Integración de Medios Digitales:

Utilizando MS/KINECT

Memoria

Gonzalo Ortega Castillo (gonorcas@inf.upv.es)

Francisco Vázquez Palacios (fravzpa7@etsid.upv.es)

Grado en Ingeniería Informática

Abril, 2016

Índice

Índice de figuras

Presentación del proyecto

El proyecto que planteamos está enfocado a la manipulación de presentaciones en formato PDF empleando para ello la tecnología que nos ofrece Kinect. La idea principal es interactuar con las diapositivas utilizando un lenguaje gestual, sencillo y natural. Ofreciendo algunos iconos en la pantalla de manera que el ponente pueda hacer un gesto para hacerlos aparecer y seleccionar la herramienta oportuna. Entre las herramientas que hemos pensado incorporar se incluye una simulación de puntero láser para señalar, un pincel para dibujar con distintos colores y una goma para borrar lo dibujado.

Para ello, como se comentará más adelante, utilizaremos librerías de código abierto para su utilización en Linux (OpenNI + NITE) con el lenguaje Python y la primera versión del dispositivo. Por contra, también realizaremos un desarrollo paralelo en Windows utilizando el SDK que Microsoft proporciona para el desarrollo de aplicaciones de Kinect, junto con la versión dos del dispositivo, para intentar sacar el máximo potencial y poder realizar una comparativa completa.

Objetivos

El objetivo de nuestro proyecto es explorar la funcionalidad que el sensor Kinect nos ofrece, para poder realizar comparativas, utilizando diferentes sistemas operativos, lenguajes de programación, librerías (middleware) y hardware.

Microsoft Kinect

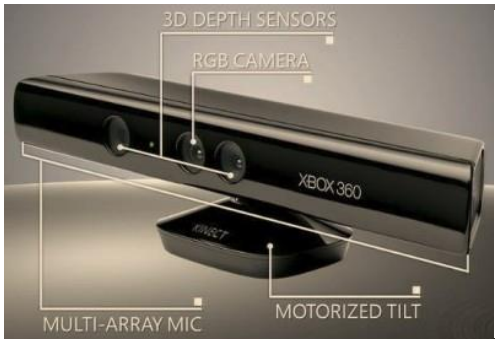
El proyecto que a continuación se va a definir gira en torno del dispositivo Microsoft Kinect. Dispositivo que facilita enormemente la obtención de información del medio para una interacción natural con la máquina. Durante el desarrollo de esta aplicación hemos conseguido el acceso tanto a un Kinect v1 (Xbox 360) como a un Kinect v2 (Xbox One). El primero cuenta con dos cámaras, una de la que obtenemos la imagen en RGB (640x480p) y otra de la que obtenemos información de profundidad. También cuenta con 4 micrófonos y una base motorizada que permite el seguimiento de formas. Por otro lado el Kinect v2 ofrece algunas mejoras técnicas:

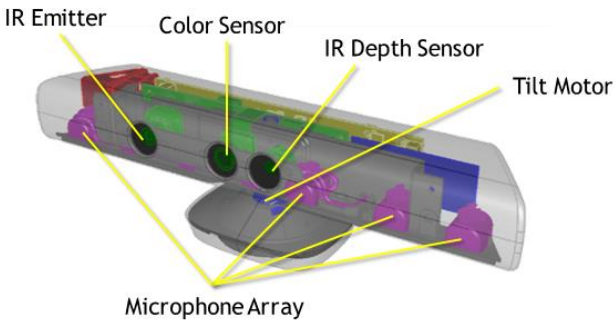
- Mayor campo de visión. 70º en horizontal (antes 57º) y 60 en vertical (antes 43º).
- Mayor resolución. 1920 x 1080 Full HD (antes 640 x 480).

- Mejora el rango de profundidad del sensor. El rango de actuación pasa a ser de 0,5 a 4,5 metros.
- USB 3.0. Mejora de la captación de sonidos.
- Captación de movimiento a oscuras.
- Kinect 2 permite calcular/analizar la fuerza de nuestros músculos y medir el ritmo cardíaco.

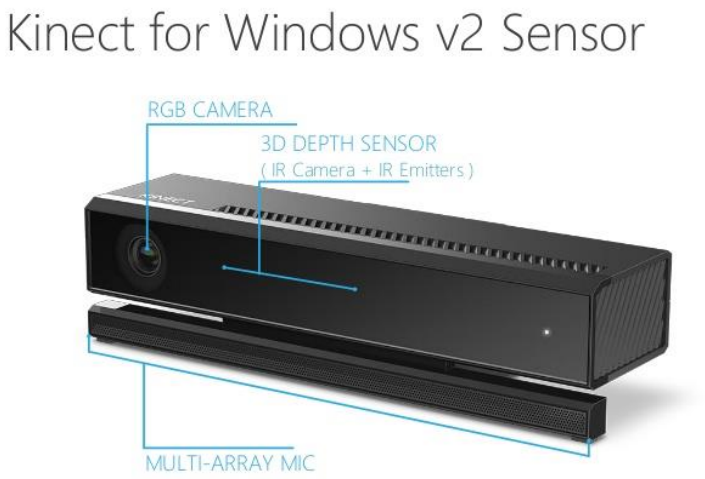
Ninguno de los aspectos que mejora una versión frente a otra nos son significativamente útiles, pero cabe reconocer que en el ámbito de los videojuegos para el que fue diseñado presenta enormes ventajas la segunda versión.

Kinect v1 (XBOX 360):





Kinect v2 (XBOX ONE):



Instalación en GNU/Linux [2]

La instalación del dispositivo de Microsoft, sin emplear el driver oficial es muy complicada debido a la falta de documentación y la existente está bastante obsoleta. A esto, cabe añadir otro problema: la compra de PrimeSense por Apple. PrimeSense se encargaba de desarrollar las librerías OpenNI y NITE para Kinect, por lo que todos los links de descarga de todo lo desarrollado por PrimeSense redirigen a www.apple.com y no se puede acceder a su contenido. Pese a todo, después de una larga tarea de investigación, hemos conseguido reunir una pequeña guía con los pasos para realizar la instalación. NITE es un middleware de detección de movimiento, que utiliza el SDK de OpenNI para su funcionamiento. Además, los desarrolladores de OpenNI han continuado con el desarrollo de las librerías, y actualmente es posible descargar OpenNI 2, que tiene algunas mejoras con respecto a su predecesor.

Hemos optado por realizar las implementaciones en Python por la brevedad del código y de las tareas de compilación. Por lo que la instalación siguiente es para utilizar Kinect con Python en Linux.

La siguiente instalación se ha realizado en Ubuntu 14.10 a fecha de 10 de Marzo de 2016:

- Creamos una directorio en el directorio 'home'

```
mkdir ~/kinect
cd ~/Kinect
```

- Obtenemos las dependencias que posteriormente necesitaremos

```
sudo apt-get install git-core cmake python-dev python-numpy freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev libgtk2.0-dev libavcodec-dev libavformat-dev libswscale-dev doxygen mono-complete graphviz
```

- Creamos un directorio para las librerías y bajamos los repositorios de git

```
git clone https://github.com/OpenKinect/libfreenect
git clone https://github.com/OpenNI/OpenNI
git clone https://github.com/avin2/SensorKinect
git clone https://github.com/jmendeth/PyOpenNI.git
sudo apt-get update
```

- ~~Instalamos libfreenect~~ [1]

```
cd ~/kinect/lib/libfreenect
mkdir build
cd build
cmake ..
make
sudo make install
```

```
sudo ldconfig /usr/local/lib64/
sudo chmod a+rw /dev/bus/usb//
```

- Instalamos OpenNI-1.5.4.0 [4]

```
cd ~/kinect/lib/OpenNI
git checkout Unstable-1.5.4.0
cd Platform/Linux/CreateRedist/
sudo ./RedistMaker
cd ../Redist/OpenNI-Bin-Dev-Linux-x64-v1.5.4.0
sudo ./install.sh
```

- Instalamos SensorKinect

```
cd ~/kinect/lib/SensorKinect/Platform/Linux/CreateRedist
./RedistMaker
cd ../Redist/Sensor-Bin-Linux-x64-v5.1.2.1/
sudo ./install.sh
```

- Instalamos NITE 1.5.2.21

```
cd ~/kinect/lib/NITE-Bin-Dev-Linux-x64-v1.5.2.21
chmod a+x install.sh
sudo ./install.sh
```

- Instalamos PyOpenNI

```
cd ~/kinect/lib/PyOpenNI/
mkdir build
cd build
cmake ..
make
```

- Ahora copiamos la librería para que Python la encuentre sin problemas y añadimos la siguiente licencia de PrimeSense:

```
<License vendor="PrimeSense" key="0KOIk2JeIBYClPWVnMoRKn5cdY4="/>

sudo cp ./lib/openni.so /usr/lib/pymodules/python2.7/
nano ~/kinect/lib/OpenNI_1.5.4.0/Data/SamplesConfig.xml
```

- Damos privilegios a la carpeta Data

```
cd /usr/etc/primesense/Features_1_5_2/
sudo chown -R <username>:<username> Data/
```

- Reinicia y prueba los ejemplos de OpenNI

```
cd ~/kinect/lib/OpenNI/Platform/Linux/Bin/x64-Release/
./Sample-NiSimpleViewer
./Sample-NiSimpleSkeleton
```

- También es necesario instalar OpenGL, se puede hacer de forma automática mediante el siguiente comando (si no funcionara, siempre se puede instalar manualmente como se explica en la siguiente página http://pyopengl.sourceforge.net/documentation/installation.html):

```
pip install PyOpenGL PyOpenGL_accelerate
```

- Por último necesitaremos instalar OpenCV 2.4.X, lo haremos utilizando git, ya que así el proceso de instalación puede extrapolarse a usuarios que utilicen otras distribuciones de Linux:

```
sudo apt-get install build-essential
sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev libavformat-dev libswscale-dev
sudo apt-get install python-dev python-numpy libtbb2 libtbb-dev libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
cd ~/<my_working _directory>
git clone https://github.com/Itseez/opencv.git
cd ~/opencv
mkdir release
cd release
cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local ..
make
sudo make install
```

Con estos pasos deberíamos tener las librerías OpenNI y OpenCV con los complementos necesarios para que nuestra máquina Ubuntu detecte nuestro Kinect y poder comunicarnos con él a través de Python.

En los pasos de instalación, se explica cómo instalar libfreenect por si en un futuro se desea realizar pruebas con FakeKinect, pero en nuestro caso no es necesario ya que no lo utilizamos.

Instalación en Windows

La instalación se realizó sobre un equipo Windows 10. Para la instalación en el sistema operativo nativo del hardware el procedimiento es muy sencillo. En nuestro caso las pruebas en el entorno de Windows se han realizado utilizando el Kinect v2, los requisitos de instalación son, obviamente tener el sensor, disponer de (al menos) un puerto USB 3.0, un adaptador para conectar el sensor a la corriente y al PC, y finalmente descargar el SDK correspondiente de la página de Microsoft:

https://www.microsoft.com/en-us/download/details.aspx?id=44561

El motivo por el que hemos decidido utilizar el SDK de Windows es que al intentar instalar las librerías libfreenect, OpenNI, y en concreto la librería PyOpenNI, nos ha resultado imposible hacerlas funcionar ya que Apple compró la empresa que desarrollaba dichos proyectos, y estos han quedado desactualizados, siendo posible descargarlos desde alguna página externa, pero quedando completamente inutilizables en éste sistema operativo.

Como este kit de desarrollo (Windows SDK Kinect v2) nos proporciona una funcionalidad total del dispositivo, no necesitamos utilizar OpenNI al trabajar en Windows. Además el kit nos permite la programación en C++ y C# de aplicaciones que utilicen nuestro sensor aprovechando al máximo su funcionalidad. El SDK incluye una serie de programas de prueba, para mostrar con diversos ejemplos sencillos todas las funciones que se proporcionan. En nuestro caso nos hemos centrado en el ejemplo denominado DiscreteGestureBasics, que nos muestra cómo capturar y tratar los gestos realizados con la mano, utilizando un parámetro interno denominado HandState.

Implementación del proyecto

Lo primero en la implementación de nuestro proyecto, fue mostrar por pantalla, lo que estaba grabando nuestro Kinect. Para ello nos fue muy útil la práctica de OpenCV realizada en el laboratorio, lo primero por tanto, es inicializar dicha librería, creando una ventana donde posteriormente se plasmarán las imágenes tomadas. Posteriormente tenemos un evento que imprimirá indefinidamente dichas imágenes. Lo descrito anteriormente ha sido implementado de la siguiente manera:

```
def initOpenCV():
    global videoImage
    # Creamos las ventanas e imagenes necesarias con OpenCV
    cv.NamedWindow('Video', cv.CV_WINDOW_AUTOSIZE)
    cv.MoveWindow('Video', 0,0)
    videoImage = cv.CreateImage((640,480), cv.IPL_DEPTH_8U, 3)

def displayOpenCV(image, window_name):
    cv.SetData(videoImage, image)
    # Dibuja la informacion recibida por la camara del kinect y la posicion de la mano
    if hands:
        for handID in hands:
            hand = hands[handID]
            center = (int(hand['currentPos'][0]), int(hand['currentPos'][1]))
            cv.Circle(videoImage, center, 30, (0.0, 0.0, 255.0), cv.CV_FILLED )

    cv.ShowImage(window_name, videoImage)
```

1 Código de OpenCV

Una vez teníamos la imagen de nuestro dispositivo, necesitábamos que nos detectara las manos, aquí es donde entran las librerías de OpenNI. Estas librerías también requieren ser inicializadas, y después hemos utilizado las funciones que nos proporcionan para realizar el seguimiento de las manos. Dichas funciones tienen un funcionamiento más complejo, pero podríamos resumirlas en las siguientes líneas de código:

```
def initOpenNI():
    global context, video, depth, handsGenerator, gestureGenerator
    try:
        # Inicializamos OpenNI
        context = Context()
        context.init()
        context.init_from_xml_file('OpenniConfig.xml')
        video = context.find_existing_node(NODE_TYPE_IMAGE) # RGB
        depth = context.find_existing_node(NODE_TYPE_DEPTH) # DEPTH
        # Manos
        handsGenerator = HandsGenerator()
        handsGenerator.create(context)
```

```
def create(src, id, pos, time):
    # Guarda informacion sobre la nueva mano detectada:
    #   currentPos: posicion donde fue detectada
    #   cache: pequeña cache que elimina ruido
    pos = openNItoOpenCVCoords(pos)
    hands[id] = {'currentPos':pos, 'cache': [pos]*10}
    print 'Detectada mano en', pos

def update(src, id, pos, time):
    # Guarda informacion acerca de la nueva posicion y actualiza la cache
    pos = openNItoOpenCVCoords(pos)
    pos = appendToCache(hands[id]['cache'], pos)
    hands[id]['currentPos'] = pos

def destroy(src, id, pos):
    # Elimina la informacion de la mano
    del hands[id]
```

2 Inicialización y seguimiento de manos con OpenNI

Una vez tenemos la posición de las manos, detectadas con OpenNI, para saber que realmente están en seguimiento, dibujamos un punto sobre ellas utilizando OpenCV. Las líneas de código encargadas de pintar dicho círculo se encuentran en la imagen 1.

El siguiente paso a realizar fue introducir OpenGL, con el que dibujamos una nueva ventana, donde se encuentra la imagen o diapositiva que vamos a pintar. Además de cuatro botones, uno en la esquina superior derecha, que será para pintar; otro en la esquina superior izquierda, que será para borrar; y por último dos botones en la parte superior central, que nos servirán para pasar la siguiente diapositiva (o imagen), o a la anterior. Para dibujar los botones, hemos utilizado una textura para la ventana de OpenGL, y un rectángulo negro en la ventana de OpenCV. El código de inicialización de OpenGL y de los botones texturizados es como sigue:

```
def initOpenGL():
    global drawicontex
    # Inicializaciones
    glutInit(sys.argv)
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
    glutInitWindowSize(640,480)
    glutInitWindowPosition(640, 0)
    glutCreateWindow('Canvas')
    # Callbacks
    glutDisplayFunc(display)
    glutIdleFunc(idle)
    glutReshapeFunc(reshape)
    glutKeyboardFunc(keyboard)
    # Texturas (iconos)
    glEnable(GL_TEXTURE_2D)
    drawicontex = texFromPNG('icons/draw-freehand-icon.jpg')
    glutMainLoop();
```

```
def texFromPNG(filename):
    img = Image.open(filename)
    img_data = np.array(list(img.getdata()), np.uint8)
    texture = glGenTextures(1)
    glPixelStorei(GL_UNPACK_ALIGNMENT,1)
    glBindTexture(GL_TEXTURE_2D, texture)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, img.size[0], img.size[1], 0, GL_RGB, GL_UNSIGNED_BYTE, img_data)
    return texture
```

3 Inicialización y botones con OpenGL

Ahora que tenemos las manos en seguimiento, un punto sobre ellas, y el botón ubicado, podemos utilizar una mano para seleccionar el botón, cuyo círculo será rojo, y la otra para pintar, cuyo círculo será amarillo. Entonces hemos encontrado un nuevo problema. Las coordenadas donde detecta las manos OpenNi, no son las mismas que donde queremos pintar el círculo con OpenCV, y además, las coordenadas donde nos pinta el círculo OpenCV, tampoco son las mismas que las coordenadas donde tenemos que pintar la imagen con OpenGL. Por tanto hemos necesitado dos funciones más, para traducir desde cada uno de estos sistemas de coordenadas. Dichas funciones han sido implementadas de la siguiente manera:

```
def openNItoOpenCVCoords(openNIPoint):
  openCVPoint = depth.to_projective([openNIPoint])[0]
  return (int(openCVPoint[0]), int(openCVPoint[1]))

def openCVtoOpenGLCoords(openCVPoint, wwidth=640, wheight=480):
  x = (openCVPoint[0] - wwidth/2.0) / (wwidth/2.0)
  y = (wheight/2.0 - openCVPoint[1]) / (wheight/2.0)
  print openCVPoint, x, y
  return (x, y)
```

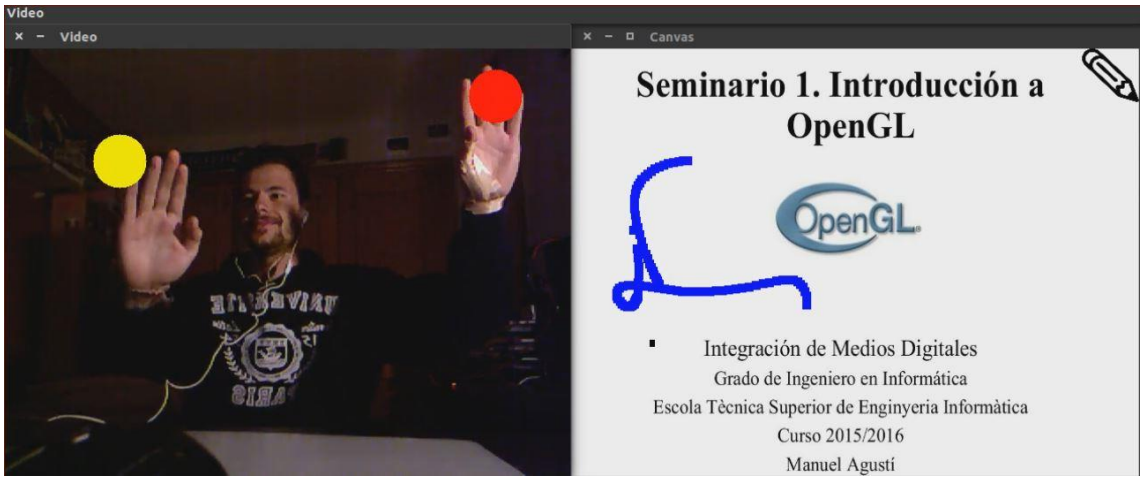
4 Conversiones de coordenadas

Y por último, la parte divertida, la función que pinta utilizando OpenGL, en las coordenadas (ya convertidas), sobre una imagen o transparencia. Hemos utilizado GL_POINTS sobre las coordenadas pertinentes, que corresponden exactamente con las de la imagen que se imprime simultáneamente capturada por el Kinect. La función queda así implementada:

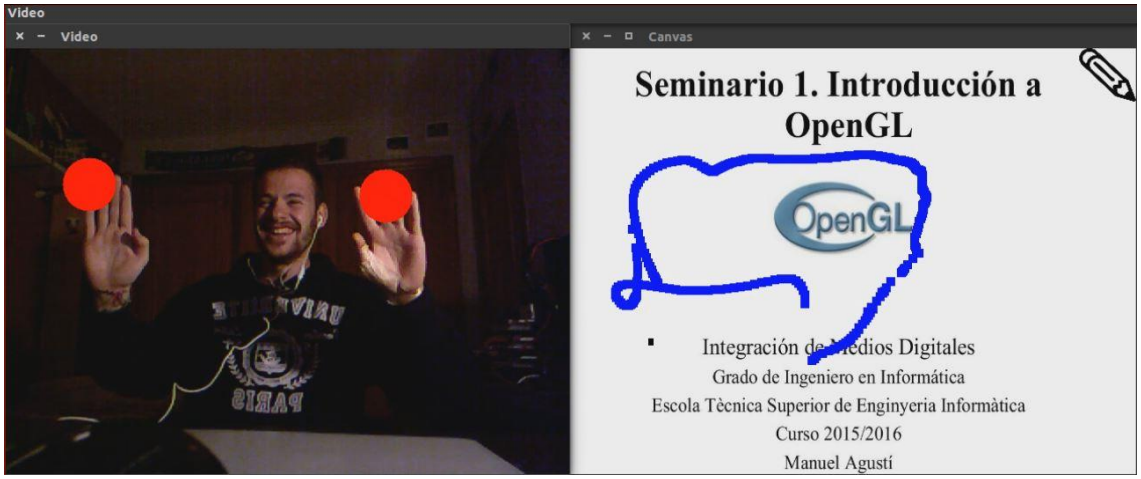
```
def display():
    for handid in [handid for handid in hands.keys() if hands[handid]['drawing']]:
        vertex, vertey = openCVtoOpenGLCoords(hands[handid]['currentPos'])
        glEnable(GL_POINT_SMOOTH)
        glPointSize(10.0)
        glColor(0.0,0.0,1.0)
        glBegin(GL_POINTS);
        glVertex2d(vertex, vertey)
        glEnd();
        glutSwapBuffers();
```

5 Dibujando a partir de la posición de las manos

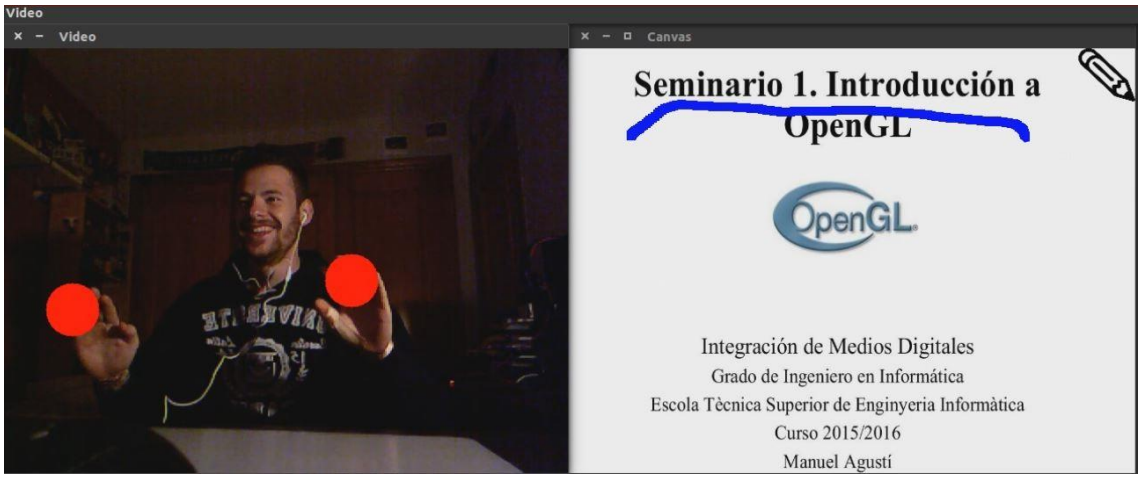
Finalmente con el primer prototipo de la aplicación, hemos empezado a jugar y tenemos las siguientes capturas como muestra de nuestro proyecto:



6 Dibujando al pulsar el botón



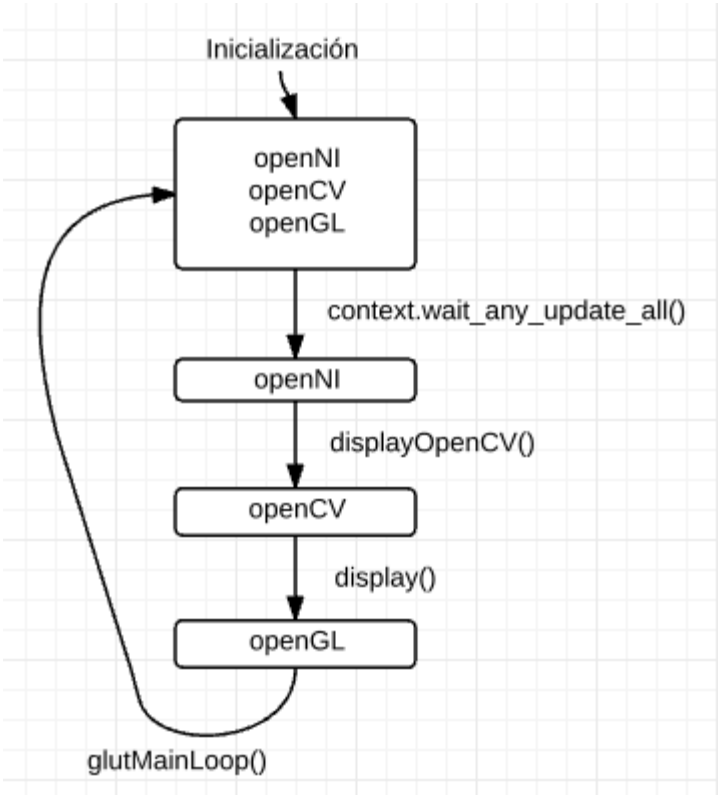
7 Aprendiendo a usar nuestra aplicación



8 Al fin lo hemos conseguido

Aplicación en Linux

Para la implementación de nuestro proyecto en Linux, realizada en Python y utilizando OpenNI, OpenCV y PyOpenGL, hemos partido de un ejemplo que mediante unos gestos específicos, seleccionaba un color, y pintaba sobre una imagen en blanco. En este ejemplo la forma de seleccionar el color, nos ha parecido bastante incómoda, por lo que hemos optado por cambiarla. La idea era que tenías que simular un clic con la mano, acercándola y alejándola rápidamente hacia la cámara, dicho gesto es bastante impreciso y provoca que, además de los problemas para que el gesto se detecte con cierta facilidad y naturalidad, empecemos a pintar y dejemos de hacerlo en un punto que no es el deseado. En nuestro caso tratábamos de realizar el gesto en numerosas ocasiones sin que el programa detectara que lo habíamos hecho, por lo que decidimos cambiarlo. Para solucionar este problema, en nuestro proyecto la forma de empezar a pintar, consiste en colocar una de las manos sobre un botón durante X segundos, y entonces pintamos con la otra mano. Para dejar de pintar, simplemente tenemos que quitar la mano del botón y la mano opuesta ya no estará pintando.



9 Diagrama de flujo del programa

Proyecto en Windows

Finalmente el proyecto a realizar en Windows, utilizando el SDK de Windows, lo dejamos a un lado para centrarnos en la aplicación en Linux, ya que sería una aplicación muy similar, pero cerrada a un único sistema operativo, ya que si creamos la aplicación utilizando el SDK propio del Kinect, no podríamos hacerla Open Source. La diferencia entre las aplicaciones radicaría en la base del código, ya que uno sería en Python mientras que el otro sería en C#, y que la detección de manos en el caso de Windows es mucho más precisa y fácil de implementar. Sin embargo la parte del programa que se encarga de pintar a partir de nuestros gestos, sería prácticamente idéntica ya que utilizaríamos OpenCV en ambos casos. En conclusión, para conseguir que el resultado final del proyecto sea más robusto, hemos decidido centrarnos sólo en una de las aplicaciones, decidiéndonos por Linux ya que el resultado final sería más portable, y además utilizamos la versión del Kinect disponible en los laboratorios de clase.

Conclusión

Iniciamos el proyecto con una idea clara, con muchas ideas e implementaciones, incluso nos propusimos realizar dos aplicaciones, una para explorar el SDK propio de Windows, y otra para experimentar con las herramientas Open Source que rodean a dicho dispositivo, las cuales no conocíamos ni sospechábamos de su existencia. Cuando tuvimos que empezar con las implementaciones, nos dimos cuenta de que llevar los dos proyectos en paralelo, iba a causarnos más problemas de lo esperado, y retrasaría demasiado la fecha en que podríamos entregarlo, por lo que nos decidimos por acabar la implementación en Linux que es más portable, pese a que puede tener algo menos de precisión en, por ejemplo, la detección de manos. Respecto al dispositivo, es un hardware muy interesante y con muchísimo potencial, pero su contexto de uso es bastante limitado, ya que por ahora, su uso se limita bastante a la realización de ciertos gestos o movimientos, que al ser realizados de manera continua resultan muy cansados.

En conclusión, este trabajo nos ha ayudado a ver, como debemos trabajar en un proyecto más de investigación, y menos de implementación, ya que en algunos casos la documentación que hemos encontrado es realmente escasa, y eso nos ha hecho consumir una cantidad de tiempo con la que no contábamos en un principio. Para un próximo proyecto, sabríamos plantearlo de otra manera y podríamos así manejar mejor los plazos, planificando estos posibles contratiempos.

Bibliografía

- https://openkinect.org/wiki/Main_Page [1]
- https://openkinect.org/wiki/Getting_Started#Ubuntu_Manual_Install [2]
- https://sigmaoctantis.wordpress.com/2015/07/07/kinect-installation-in-ubuntu-14-10/#comments [3]
- https://bitbucket.org/samirmenon/scl-manips-v2/wiki/vision/kinect [4]
- http://docs.opencv.org/2.4/doc/tutorials/introduction/linux_install/linux_install.html [5]