UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Desarrollo de aplicaciones para interacción con el computador mediante el uso del Kinect


| | |
|---|---|
| **Apellidos, nombre** | Agustí i Melchor, Manuel (magusti@disca.upv.es) |
| **Departamento** | Departamento de Informática de Sistemas y Computadores (DISCA) |
| **Centro** | Escola Tècnica Superior d'Enginyeria Informàtica<br>Universitat Politècnica de València |

# 1 Resumen de las ideas clave

Este artículo constituye una introducción **al desarrollo de aplicaciones que hagan uso de dispositivos de interacción** complejos denominados **sensores 3D** (o también D*epth Cameras, Ranging Cameras* o *RGB-D Cameras*). Estas "cámaras" proporcionan[1], fig. 1, dos fuentes de información visual sincronizadas: una imagen con la información de color (*Color RGB image*) sincronizada con otra imagen que contiene el valor de distancia al sensor (*Depth image*) para cada píxel. También es habitual que permitan capturar, con uno o varios micrófonos, la información de audio (*Audio stream*) de alrededor. En este artículo veremos cómo se pueden desarrollar aplicaciones y probarlas sin tenerlo físicamente delante.

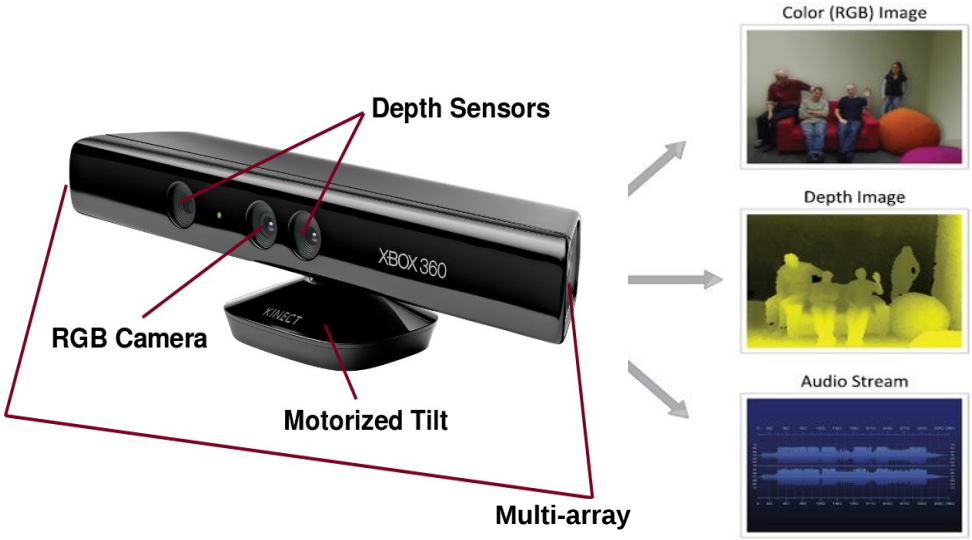



*Figura 1: Tipos de información de entrada habituales en un sensor 3D.*

# 2 Objetivos

En el presente documento, veremos cómo se puede realizar una aplicación que utilice uno de estos sensores 3D. Para ello se expondrá cómo estructurar una aplicación desarrollada en *C* y bajo *GNU/Linux* con la librería *libfreenect/OpenKinect* debido a la posibilidad de trabajar sin tener físicamente el dispositivo conectado.

El presente documento está encaminado a ofrecer una perspectiva inicial del modo de funcionamiento del *kinect* con OpenCV como medio de integración, en una aplicación, de la información de carácter visual y de distancias. Los **objetivos** de este documento son:

[1] Composición de imágenes extraídas de [2] y de <http://www.hizook.com/files/users/3/PrimeSense_DepthCamera.jpg>.

- Dar a conocer al alumno, de un modo participativo, las diferentes fuentes de información visual y sonora de este dispositivo, así como también la posibilidad de modificar su orientación vertical y el *led* de actividad.
- Dar a conocer al alumno, de un modo participativo, las operaciones básicas de captura de imagen RGB y de profundidad que se puede obtener de este dispositivo mediante el interfaz de *OpenKinect* y cómo se integran en una aplicación realizada con *OpenCV*.
- Describir una plataforma de experimentación abierta y multiplataforma que permita al alumno proseguir su autoaprendizaje de forma independiente, con o sin el dispositivos físico conectado al computador.

A partir de la exploración de ejemplos de uso, se utilizarán operaciones básicas de manipulación de la estructura de datos que soporta el concepto de imagen en mapa de bits, operaciones habituales que permiten la manipulación de ficheros de formatos gráficos, así como el uso de cámaras para la obtención de imágenes. Por ello es recomendable tener a mano la documentación correspondiente a la versión del API de OpenCV **que no es uno de nuestros objetivos** (en línea[2] o en local).

# 3 Introducción

Nos referimos en este documento, al hablar de sensores 3D, a dispositivos que estan muy presentes en entornos lúdicos donde la interacción es alta y utiliza diferentes fuentes de información, lo que ha dado lugar a la aparición de nuevos dispositivos y aplicaciones tanto dentro como fuera del contexto de los videojuegos.

Aparte del desarrollo inicial de *Prime Sense* para Microsoft Xbox (Kinect [1]), otros ejemplos de estas cámaras de tipo RGB-D han ido apareciendo, la *fig. 2* muestra[3] algunos*:* el *Xtion* de *Asus*, también creado en colaboración con *Prime Sense*; el *RealSense 3D* de Intel y Creative, el *Structure Sensor*[4] de *Occipital (para* tabletas)[5] y son una de las piezas claves de las que se están dando en llamar *Smartglasses* como[6] las *SpaceGlasses* de *Meta*, *Google Glass*, *Atheer*, o las *HoloLens* de *Microsoft*.

## 3.1 Características del *Kinect*

En las consolas de videojuegos hemos visto aparecer dispositivos con los que el usuario interactúa con las aplicaciones de estos equipos, sustituyendo al uso de los tradicionales teclado y ratón, haciendo más natural la interacción entre el hombre y la máquina. Todos ellos determinan la posición del usuario y la envían al computador junto con otras informaciones como la orientación y aceleración del mismo.

---

[2] En el sitio web de Documentation | OpenCV <http://opencv.org/documentation.html>

[3] Imágenes tomadas respectivamente de <https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE/>, <https://software.intel.com/en-us/blogs/2015/01/26/can-your-webcam-do-this> y <http://structure.io/>.

[4] Véase la página web del fabricante en <http://structure.io/>.

[5] Véase la página web del fabricante en <http://structure.io/>.

[6] En el orden en que son citadas, las páginas web de los fabricantes son: <https://www.getameta.com>, <https://developers.google.com/glass/>, <http://atheerair.com/> y <https://www.microsoft.com/microsoft-hololens/en-us/>.

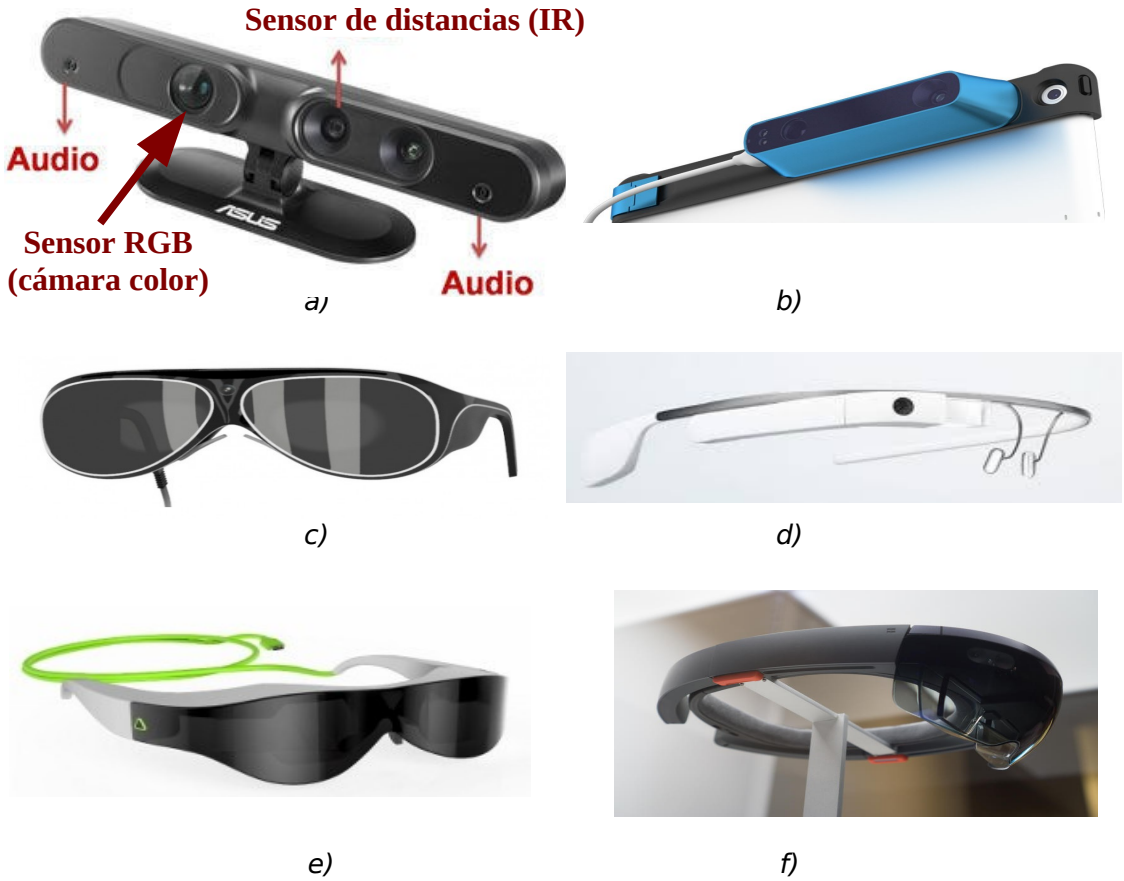



*Figura 2: Sensores 3D de a) Xtion y b) Structure Sensor. Smartglasses de c) SpaceGlasses, d) Google Glasses, e) Atheer Glases y f) HoloLens.*

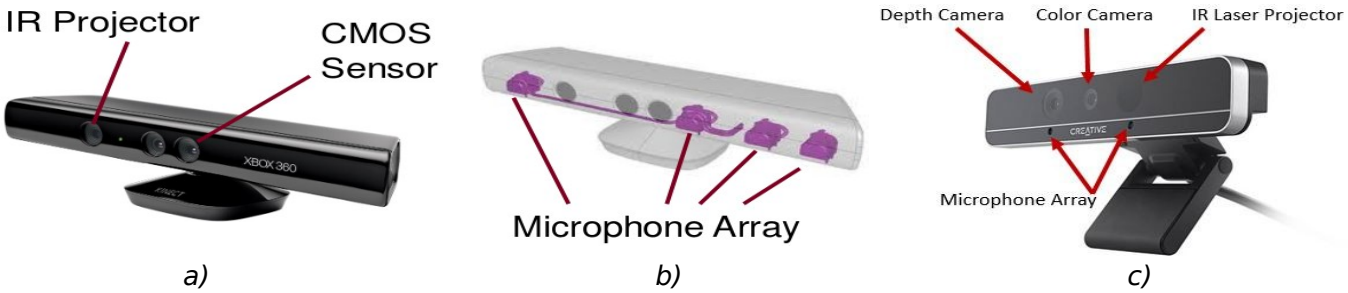
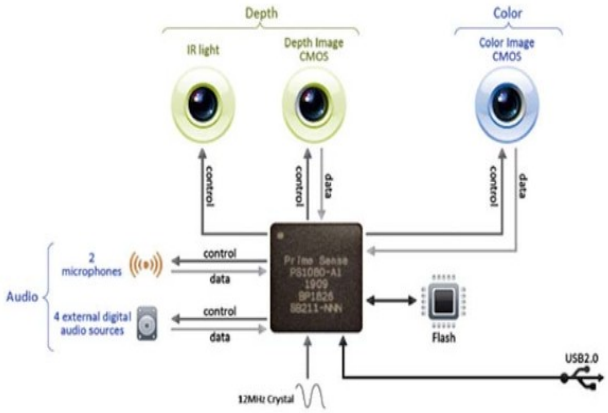



*Figura 3: Elementos del Kinect: a) Detalle de la posición de los componentes del "sensor de distancias", b) de los micrófonos y c) variaciones en el RealSense 3D.*

Para abordar cómo se puede realizar una aplicación que utilice uno de estos dispositivos, en concreto el *kinect* ([*1*] y [2]), conectado a nuestro computador, veamos primero, fig, 3, los componentes[7] (cámaras y micrófonos) del mismo. Todos

[7] Imágenes extraídas de [2] y de <https://software.intel.com/en-us/blogs/2015/01/26/can-your-webcam-do-this>.

ellos interconectados por un procesador [3] que es el encargado de enviar a través de la conexión USB la señal decodificada de los componentes [3] del *kinect*, como muestra la fig. 4a. Las características [4] de este procesador (*Carmine* (PS1080), que después daría paso al *Capri* (PS1200)) se listan en la fig. 4b.




| PROPERTY | | CARMINE 1.08 | CARMINE 1.09 | UNIT |
|---|---|---|---|---|
| Operating Temperature | | 5 – 40 | 10 – 40 | [°C] |
| Data Interface | | USB 2.0 / 3.0 | USB 2.0 / 3.0 | |
| Operation Range | | 0.8 – 3.5 | 0.35 – 1.4 | [m] |
| Field of View (Horizontal, Vertical, Diagonal) | | 57.5, 45, 69 (H,V,D) | 57.5, 45, 69 (H,V,D) | [deg] |
| Depth Image Size | | 640 x 480 (VGA) | 640 x 480 (VGA) | [pixel x pixel] |
| Spatial x/y Resolution (2-Sigma Values) | @2m | 3.4 | | [mm] |
| | @0.5m | | 0.9 | [mm] |
| Depth Resolution (2-Sigma Values) | @2m | 1.2 | | [cm] |
| | @0.5m | | 0.1 | [cm] |
| Maximal Frames-per-Second Rate | | 60 | 60 | |
| Color Image CMOS | @ 30 FPS | 640 x 480 (VGA) | 640 x 480 (VGA) | [pixel x pixel] |
| Built-in Microphones | | 2 | 2 | |
| Data Format | | 16 | 16 | [bit] |
| External Digital Audio Inputs | | 4 | 4 | [inputs] |
| Dimensions: Width x Height x Depth | | 18 x 2.5 x 3.5 | 18 x 2.5 x 3.5 | [cm] |
| Power Supply | | USB | USB | |
| Maximal Power Consumption | | 2.25 | 2.25 | [Watt] |

*a)* *b)*

*Figura 4: Procesador interno del kinect: a) Interconexión de los componentes y b) características.*

## 3.2 Conexión con el computador

Para conectar el dispositivo con el computador es necesario disponer de la conexión física y lógica. **La conexión física** es la de los cables y conectores que permiten conectar el ordenador con el *kinect*, lo cual es más o menos directo, ya que existen diferentes modelos de este dispositivo. En los primeros modelos que aparecieron en el mercado (1414 y 1473, que se vendían junto a la consola), necesitan un adaptador para poder conectar el dispositivo a un puerto USB y, directamente, a la alimentación. Existen, al menos[8], otros dos modelos: el 1517 y el *Kinect* v 2.0 (Nov. 2013).

**La conexión lógica** es la que implementa el manejador (*driver*) y que consiste en el interfaz de bajo nivel, que permite (si el núcleo del operativo al que se le conecta no reconoce el dispositivo) acceder a los datos en bruto que suministra el dispositivo (imagen o audio), así como a los registros que permiten su configuración (color y parpadeo del led y movimiento del motor). Sobre este una biblioteca de funciones de alto nivel ofrece un conjunto de operaciones de mayor complejidad como detección de gestos, del esqueleto, de caras, etc.
Para implementar este interfaz existen dos grandes alternativas de carácter multiplataforma: *OpenKinect [5] / libfreenect* y *OpenNI2*[9] */ OpenNI (NITE + Prime Sense)*. Nos vamos a deterner en la primera opción por la particularidad de ofrecer un "emulador" del dispositivo, lo que nos permitirá implementar y probar ciertas funcionalidades en código como si tuviéramos realmente conectado un "kinect real".

[8] Vése "Kinect Hacking" disponible en <http://idav.ucdavis.edu/~okreylos/ResDev/Kinect/MainPage.html>.

[9] Tras la compra de Prime Sense por parte de Apple y el cierre del sitio original de OpenNI (http://www.openni.org/), otros fabricantes de aplicaciones y dispositivos basados en las mismas ideas mantienen la nueva versión denominada OpenNI2 en <http://structure.io/openni>.

# 4 Desarrollo: explorando el uso del kinect

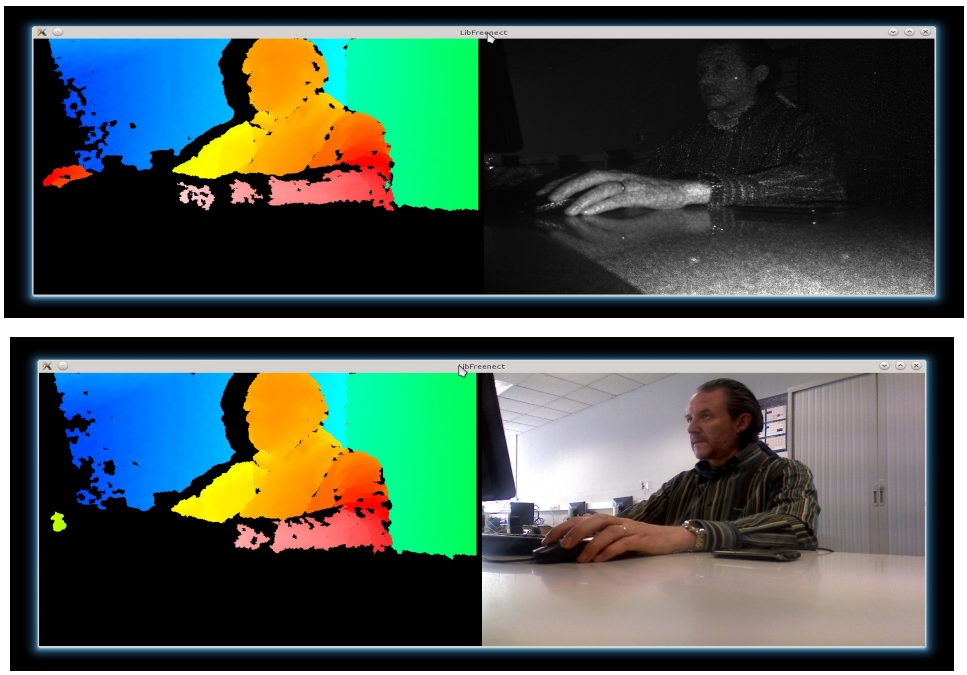
*OpenKinect* contiene utilidades que permiten el acceso a través del manejador (*driver*) *libfreenect* a: las cámaras (de color y de profundidad), el motor, los acelerómetros, el *led* de funcionamiento y el los micrófonos. **Además permite el acceso a un dispositivo simulado** (*Fakenect*), lo que permite ejecutar el código sin tener el *kinect* conectado.

## 4.1 Utilidades de acceso: OpenKinect/libfreenect

En la instalación de *libfreenect* vienen tres aplicaciones que vamos a describir. Otros ejemplos están disponible en el GITHub del proyecto[10], como son *camtest.c*, *tiltdemo.c* (para el acceso al motor) o *micview.c* y *wavrecord.c* (para acceder al registro de audio a través del vector de micrófonos disponible). Si necesita compilar algún ejemplo con *libfreenect*, los parámetros para el compilador son:

*$ pkg-config libfreenect --cflags --libs*

*-I/usr/include/libusb-1.0 -lfreenect -lusb-1.0*



*a)*

*b)*

*Figura 5: Salida gráfica de glview en modo a) hires_color y b) IR.*

La más básica es *freenect-glview*, la fig. 5 muestra su salida en pantalla. Pulsando la tecla 'f' se va cambiando el modo de vídeo: de RGB (fig. 5a) a la imagen de la cámara de profundidad (infrarrojos, fig.5b). Es el punto de partida para una aplicación típica sobre este dispositivo. En la fig. 5, las dos imágenes de la izquierda, muestran la imagen de profundidad; coloreada en base a intervalos de los valores ofrece la cámara de profundidad. Los valores en negro indican que el procesador interno no ha sido capaz de determinar la distancia para esos puntos,

[10] En la URL <https://github.com/OpenKinect/libfreenect/tree/master/examples>.

por caer fuera del rango de trabajo (tanto por demasiado cerca como por demasiado lejos).

Por otro lado, *freenect-cppview*[11], permite el acceso al led de actividad del frontal del dispositivo y al motor. Lo que ofrece la posibilidad de cambiar el color del led, pulsando '1', '2', '3'. '4' ('5' da el mismo resultado), '6' y '0'. Lo que hace variar el color, respectivamente, entre verde, amarillo, verde parpadeando, rojo y amarillo alternativamente y apagado. También se puede activar el motor para cambiar la posición vertical del dispositivo (dentro de un ángulo de ±30º), incrementado con 'w', decrementando con 's' (o 'd'), e inicializándolo a cero con 'x', a 10º con 'e' y a -10º con 'c'.

Para acabar, *freenect-glpclview*[12],realiza un alzamiento 3D de la escena, a partir de la imagen de distancias y con la textura de la imagen RGB sobrepuesta. Aquí se pueden utilizar las teclas 'w' y 's' para aumentar o decrementar el zoom de la imagen y, también, la tecla 'c' para activar la visualización, o no, de la textura.

## 4.2 Fakenect: el simulador de kinect

Para experimentar sin tener un *kinect* real conectado, se utilizará un conjunto de ficheros grabados con toda la información que se puede obtener de un *kinect* real en ejecución. Esto se realiza en base a dos utilidades (*fakenect* y *record*). Por un lado, **la biblioteca dinámica de funciones *fakenect***, que permite, sin modificar el código a probar, ejecutarlo sobre un "dispositivo simulado" que va tomando como entrada los datos grabados previamente. En la distribución original de *OpenKinect* hay tres conjuntos de datos[13] (*legos*, *sophie*, y *thanksgiving*). Vienen en formato comprimido, por lo que habrá que descomprimirlos previamente. Así, por ejemplo, para trabajar con "thanksgiving0.tar.bz2"ejecutaremos:

*$ cd /tmp*

*$ tar xvf thanksgiving0.tar.bz2*

Con lo que obtendremos un directorio "thanksgiving0" que ya podremos utilizar para simular un *kinect*, Veamos ahora cómo se utiliza el "falso" *kinect* con las mismas aplicaciones que acabamos de describir, lo que es aplicable a cualquier otra que se desarrolle. Es posible ejecutar cualquiera de las aplicaciones desarrolladas con una sola orden (recuerde que ha de escribirlo todo en una sola línea) del estilo:

*$ LD_PRELOAD="/usr/lib/fakenect/libfreenect.so"*
*FAKENECT_PATH="thanksgiving0" /usr/bin/freenect-glview*

Observe que se ha indicado que se cargue (LD_PRELOAD), previamente a la ejecución de la aplicación. una biblioteca dinámica (DLL) con el mismo nombre de la que se utiliza con el dispositivo real, pero que implementa el mecanismo de acceso a los ficheros del directorio indicado (FAKENECT_PATH) y, finalmente, la ruta (en este caso de forma absoluta) de la aplicación que se quiere probar.

---

[11] En la URL <https://github.com/OpenKinect/libfreenect/blob/master/wrappers/cpp/cppview.cpp>.

[12] En la URL <https://github.com/OpenKinect/libfreenect/blob/master/examples/glpclview.c>.

[13] Originalmente disponibles en <http://dl.dropbox.com/u/15736519/legos0.tar.bz2>, <http://dl.dropbox.com/u/15736519/sophie0.tar.bz2> y <http://dl.dropbox.com/u/15736519/thanksgiving0.tar.bz2>.

Por el otro lado, hay que generar esa "información pregrabada" y para ello se utiliza la aplicación *record*. Esta utilidad no viene con la instalación por defecto[14]. Para compilar la bastará ejecutar la orden:

*$ gcc record.c -o record -I/usr/include/libusb-1.0 -lfreenect -lusb-1.0*

Se puede ejecutar indicando el nombre de un directorio donde se dejara el resultado de la captura con la orden:

*$ record prova_manolo*

hasta que lo pare con *Ctrl+C*. En pantalla, no podrá ver las imágenes que se están grabando. El contenido del directorio es un conjunto de archivos, con la marca temporal de cuándo se han obtenido para poder ser reproducidos después en orden. Se nombran con extensión ".dump" para los valores de los acelerómetros, ".pgm" para la imagen de grises de la cámara de distancias y ".ppm" para la imagen de la cámara de vídeo RGB. Para visualizar lo que se ha grabado puede utilizar cualquiera de las utilidades ya vistas, p. ej. *glview*, indicando la ruta hasta el directorio donde se haya guardado la grabación:

*$ LD_PRELOAD="/usr/lib/fakenect/libfreenect.so"*
*FAKENECT_PATH="prova_manolo/" /usr/bin/freenect-glview*

# 5 Programas de ejemplo

Nos interesa profundizar en la relación entre este dispositivo y cómo se puede incorporar en una aplicación. Veamos cómo OpenCV puede acceder al dispositivo y cómo puede trasladar la información de vídeo a sus estructuras de datos. Para poder aplicar a partir de ahí cualquier algoritmo desarrollado con OpenCV.

El manejador de *libfreenect* proporciona dos métodos de acceso a las cámaras. Uno asíncrono, que permite asociar funciones (*callbacks*) que serán llamadas cuando haya datos disponibles. Y otro síncrono[15], que ofrece dos funciones ("freenect_sync_get_depth" y "freenect_sync_get_video"). Estas funciones obtendrán la imagen de la cámara correspondiente y esperarán a que se complete la transferencia de los datos al espacio de memoria de la aplicación de usuario antes de devolver el control.

Desde el sitio web de *OpenKinect [5]* donde se describe este interfaz, se proporciona un ejemplo de código que se compila con la orden:

*$ gcc cvdemo.c -o cvdemo `pkg-config opencv libfreenect --cflags --libs` -lfreenect_sync*

Hay que hacer notar que, a diferencia de una típica aplicación con OpenCV, se necesita la cabecera para la nueva librería *libfreenect_sync*, así como las funciones *freenect_sync_get_video* y *freenect_sync_stop*

Es importante hacer notar que, del dispositivo, se obtiene una secuencia de bits que hay que saber interpretar y asignar a una estructura de datos de OpenCV para que esta la pueda manipular, en este caso simplemente mostrarla en pantalla. Por ello se ha inicializado una variable *image* (con *cvCreateImageHeader*) y se le asignarán posteriormente los valores correspondientes a la información de color de los píxeles

[14] Habría que bajarla del sitio Web de OpenKinect. En concreto *record* está en <https://github.com/OpenKinect/libfreenect/tree/master/fakenect>.

[15] Esta recibe el nombre de *C Sync Wrapper* <https://openkinect.org/wiki/C_Sync_Wrapper>.

(con *cvSetData*). De las características del dispositivo se puede obtener la información de frecuencia de muestreo y tamaño de muestra para el flujo de información de vídeo en color (RGB). Este ejemplo se puede extender para obtener la información de la cámara de distancias y visualizarla.

# 6 Desarrollo: explorando el uso del kinect

Dado el formato de la imagen de profundidad, aunque corresponde en resolución espacial con la de RGB, se corresponde en resolución de "color" con cantidades de 16 bits y valores válidos en un rango de 11 bits[16], en las versiones anteriores al *kinect v2.0* al menos. Esto nos da valores en el rango [0 .. 2047] (o lo que es lo mismo, entre el 0x0000 y el 0x07FF), siendo el valor máximo ($2^{11}$-1) el de distancia no valida.

Con intención de mostrar también esta información el listado 1 ha inicializado y creado nuevas variables de tipo *IplImage* (con *cvCreateImageHeader* y *cvCreateImage*). Ha obtenido la información de la cámara de infrarrojos (con *freenect_sync_get_depth*) y la ha asignado a una variable de tipo *IplImage* con la información (con *cvSetData*) de distancias. Ha convertido la imagen con los valores de profundidad a una que permita su visualización, con una conversión (con *cvConvertScaleAbs*) lineal: 0.. $2^{11}$-1 → 0 .. $2^{8}$-1. Es muy simple para obtener altos resultados de precisión, pero lo es lo suficiente como para indicar la necesidad de este proceso y obtener una versión representable en pantalla.


*Figura 6: Salida las cámaras obtenida en exempleBasic__OpenCV_Kinect: a) RGB y b) Distancias.*

[16] Esta información se ha extraído de [2] y de G. Borenstein. (2012). "Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot". O'Reilly Media, Inc. ISBN1449327788.

```c
#include <stdlib.h>
#include <stdio.h>
#include <opencv/cv.h>
#include <opencv/highgui.h>
#include <libfreenect_sync.h>

int main()
{
    int x, y; CvScalar color;

    IplImage *image, *imageIR, *imageIR8b;
    // imageIR es de 16 bits vs  imageIR8b que es de 8 bits
    image = cvCreateImageHeader(cvSize(640,480), 8, 3);
    imageIR = cvCreateImageHeader(cvSize(640,480), IPL_DEPTH_16U, 1);
    imageIR8b=cvCreateImage(cvSize(640,480), IPL_DEPTH_8U, 1);

    cvNamedWindow("RGB", CV_WINDOW_AUTOSIZE);
    cvNamedWindow("IR", CV_WINDOW_AUTOSIZE);

    while (cvWaitKey(10) < 0)
    {
      char *data, *dataIR; unsigned int timestamp;
      double min_val, max_val;

      freenect_sync_get_video((void**)(&data), &timestamp, 0,
              FREENECT_VIDEO_RGB);
      cvSetData(image, data, 640*3);
      cvCvtColor(image, image, CV_RGB2BGR);
      cvShowImage("RGB", image);

      freenect_sync_get_depth((void**)(&dataIR), &timestamp, 0,
                    FREENECT_DEPTH_11BIT);

      cvSetData(imageIR, dataIR, 640*2);
      cvMinMaxLoc(imageIR, &min_val, &max_val, NULL, NULL, NULL );
      printf("ImageIR: min %f, max %f; ", min_val, max_val);

      cvConvertScaleAbs(imageIR, imageIR8b, 255.0/2047,0);
      cvMinMaxLoc(imageIR8b, &min_val, &max_val, NULL, NULL, NULL );
      printf("ImageIR8b: min %f, max %f\n", min_val, max_val);
      cvShowImage("IR", imageIR8b);
    }
    freenect_sync_stop();
    cvFree(&image);
    return EXIT_SUCCESS;
}
```

*Listado 1: Ejemplo ampliado del interfaz síncrono para C: obteniendo la imagen de IR.*

El ejemplo que muestra el el listado 1, al ejecutarse va mostrando las imágenes en pantalla (fig. 6) y esos rangos de valores diferentes (lo que devuelve la cámara y la que hemos generado para visualizarla) en el terminal. Se muestran solo los valores máximo y mínimo de cada imagen en la terminal :

*$ gcc exempleBasic__OpenCV_Kinec.c -o exempleBasic__OpenCV_Kinec `pkg-config opencv libfreenect --cflags --libs` -lfreenect_sync*

*$ exempleBasic__OpenCV_Kinec*

*ImageIR: min 417.00, max 2047.00; ImageIR8b: min 52.00, max 255.00*

...

# 7 Conclusión

En este documento, el lector habrá utilizado básicamente el uso de la información de carácter visual que proporciona un sensor 3D como el *kinect* porqué es lo que compete al API de OpenCV. Pero también hemos visto que con el interfaz de acceso que ofrece OpenKinect es posible acceder al motor y los micrófonos que dispone este dispositivo.

Dejo de la mano del lector interesado y que disponga del dispositivo, explorar otros caminos[17] para utilizar el *kinect* como dispositivo de obtención de información de carácter visual dentro de OpenCV. Profundizar en la detección y el uso de gestos debería ser el siguiente paso. Combinar las opciones de OpenKinect y OpenNI[18] es otra posibilidad.

# 8 Bibliografía

[1] Kinect for Windows Sensor Components and Specifications. Disponible en <https://msdn.microsoft.com/en-us/library/jj131033.aspx>.

[2] Bloisi, D. d. y Pennisi, A. (2015). Robot Programming. Section of Elective in Artificial Intelligence. Master Artificial Intelligence and Robotics. Department of Computer, Control and Management Enginneering Antonio Ruberti. Sapienza Università Di Roma. Disponible en <http://www.dis.uniroma1.it/~nardi/Didattica/CAI/matdid/image_processing_with_opencv.pdf>.

[3] Billie, G. (2011). Microsoft Kinect Sensor Evaluation NASA USRP. Internship Final Report. Johnson Space Center. 8/5/2011. Disponible en <http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20110022972.pdf>.

[4] PrimeSense™ 3D Sensors. Disponible en <www.i3du.gr/pdf/primesense.pdf>.

[5] OpenKinect. Disponible en: <https://openkinect.org/wiki/Main_Page>.

[6] Documentación de OpenCV. Using Kinect and other OpenNI compatible depth sensors. <http://docs.opencv.org/3.1.0/d7/d6f/tutorial_kinect_openni.html#gsc.tab=0>.

[17] Por ejemplo, “how to use Kinect with openni and opencv” <http://stackoverflow.com/questions/22531480/how-to-use-kinect-with-openni-and-opencv>.

[18] Véase el enfoque combinado en “Setting up Kinect for programming in Linux (part 1)” <https://www.kdab.com/setting-up-kinect-for-programming-in-linux-part-1/> y “Setting up Kinect for programming in Linux (part 2)” <https://www.kdab.com/setting-up-kinect-for-programming-in-linux-part-2/>.