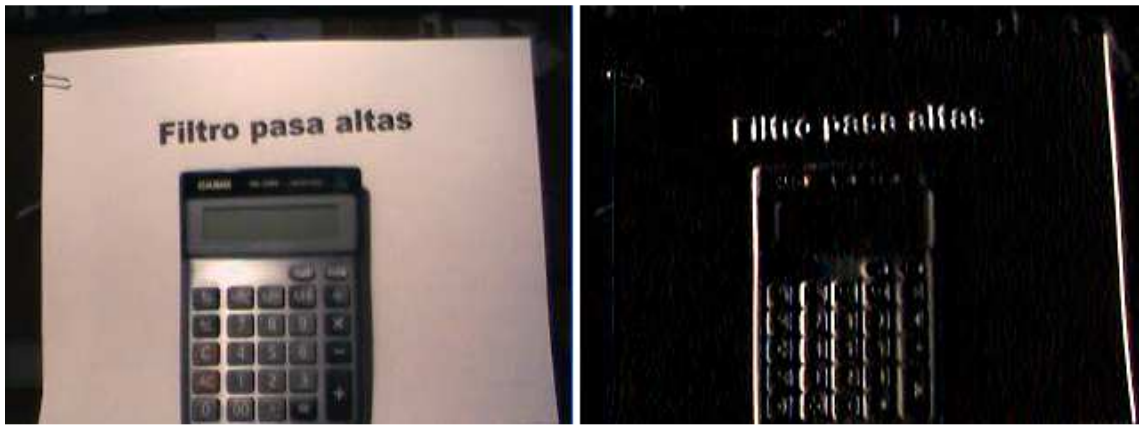

Fig. 1.6. Izquierda: entrada del filtro, derecha: salida del filtro, se puede comprobar que remarca los bordes izquierdos.

Podemos ver también que este filtro detecta durante más tiempo las transiciones (línea de transición más ancha), esto es debido a los factores de los vecinos del píxel, en el caso del filtro laplaciano los factores de los vecinos eran menores que en el caso de este filtro. Podemos someter a este filtro a una etapa de umbralización para obtener una salida más “limpia”.

1.2.3 Detección: color tracking, motion, reconocimiento de patrones.

Una vez acondicionada la imagen podemos proceder a hacer las detecciones necesarias, básicamente podremos clasificarlas en los tres tipos siguientes.

Detección de color o color tracking: En ella básicamente se busca un color o nivel de luminancia determinado en la escena. Con color tracking podemos hacer más cosas que el mero hecho de saber si un color está en la escena o no. Por ejemplo, podemos usarlo como un detector de objetos si nos aseguramos que dichos objetos son de un color que no aparecerá en la escena.

El problema que he encontrado con esta técnica es que es realmente sensible a la iluminación de la escena, ya que la iluminación es un factor importante a la hora de que la cámara asigne un valor a un píxel. Podemos corregir medianamente este problema creando alguna función de distancia entre colores. Sería como decirle al ordenador “busca este nivel de rojo y todos los rojos que se parezcan en un nivel x”, aunque si la iluminación es realmente mala será muy difícil poder hacer un buen color tracking.

Ver ejemplo 6 del CD adjunto.

Detección de movimiento o motion detection: También podemos hacer una detección del movimiento, en la que detectaremos cualquier movimiento que tenga lugar en la escena. Tenemos varias maneras y criterios para detectar movimiento, el primero, y más sencillo, es el de hacer una detección en función de una imagen base denominada background o fondo. Este tipo de método puede ser útil en sistemas de vigilancia por ejemplo, en el que realmente no queremos detectar el movimiento como tal, sino que lo que buscamos es detectar el movimiento respecto a un fondo inicial, también puede ser que ese fondo vaya cambiando a partir de algún evento en la escena o en la ejecución.

Podemos hacer una detección de movimiento restando el frame actual al frame previo. El resultado nos dará la diferencia, por tanto el movimiento que ha habido entre un frame y otro. Este método es muy simple y no nos indica la dirección del movimiento, ni su velocidad, simplemente nos indica que píxeles son distintos respecto a la imagen anterior.

Por último la manera más eficiente de detectar movimiento es la de coger cada píxel y buscarlo en el frame siguiente. Habrá píxeles que no se moverán, pero en cambio otros si lo harán, podremos buscar dónde han ido en el frame siguiente y de esta manera con los dos puntos podremos determinar la dirección del movimiento, velocidad y otros valores similares.

Al ver este método puede parecer que falla al detectar un objeto con un color uniforme, ya que los píxeles de su interior no se detectarán como movimiento al ser ocupada su posición por un píxel del mismo color. Puesto que en el nuevo frame su posición ha sido ocupada por un píxel del mismo color perteneciente al mismo objeto. El sistema tendrá la sensación que ese píxel no se ha movido, por tanto no existe movimiento. Esto no tiene por qué ser un problema ya que lo que queremos es detectar movimiento de objetos y no tiene mucho sentido querer detectar el movimiento de un píxel. Finalmente si queremos saber hacia dónde se dirige nuestro objeto, podemos hacer una media de las direcciones de todos los píxeles que se han movido, obteniendo así una media global para todo el objeto.

Ver ejemplo 7 del CD adjunto.

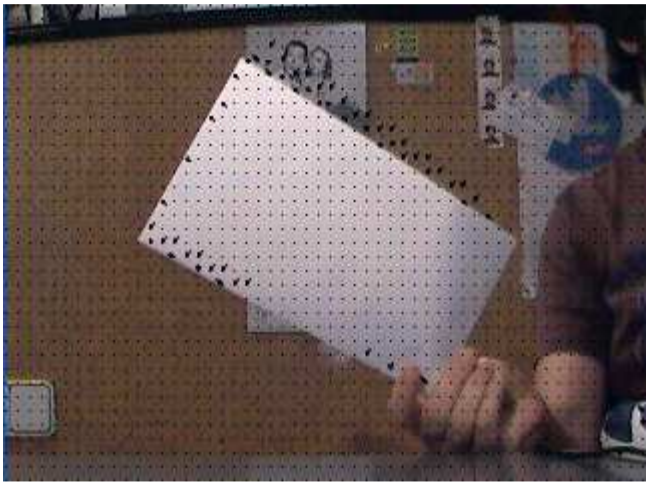

Fig. 1.7. Vectores de movimiento señalando dirección del movimiento

Reconocimiento de patrones o pattern recognition: Una de las formas de visión artificial más robusta podría ser el reconocimiento de patrones, en el que la aplicación, por medio de conocimientos previos y conocimientos estadísticos adquiridos puede llegar a reconocer formas, letras, etc. Aunque este tipo de técnicas no se ciñen solamente a temas de vídeo, también pueden usarse en audio, filtros anti-spam etc.

Un ejemplo de detección de patrones puede ser la detección de caras, para hacer esta detección se aplican una serie de filtros a toda la imagen, estos filtros son los encargados de buscar en la imagen diferentes características de la cara humana, como pueden ser los ojos, nariz, boca, etc. Cuando son

aplicados, estos filtros devuelven un valor, aquí es donde tendremos que ajustar de una forma empírica las etapas que deciden si se está viendo una cara o no.

Ya que la finalidad de este proyecto no es la de programar un sistema de reconocimiento de patrones no se ha programado ningún ejemplo, puesto que esto podría ser un proyecto ya por sí solo.

CAPÍTULO 2. PROCESSING

2.1 Que es Processing?

Processing es un proyecto opensource desarrollado por dos alumnos del "Aesthetics and Computation Group" del Massachusetts Institute of Technology, Ben Fry y Casei Reas. Es un lenguaje y entorno de programación multiplataforma basado en java que incluye una serie de librerías destinadas a crear aplicaciones con gran contenido visual e interactivo. Processing nace de otro proyecto creado por John Maeda, "Design by numbers" en el que se planteaba la necesidad de un lenguaje potente pero sencillo para personas relacionadas con el mundo del arte y el diseño pero interesadas en el mundo de la programación, interacción etc. De manera que pudieran crear aplicaciones sin necesidad de grandes conocimientos de programación e informática.

El entorno de programación de Processing está bajo los términos de licencia pública GPL, y las librerías del "core" bajo licencia LGPL, hecho que entre otras cosas nos permitirá modificar las librerías si lo creemos necesario, y que ha permitido a Processing llegar a muchas más personas que si hubiera sido un programa cerrado o incluso de pago. Desde la aparición del proyecto en 2001 se le han ido sumando seguidores que han ido creando una serie de librerías y proyectos hermanos, creando librerías para trabajar con audio, vídeo, formas 3D, diferentes protocolos de comunicación y un largo etcétera de utilidades diversas. Entre los proyectos hermanos de Processing podemos destacar "Wiring" o "Arduino" que son dos proyectos en los que se ha adaptado Processing para programar una plataforma de hardware basada en una placa integrada con un microcontrolador, también se puede destacar "Processing mobile", para teléfonos móviles que soporten Java.

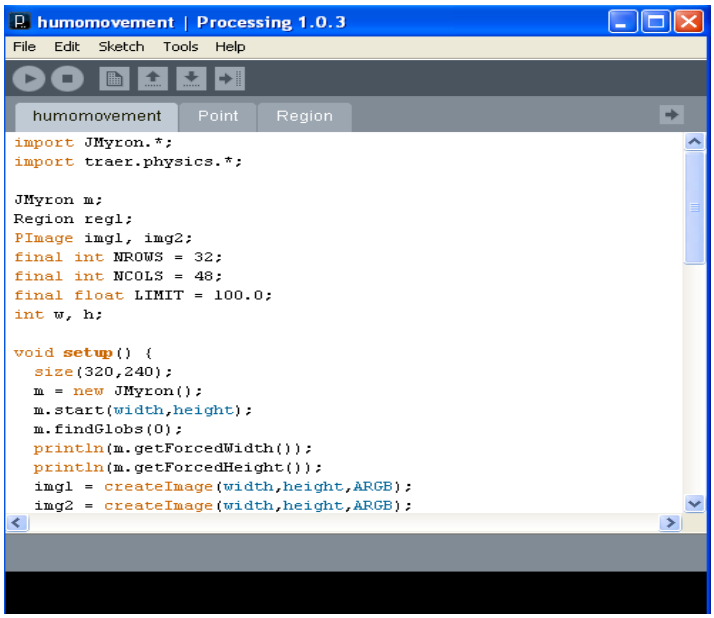

Fig. 2.1. Entorno de programación de Processing

El hecho de que Processing este escrito en Java nos da una gran versatilidad, ya que si hay alguna funcionalidad que no tenemos, o que no cumple nuestras expectativas, podremos crearnos una librería en Java para ello, además de una manera muy fácil. También, si no nos convence el entorno de programación de Processing, podremos usar Eclipse, ya que el entorno que nos proporcionan en Processing no dispone de debugger, ni de explorador de proyectos, entre otras cosas.

En nuestro caso vamos a usar eclipse como entorno de programación, en el que importaremos las librerías necesarias de Processing, de esta manera podremos tener un debugger y usar una serie de características de eclipse que nos ayudarán con nuestro proyecto.

2.2 Interacción en Processing

Processing nos da muchas posibilidades de interacción con el ordenador, incorpora todo tipo de librerías que nos permiten interaccionar con diferentes elementos. De entrada podemos interactuar con el hardware típico de un ordenador, por ejemplo con teclado y ratón. Además hay librerías para interaccionar con infinidad de hardware, por ejemplo el mando de la wii, los sensores de movimiento de los MacBooks, o con cámaras.

Además el proyecto paralelo a Processing, “arduino” nos permitirá conectar a Processing casi cualquier hardware, circuito o sensor que podamos crear ampliando así las posibilidades de interacción con el ordenador mediante Processing.

2.3 Librerías

Como hemos comentado antes, Processing incorpora una serie de librerías para trabajar con vídeo, muchas de ellas en principio fueron pensadas principalmente para trabajar con vídeos hechos y no para capturar desde una cámara, por tanto no tienen demasiados métodos que ayuden a implementar un sistema de visión, aun así, pueden ser útiles por su velocidad al tratar el vídeo.

Cada librería es distinta a la hora de capturar vídeo desde una cámara, algunas necesitan algún software especial otras usan DLLs especificas o están escritas en diferentes lenguajes.

La siguiente figura es un esquema general de las librerías de Processing.

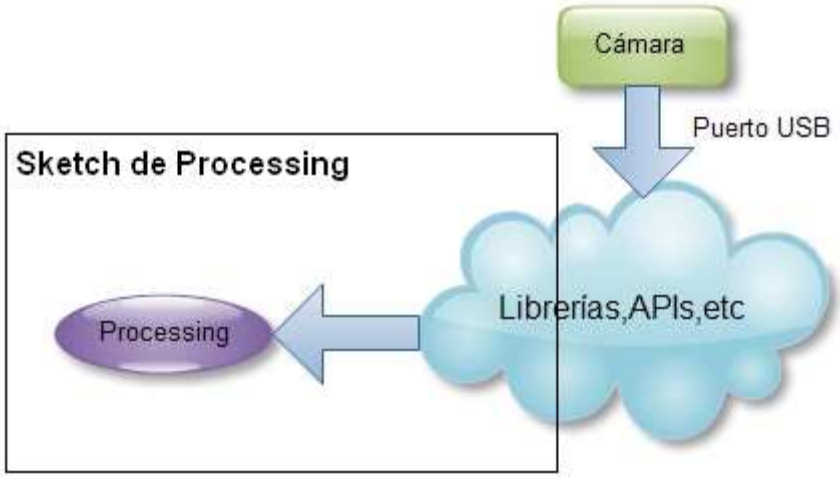

Fig. 2.2. Esquema de funcionamiento de una librería genérica.

2.3.1 Vídeo

La librería vídeo es una librería integrada en Processing, que nos permite cargar vídeos .mov en un sketch de Processing, crear archivos .mov a partir de un sketch, o capturar frames desde un dispositivo externo de captura, como una cámara. Al estar integrada en Processing, no necesitaremos ni descargarla ni instalarla, simplemente deberemos incluirla en el sketch mediante un import. Se compone de tres clases diferentes, con métodos acordes a la utilidad de cada clase: clase para cargar vídeos, clase de captura desde una cámara o clase de creación de vídeo. En nuestro caso, nos centraremos en describir la clase “Capture”, que es la que tiene relación con nuestro proyecto.

Esta librería usa funciones de QuickTime para funcionar, por tanto necesitaremos tener instalado el reproductor QuickTime, esto hace que la librería de vídeo no funcione bajo Linux, y que, para utilizarla sobre Windows tengamos que instalar un software digitalizador/codificador de vídeo compatible con QuickTime, ya que la mayoría de cámaras no soportan directamente QuickTime. Un software que nos servirá para esta tarea es winVDIG, que permite a QuickTime capturar vídeo de diferentes fuentes, tarjetas PCI, cámaras USB, o DV.

En nuestro caso se utilizó la versión de winVDIG 1.0.1, ya que las otras versiones no funcionan correctamente con la librería vídeo de Processing.

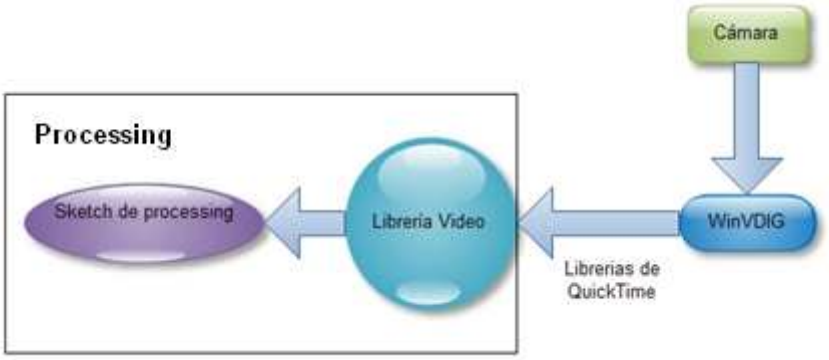

Fig. 2.3. Esquema del funcionamiento de la librería Vídeo.

A continuación mostramos los métodos que nos proporciona esta librería y pasamos a valorar su utilidad en nuestro proyecto.

String[] Capture.list(): Devuelve una lista de los nombres de los diferentes dispositivos de captura disponibles. Puede ser útil si se emplea un sistema con varias cámaras.

Void format (capture cam, string MODE): Fija el formato de vídeo que queremos que sea el estándar para el digitalizador de vídeo, las opciones son: NSC, PAL, SECAM.

Void settings(): Abre la ventana para la configuración de la cámara.

Void capture(): Lee el frame actual de la cámara, esta función será de las más usadas, nos servirá para cargar la imagen capturada por la cámara en el sketch de Processing.

Bool available(): Devuelve “true” cuando hay un frame nuevo de vídeo disponible.

Void frameRate(): Fija el número de frames por segundo que se leerán de la cámara.

Void crop(int x, int y, int width, int height): Recorta la imagen. Coge solamente la imagen contenida en el rectángulo (x,y) con ancho width y altura height. Algunas cámaras introducen líneas que no queremos arriba o debajo de la imagen, esta función nos será útil para quitarlas.

Void noCrop(): Deja de aplicar el crop definido.

Void stop(Capture cam): Deja de capturar del dispositivo “cam”.

Como podemos ver por los métodos de la librería, ésta está pensada solo para capturar desde un dispositivo, no incorpora ningún tipo de función de tratamiento de imagen o de visión. Si la comparamos con las otras librerías de captura de Processing, la librería vídeo no nos aporta nada que las otras no

nos aporten, además de precisar de la instalación de QuickTime y winVDIG para funcionar.

2.3.2 GSVideo

GSVideo es una librería desarrollada especialmente para Processing, que nos aporta métodos para capturar, importar y crear vídeo desde un sketch de Processing. La finalidad de ésta es aportar una librería basada completamente en software de código abierto al conjunto de librerías de Processing. En un futuro se pretende que esta librería substituya a la librería Vídeo que incorpora Processing por defecto, ya que ésta necesita QuickTime para su funcionamiento.

GSVideo utiliza la librería GStreamer para manejar vídeo. GStreamer es un Framework libre multiplataforma, escrito en C, pensado para crear aplicaciones multimedia. En este caso, GSVideo utiliza concretamente GStreamer-Java, un set alternativo basado en GStreamer para entornos Java.

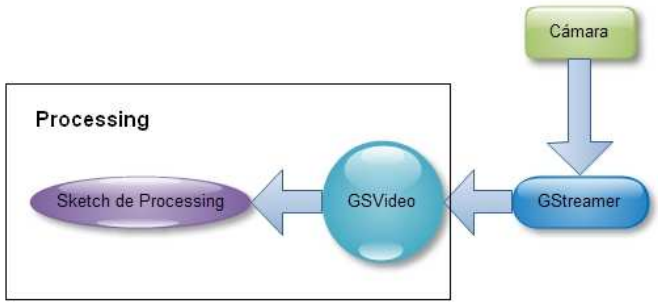

Fig. 2.4. Esquema del funcionamiento de la Librería GSVideo

Inicialmente para usar la librería GStreamer era necesario tener instalado algún software tipo GStreamer-Win, para poder usar las funciones de codificación de vídeo o audio, aunque en las últimas versiones de GSVideo incorporan ya las librerías de GStreamer.

En la librería GSVideo podemos encontrar todo tipo de métodos para capturar, cargar y crear vídeo, tenemos una clase para cada una de estas finalidades:

GSCapture: Clase para capturar y manipular vídeo desde una cámara. Con métodos como “read” “stop”, etc. Tiene métodos muy parecidos a la clase capture de la librería Video, pero tiene una velocidad mayor a la hora de gestionar el video.

GSMovie: Con esta clase podemos cargar vídeos en el sketch y reproducirlos de diferentes maneras. Con métodos como “loop”, “pause” ,“speed” etc.

GSMovieMaker: Esta clase nos permite crear vídeos a partir de algún sketch en ejecución.

GSPipeline: Nos permite crear nuestros propios Pipelines de GStreamer. Estos pipelines son como cadenas de elementos, en los que cada elemento puede tener una función, con estas pipelines por ejemplo, podríamos por ejemplo pasar un AC3 (dolby digital) a un mp3.

GSPlayer: Clase que permite cargar cualquier tipo de contenido multimedia remoto a través de playbin (un pipeline Gstreamer ya creado), éste no tiene porque ser un vídeo.

GSVideo: Contiene algunos métodos básicos que usan el resto de clases, como por ejemplo el path donde se encuentra el GStreamer.

2.3.3 JMyron

Jmyron es una librería multilenguaje, multiplataforma escrita en C++ que nos permite capturar imagen desde un dispositivo externo (cámara) con Processing. Principalmente incorpora métodos de visión.

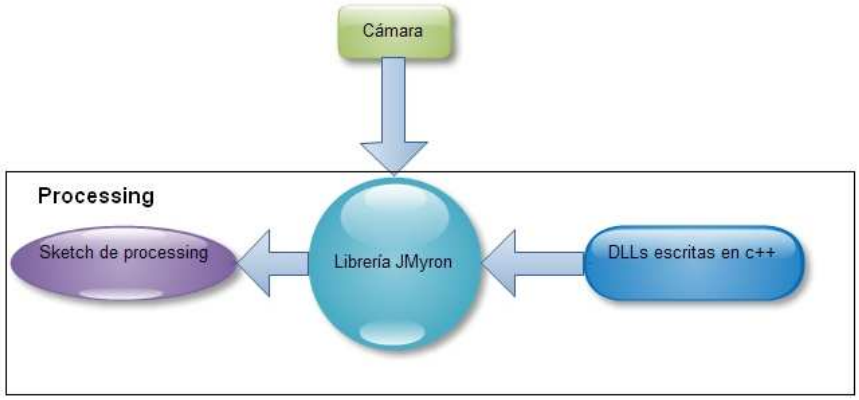

Fig. 2.5. Esquema del funcionamiento de la librería

Una particularidad de esta librería es que nos puede devolver diferentes capturas.

cameraImage() or image() Nos devuelve la imagen capturada por la cámara, en este caso no hace nada, simplemente captura.

globsImage() Nos devuelve una imagen con los globs capturados, en las especificaciones de la librería se recomienda para debugar código, pero para nada más.

differenceImage() Devuelve la imagen diferencia, nos puede ser útil para calcular movimiento, extraer el fondo etc.

Otra utilidad de esta librería es que nos proporciona una serie de métodos de detección de “globs”, estos son áreas con grupos de pixeles con color y luminancia parecida, que podemos utilizar para extraer el fondo, hacer tracking de algún color, etc.

Int[][2] GlobCenters() : Devuelve una de puntos que son los centros de cada glob encontrado.

Fig. 2.5. Ejemplo de GlobCenters

Int[][4] GlobBoxes() : Devuelve una lista de 4 puntos por cada glob, que son los vértices del cuadrado que contiene completamente ese blob.

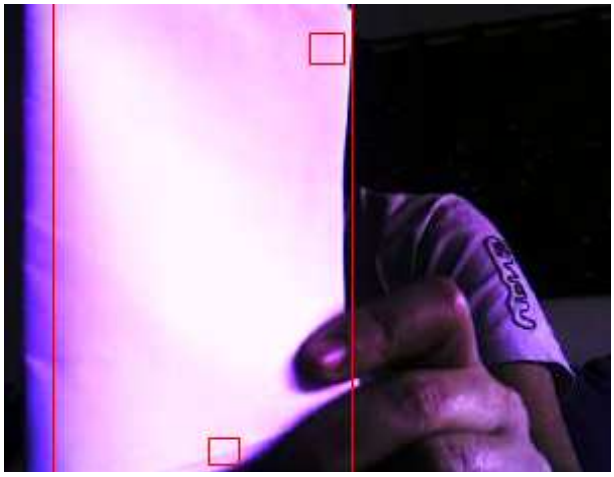

Fig. 2.6. Ejemplo de GlobBoxes

int[][][2]globEdgePoints(int segmentLength) Devuelve una lista de segmentos que envuelven cada glob, con el parámetro segmentlenght fijamos la longitud de dichos segmentos**.**

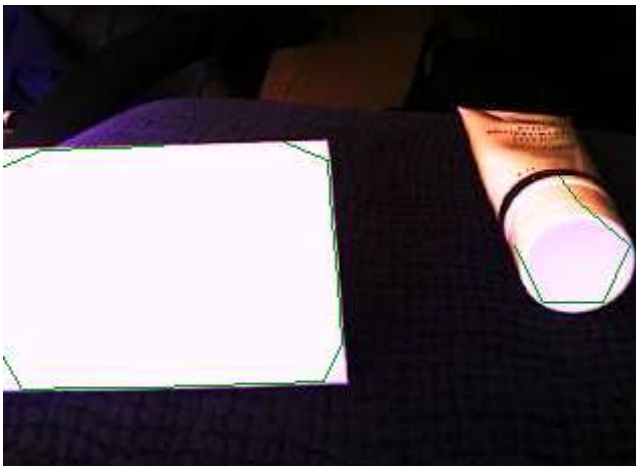

Fig. 2.7. Ejemplo de GlobEdgepoints

int[][][2]globPixels() Devuelve una lista de puntos que son los que envuelven cada glob.

Fig. 2.8. Ejemplo de GlobPixels

minDensity(int val) and maxDensity(int val) Con estas funciones podremos cambiar el número máximo y mínimo de píxeles necesarios para que los métodos anteriores acepten el glob como válido.

Void trackColor(red,green,blue,tolerance) y trackNotColor(red,green,blue,tolerance) : Con estos metidos fijaremos el color y la tolerancia para las funciones de detección de globs.

sensitivity(float value) Nos permite cambiar la sensibilidad de la detección de los globs.

void adaptivity(float val) Fija la velocidad a la cual la imagen retina se adapta.

Con cero (por defecto) no habrá adaptivity. La retina es donde se guardan los frames anteriores, que pueden ser útiles para calcular la imagen diferencia, por ejemplo.

void adapt() Fijamos el valor de retinal image al valor actual de la imagen de la cámara, se usa para reinicializar el valor de la imagen retina.

2.3.4 TUIOclient

TUIOclient es un framework de código abierto que define una API y un protocolo pensados inicialmente para interfaces táctiles. Permite la transmisión desde una aplicación de tracking(la que se encarga de adquirir la señal de vídeo, detectar objetos, etc.) ha otra aplicación capaz de entender dicho protocolo. En esa transmisión se envían posición y estado de los objetos detectados, globs, etc. Hay un número creciente de programas que se dedican a hacer dicho tracking, aunque para Processing usaremos reacTIVision. El protocolo que usa TUIO para comunicar la aplicación de tracking con Processing está basado en OSC (open sound control), un estándar para entornos interactivos, aunque por su nombre parezca lo contrario, este estándar no está limitado solo al audio.

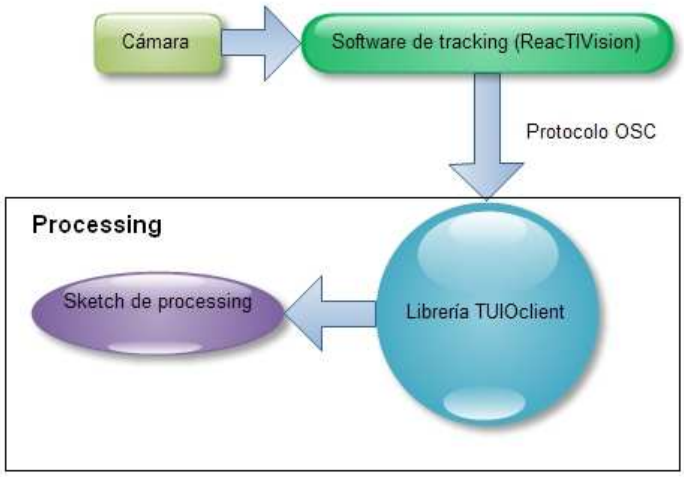

Fig. 2.9. Esquema de funcionamiento de la librería

Para usar esta librería en Processing necesitaremos un programa de tracking, en este caso reacTIVision. También disponemos de un simulador (ver fig.6) para hacer el tracking, por si no tenemos cámara o queremos hacer pruebas. También necesitaremos la librería TUIO. Ésta nos permite detectar tamaño y posición de globs, y detectar las propiedades de una serie de objetos llamados fiducials, cada uno con su id, de manera que podremos otorgar a cada objeto diferentes funciones.

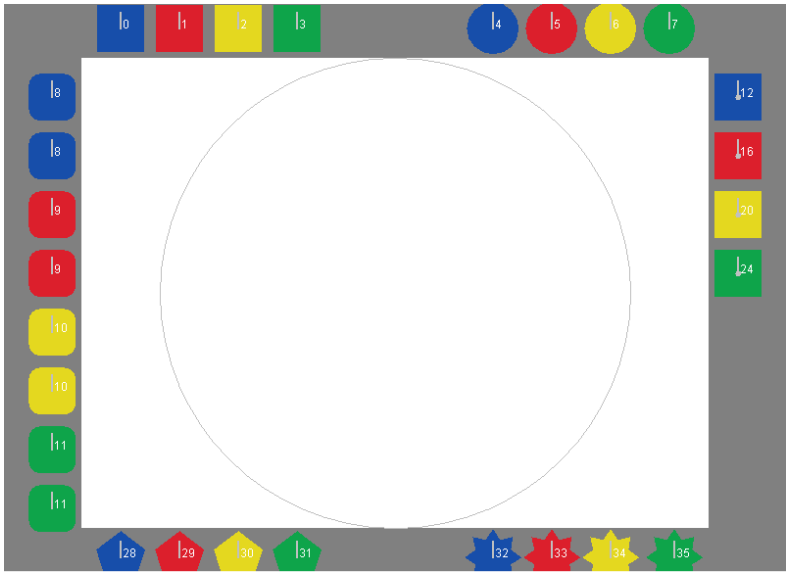

Fig. 2.10. Simulador TUIO

Fig. 2.11. Ejemplo de fiducial

Esta librería proporciona varias clases para JAVA usadas para manejar objetos, cursores etc.

TuioClient : Clase encargada de recibir y decodificar la información transmitida por la aplicación de tracking. Para recibir mensajes TUIO esta clase tiene que ser creada.
TuioCursor : Clase para describir cursores(detectados mediante globs), incluye métodos de todo tipo, como velocidad del cursor, aceleración, posición, ID, etc.

TuioObject : Clase que describe propiedades y métodos para actuar sobre objetos (fiducials) tenemos también muchos métodos y campos como aceleración, ángulo, posición etc.

TuioPoint : Creada para gestionar posiciones de la librería en general, y además es la clase contenedora de TuioCursor y TuioObject.

TuioTime: Clase que describe el tiempo que ha pasado desde que empezó el programa, tiene métodos tipo, “GetSessionTime” o “Reset”.

TuioListener : TUIOclient implementa una serie de eventos que nos permiten gestionar diferentes situaciones posibles durante la ejecución del programa, esta interface que tiene definidos los eventos para la librería TUIO, estos son los eventos implementados:

- **addTuioObject(TuioObject tobj)** es llamado cuando un objeto TuioObject pasa a ser visible.
- **removeTuioObject(TuioObject tobj)** llamado cuando un objeto se saca de la escena o de la superficie.
- **updateTuioObject(TuioObject tobj)** llamado cuando un objeto cambia de estado (posición, rotación, etc.).
- **addTuioCursor(TuioCursor tcur)** se llama cuando se detecta un Nuevo cursor.
- **removeTuioCursor(TuioCursor tcur)** se llama cuando un cursor deja de ser visible
- **updateTuioCursor(TuioCursor tcur)** llamado cuando el cursor cambia de estado.
- **refresh(TuioTime bundleTime)** se llama después de cada frame.

Las clases descritas anteriormente están creadas para trabajar con Eclipse, en el caso de la librería para Processing se ha creado solo una clase simplificando así su utilización, que describe una mezcla de los métodos más útiles de casa clase.

2.3.5 BlobDetection

BlobDetection es una librería creada inicialmente solo para Processing, aunque puede ser utilizada en cualquier programa escrito en Java. Esta creada para detectar “globs”, que como hemos comentado ya son zonas de píxeles con color y luminancia parecidos.
Esta librería, a diferencia de todas las que hemos visto anteriormente, no captura imagen desde la cámara, por tanto necesitaremos usar otra librería de captura para poder usar BlobDetection.

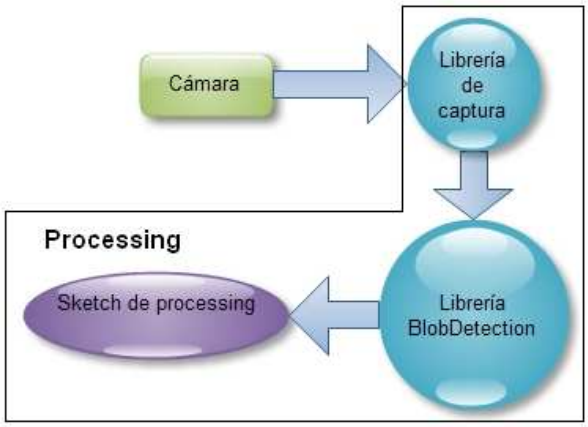

Fig. 2.12. Esquema de funcionamiento de la librería

BlobDetection está compuesta por 4 clases:

Clase EdgeVertex: Esta clase no tiene ningún método, se compone sólo de dos campos, “x” e “y”, que representan las coordenadas de los puntos de los bordes(blobs). Todas las coordenadas de esta librería están normalizadas, es decir sus valores oscilan entre 0 y 1, para acceder a la coordenada real solo tendremos que multiplicar por el width o el height.

Clase BlobTriangle: Contiene los vértices del triangulo. Cada blob incorpora una lista de triángulos que definen un “polígono blob”, esta lista solo se rellenara si llamamos al método computeTriangles() al inicializar el objeto.

Clase Blob : Esta clase contiene campos como coordenadas del blob, tamaño, y coordenadas máximas y mínimas. Así como métodos de acceso a las coordenadas de los puntos del blob. También, han incorporado recientemente otros métodos que devuelven los puntos de los triángulos de cada blob.

Clase BlobDetection: Clase encargada de detectar los blobs, en ella encontramos métodos para decidir si queremos blobs oscuros o brillantes, fijar el umbral de detección, o definir eventos cuando se detecta un nuevo glob.

2.3.6 ColorMatcher

Esta librería ha sido desarrollada por Nikolaus Gradwohl, creada para copiar a Processing alguna de las ventajas que proporciona la librería “Touchless SDK” de Microsoft. Es una sencilla librería pensada para crear aplicaciones de color tracking, en ella solo se implementa una clase “ColorMatcher” con un solo método muy útil para detectar color:

Point[] matchColors(Pimage img, color[] colors): Busca en la imagen “img” los colores dentro del vector “colors” y devuelve sus coordenadas como un vector de puntos.

2.4 Usando Eclipse para programar con Processing

Como hemos comentado, una manera de potenciar Processing es usarlo mediante Eclipse, ya que esto nos introduce una serie de ventajas respecto al entorno de Processing, además de ser un entorno más amigable para el programador. En este apartado explicaremos cómo crear un sketch de Processing en el entorno eclipse.

Para usar Processing en el entorno eclipse primero deberemos crear un proyecto Java, después importar las librerías de Processing necesarias, para un sketch sencillo que no use librerías especiales tan solo necesitaremos importar la librería “core.jar” alojada en la carpeta de Processing e incluirla en el “build path”.

Una vez hecho esto, procederemos a crear una clase, que será nuestro sketch. Para que eclipse sepa que esta clase es un sketch de Processing, deberemos hacer que nuestra clase extienda la clase “PApplet”, esta clase es la clase base para todos los sketch de Processing, está contenida en el “core.jar” y extiende a la clase java.applet.Applet, y a su vez implementa una serie de interfaces enumerados a continuación:

java.lang.Object
  └java.awt.Component
    └java.awt.Container
      └java.awt.Panel
        └java.applet.Applet
          └**Processing.core.PApplet**

Interfaces implementadas:

java.awt.event.FocusListener, java.awt.event.KeyListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.image.ImageObserver, java.awt.MenuContainer, java.io.Serializable, java.lang.Runnable, java.util.EventListener, javax.accessibility.Accessible, PConstants.

Con la librería “core.jar” incluida y nuestra clase extendiendo la clase PApplet ya podremos programar nuestro sketch y ejecutarlo como un applet de java. Si quisiéramos ejecutarlo como una aplicación Java, deberíamos definir un “main” para dicha aplicación, podemos hacerlo con el siguiente código.

```
public static void main(String args[]) {
    PApplet.main(new String[] { "--present", "MyProcessingSketch" });}
```

MyProcessingSketch es el nombre de la clase, si la tuviéramos dentro de un package “package” deberíamos poner package.MyProcessingSketch.

2.4.1 Múltiples clases en eclipse

A la hora de tener múltiples clases en eclipse la cosa puede variar un poco a como las teníamos en Processing, ya que en Processing todas las clases son tratadas como “inner clases”, esto quiere decir que no son entes por ellas mismas, sino que son clases dentro de una clase PApplet. Esto nos permitía en el entorno de Processing, entre otras cosas, dibujar rectas en el PApplet o acceder a las variables de PApplet como “width” des de una clase creada por nosotros.

Podemos seguir teniendo todas las clases que vayamos creando como “inner clases” en nuestra clase principal, como sucedía en Processing, aunque a medida que nuestro proyecto vaya adquiriendo un cierto volumen y aumentando el número de clases nos irá mejor definir las clases que vayamos creando como clases independientes de nuestra clase principal. Para ello crearemos una clase nueva en eclipse, en el mismo package que nuestra clase principal. En este caso, esta nueva clase no extenderá a PApplet, ya que realmente no es un PApplet, es una clase que define algún objeto que se usará en un PApplet, por eso, deberemos indicarle en que PApplet queremos que se use definiendo un PApplet dentro de dicha clase, con el siguiente código:

```
public class Clase2 {
  PApplet parent; // Es el padre PApplet dónde crearemos el objeto de
clase Clase2.
…
}
```

Con esto podremos llamar a métodos de Processing pertenecientes a la clase PApplet usando el siguiente código, por ejemplo si queremos acceder a la propiedad width del sketch :

```
a = parent.width;
```

En el constructor de la clase deberemos añadir el sketch que será el padre :

```
Clase2(PApplet p) {
    parent = p;
…}
```

Y al llamar al constructor de la clase deberemos usar:

```
objeto = new Clase2(this);
```

Con esta información, podremos crear clases independientes en eclipse, que puedan llamar a métodos de Processing sin ser un sketch de Processing ellas mismas.

Finalmente, cabe decir que hay otro pequeño cambio al migrar a eclipse, y es que en Java no existe la primitiva “color”, de hecho, en Processing un “color” realmente es un entero (32 bits con componentes RGB y luminosidad, de 8 bits cada uno), Processing solo traduce una variable de tipo color a int, pero por lo visto en Eclipse no se hace automáticamente, por tano simplemente tendremos que cambiar las variables de tipo color por variables de tipo int.

Donde en Processing definíamos así:

```
color rosa = color(255,200,200);
```

En eclipse será de esta otra forma:

```
int rosa = color(255,200,200);
```

2.5 ProcessingBCN

ProcessingBCN fue un encuentro para personas interesadas en el mundo de Processing los días 23 y 24 de Mayo de 2009. Este encuentro tuvo lugar en Hangar, un centro de producción de artes visuales situado en el barrio Poblenou de Barcelona. En estas jornadas se trataron diferentes temas, todos ellos entorno a Processing, desde representación de movimiento en Processing hasta interacción mediante mesas multitouch y acudió un público muy heterogéneo: artistas, estudiantes, arquitectos, programadores, etc. Para este proyecto fue especialmente útil la presentación de Oriol Aragonés, de Pingpong Technologies, el cual habló sobre el proyecto que está llevando a cabo sobre creación de un framework para mesas multitouch, entre otras cosas, habló sobre diferentes librerías de captura y procesado de vídeo, lo cual me fue realmente útil para conocer más librerías de vídeo. Además, también pudimos tener una sesión práctica de cómo utilizar Processing en el entorno Eclipse.

Fig. 2.13. Jornadas de ProcessingBCN

CAPÍTULO 3. SKETCH

3.1 Objetivos

El objetivo de este sketch es el de aplicar algunas de las técnicas y métodos de los que hemos ido hablando en este documento, así como algunas librerías, de manera creativa.

La idea es crear una especie de aplicación para pintar murales virtuales, mediante un sistema de partículas, movido gracias a la detección de un objeto “lápiz”. Por otro lado también se incorpora la función de borrar.

3.2 Elementos de interacción

A lo largo de este documento hemos explorado diferentes formas de interacción con Processing mediante webcam, al final se han escogido dos maneras de interactuar, entre varias opciones posibles.

Lápiz u objeto para pintar: Este objeto será de un color que no aparecerá en la escena, el cual se pretende hacer una detección de su color, para obtener su posición. Una vez obtenidas las coordenadas x e y del objeto utilizaremos dichas coordenadas para que sean el punto de atracción de las partículas, si el objeto es detectado en la escena el sistema de partículas será atraído hasta dicha posición, dejando dibujadas en el sketch las trayectorias de cada partícula, si el objeto no es detectado el sistema de partículas permanecerá inmóvil.

Detección de movimiento: Para explorar el comportamiento de otra manera de interaccionar con el sketch de Processing, también se ha añadido una funcionalidad de detección de movimiento, en este caso usamos esta detección para borrar las líneas dibujadas por el sistema de partículas. Cuando dejen de dibujarse las partículas, es decir, cuando el sistema no detecte el “lápiz”, éste buscara en la escena posible movimiento, donde este movimiento sea detectado se borrarán las líneas dibujadas anteriormente, como si de una goma de borrar se tratase.

3.3 Elementos utilizados

3.3.1 Hardware

Para la realización de este sketch se ha utilizado el siguiente hardware y dispositivos físicos.

Cámara: Se ha utilizado una cámara Samsung VP-D453. Tiene un sensor CCD con resolución de 0,8 megapíxeles, salidas FireWare y USB. Tiene un zoom

digital de 900x y entre otras cosas proporciona un sistema de compensación de luz de fondo para eliminar la luz que provenga de detrás del objeto a filmar.

Ordenador: Para programar el sketch se ha utilizado un ordenador Toshiba Satellite L20 -101 con procesador Intel Centrino a 1.74GHz , memoria DDR2 RAM con 1GB, con tarjeta gráfica integrada 915GM Express. Sistema Operativo WindowsXP.

Iluminación: La iluminación es un factor importante para los tipos de detección que vamos a hacer. Tendremos el foco situado al lado de la cámara, y se intentará que no haya otra fuente de luz en la sala.

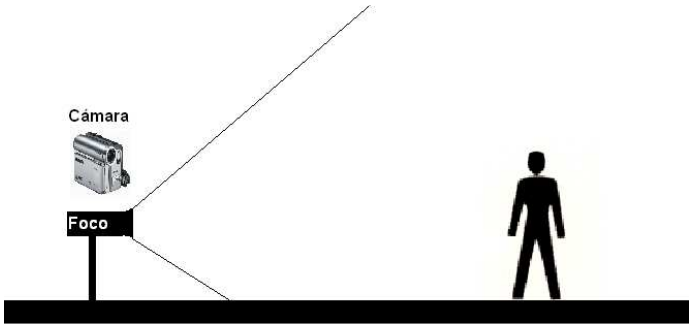

Fig. 2. Esquema de la posición de la cámara y el foco

3.3.2 Software, librerías

Para desarrollar el sketch final se usado como entorno de programación el entorno de Eclipse, ya que nos ofrece muchas más ventajas que el propio entorno de Processing. Además desde Eclipse se han importado una serie de librerías que han sido útiles a la hora de programar el sketch:

Core: Esta librería es la librería principal de Processing, en ella encontramos la clase PApplet que define todos los sketch de Processing, y que incorpora todos los métodos necesarios a la hora de programar en Processing.

GSVideo: Ya hemos hablado de esta librería anteriormente, esta librería nos aporta clases para la captura de vídeo desde la cámara, de entre las 3 librerías de captura de vídeo de las que disponíamos (Vídeo, GSVideo y JMyron), se descartó JMyron primero porque, aunque también incorpora funciones de captura, principalmente es una librería para la detección de “Blobs”, ofreciendo menos prestaciones para la captura de vídeo. De entre GSVideo y Vídeo se eligió GSVideo, ya que este no necesita QuickTime, y por tener mayor velocidad que “Vídeo” a la hora de procesar señal de vídeo. He tenido muchos problemas a la hora de utilizar GSVideo en Eclipse, ya que es una librería aún en desarrollo y según que versiones pueden no ser compatibles con el SO utilizado, además de que tuve que modificar algunos archivos de la librería porque esta no encontraba GStreamer en mi maquina.

OpenGL: OpenGL (Open Graphics Library) es una especificación estándar multiplataforma con funcionalidades para trabajar con gráficos 2D y 3D. Fue desarrollada originalmente por Silicon Graphics en 1992 y se usa entre otras cosas en el desarrollo de videojuegos, donde compite con direct3D en plataformas Windows.

La librería de Processing "OpenGL" es una adaptación de OpenGL para ser usada en Processing. Nos permite por ejemplo trabajar con pantallas más grandes, renderizar gráficos a una velocidad mayor, etc. Processing se comunica con OpenGL vía JOGL (Java OpenGL), una librería que nos permite acceder a OpenGL con Java. En Processing esta librería no incorpora ningún método adicional al que podamos llamar, simplemente tendremos que importarla y especificar que se está usando como render, en el tercer argumento del "size()",

Hay tarjetas gráficas especialmente creadas para ser usadas con OpenGL, y depende de la tarjeta que tengamos podremos sacar más o menos partido a la librería OpenGL.

GLGraphics: Esta es una librería especialmente escrita para Processing, incorpora un render OpenGL mejorado y opciones para usar efectos de tarjeta aceleradora de gráficos. Ha sido empleada porque mejora la velocidad del render de OpenGL para Processing, ya que sin ella experimentábamos un funcionamiento bastante más lento a la hora de procesar las imágenes des de la cámara y esto entorpecía la interacción. Además, en un futuro, se pretende usar una clase definida en esta librería, GLTexture, que nos permitirá un procesado aún mejor del vídeo capturado desde la webcam.

ColorMatcher : También se ha hablado ya de esta librería en este documento, en el sketch final se ha usado para detectar el color del "lápiz", de manera que se pueda saber su posición en cada frame.

3.4 Estructura y funcionamiento del Sketch

Aparte de las clases importadas, el sketch se compone de tres clases. Primero la clase principal, donde se encuentra el main de la aplicación, aquí definimos las variables y los objetos necesarios.

Una segunda clase llamada DetectaMov que es la que se encarga de hacer la detección de movimiento para borrar los trazos dibujados por el pincel. Tiene un método principal llamado update, utilizado para buscar donde se produce el movimiento y pintar allí círculos de color negro.

Por último tenemos la clase partícula. Ésta define los campos dentro de la clase partícula así como clases para mover y para cambiar el color de las

partículas. El cálculo de la velocidad de cada partícula se hace en la clase principal.

El sketch empieza primero con una etapa para elegir el color que vamos a detectar, se escoge haciendo clic sobre el color que queramos detectar en la imagen de video que aparece, esta detección de color se hará por medio de la librería ColorMatcher. Una vez elegido el color que queramos pasaremos a la siguiente etapa del sketch. En esta siguiente etapa tendremos la pantalla negra inicialmente. Podremos ir pintando en ella moviendo el objeto lápiz, el cual atraerá las partículas hacia su posición, calculando en el main su velocidad y aceleración. Si sacamos el lápiz de escena, pasaremos a detectar movimiento, podremos borrar los trazos pintados moviendo la mano o algún objeto.

Conclusiones

En este proyecto, al haber tocado varios temas, he aprendido muchas cosas.

En primer lugar, partiendo de un conocimiento muy básico de los sistemas de visión por ordenador y del procesado de imagen, he aprendido y comprendido sus distintas etapas, métodos etc. Esto me ha servido para comprender que el campo de la visión artificial es realmente muy extenso y no se ha podido hablar de todo en este proyecto, por falta de tiempo y porque no era el objetivo principal del proyecto. También he comprendido la dificultad de crear un sistema de visión robusto, que no se deba solo a una escena configurada donde tenemos controlados los colores que aparecen en ella y la iluminación.

En segundo lugar, he aprendido un nuevo lenguaje de programación, Processing, he investigado diferentes librerías escritas para Processing por la comunidad, lo cual me ha servido para comprender la potencia que puede tener una buena comunidad alrededor de un proyecto opensource. Al haber programado con Processing bajo Eclipse he aprendido a usar desde cero Eclipse, he entendido como están estructuradas las clases en Processing y con ello he aprendido conceptos entorno a estructura de las clases en Java, como qué son las Interfaces, clases heredadas, etc. También he conocido diferente software/librerías para la programación multimedia, no solo para Processing, sino para todo tipo de entornos, todas ellas opensource, como pueden ser OpenGL, GStreamer etc.

Al ser Processing un proyecto pensado para gente relacionada con el mundo del arte, he conocido todo un mundo de artistas digitales que se apoyan en proyectos como Processing para hacer sus creaciones, he podido ver muchos proyectos que se están llevando a cabo en el mundo artístico y que tienen una componente técnica muy grande, en mi opinión el artista de creaciones digitales cada vez más tendrá un perfil de programador o una base más técnica, por tanto cada vez más, este tipo de entornos enfocados a la creación arte digital serán más utilizados y más completos.

He podido también explorar y experimentar con diferentes tipos de interacción con el ordenador, es un mundo que aún está por descubrir y mejorar, que aún tiene mucho que decir sobre cómo será nuestro modo de comunicarnos con los ordenadores en un futuro.

Bibliografía

[1] Ben Fry, Casey Reas. *Processing: A Programming Handbook for Visual Designers and Artists*. MIT Press, Cambridge MA, 2007.

[2] Bernd Jähne & Horst Haubecker, *Computer Vision and aplications*, Academic Press, 2000.

[3] Daniel Shiffman, *Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction* ,Morgan Kaufman 2008.

[4] Ira Greenberg, *Processing: Creative Coding and Computational Art*, Friends of Ed., 2007

[5] Javier González Jiménez, *Visión por Computador,* Parainfo*,*2000.

[6] Web “Blob Detection” librería, http://v3ga.net/Processing/BlobDetection/

[7] Web “Blog de Bryan Chung” http://www.bryanchung.net/

[8] Web “Foro de Processing en castellano visualP5”. http://foro.visualp5.net/

[9] Web “JMyron” http://webcamxtra.sourceforge.net/

[10]Web “Learning Processing” http://www.learningProcessing.com/

[11] Web “MatchColor” http://www.local-guru.net

[12] Web “Processing” http://Processing.org/

[13] Web “TUIOClient” http://www.tuio.org/?Processing

[14] “Wikipedia” http:es.wikipedia.org/