Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Integración de presencia humana virtual en un simulador de un espacio inteligente usando cámaras de tiempo de vuelo

Autor: Álvaro de la Fuente Díaz

Tutor: Javier Macías Guarasa

2018

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Electrónica de Comunicaciones

Trabajo Fin de Grado

Integración de presencia humana virtual en un simulador de un espacio inteligente usando cámaras de tiempo de vuelo

Autor: Álvaro de la Fuente Díaz

Tutor: Javier Macías Guarasa

Tribunal:

Presidente: Marta Marrón Romera

Vocal 1º: Cristina Losada Gutiérrez

Vocal 2º: Javier Macías Guarasa

Calificación: .....

Fecha: .....

"El fracaso es una opción. Si las cosas no están fallando, no estás innovando lo suficiente."
Elon Musk

Agradecimientos

A todos los que la presente vieren y entendieren.

Inicio de las Leyes Orgánicas. Juan Carlos I

El presente trabajo es el fruto de muchas horas invertidas. Ya no solo en su elaboración, sino también es el reflejo del esfuerzo y los conocimientos adquiridos durante los años dedicados plenamente al Grado.

Por lo tanto, para mí significa por un lado el broche final a mi primera etapa universitaria y por otro lado una puerta abierta a lo que será mi futuro como Ingeniero en Electrónica de Comunicaciones.

Es por ello que quiero agradecer a todas las personas que han estado a mi lado durante todo este tiempo y que de un modo u otro me han aportado algo.

Gracias.

Resumen

El trabajo realizado consiste en la integración en un espacio inteligente virtual de una representación de uno o varios usuarios reales a través de la pose captada mediante una cámara de profundidad en un espacio real.

En dicho espacio se reflejarán los movimientos que los usuarios realicen a través del procesado de la información captada por el dispositivo Kinect.

Como resultado se obtiene una aplicación capaz de simular una habitación en la que se inserta la presencia humana detectada, así como el tratamiento de las diferentes muestras obtenidas de forma que se pueda representar dicho mapa de profundidad.

Palabras clave: Cámara de profundidad, Kinect, espacio virtual, mapa de profundidad , detección de pose.

Abstract

This work is about the integration in an intelligent virtual space of a representation of one or more useres through the pose captured by a depth cam in a real space.

In those space will be reflected the movements users made through the information processed captured by Kinect.

As a result, an application able to simulate a room in which its inserted human presence, as well as the different samples treatment such a way that depth map can be represented, it will be obtained.

Keywords: Depth cam, Kinect, virtual space, depth map, posed detected.

Resumen extendido

El presente trabajo: *"Integración de presencia humana virtual en un simulador de un espacio inteligente usando cámaras de tiempo de vuelo"*, consiste en integrar la presencia de uno o varios usuarios en un entorno virtual simulado. El objetivo principal de este proyecto ha sido la obtención de información a través de una cámara de profundidad de un entorno de trabajo, interpretarla y posteriormente integrarla en un espacio simulado.

El proyecto parte de un trabajo previo en el que se desarrolló la simulación del entorno de trabajo virtual basado en la librería de libre distribución `OpenGL`. Dicho espacio virtual fue cedido por el tutor y profesor de la universidad Javier Macías Guarasa. Tras la obtención del código de simulación, se realizó un análisis de funcionalidades así como de posibles modificaciones que pudieran llevarse a cabo para la adaptación de dicho espacio virtual al objeto de estudio de este trabajo.

De entre todas las cámaras de profundidad disponibles en el mercado, se ha seleccionado para desarrollar el proyecto la cámara Kinect de Microsoft.

El dispositivo elegido es común y de fácil acceso ya que se distribuye como un periférico para la famosa consola Xbox 360. Este dispositivo, lejos de ser un complemento de juego simple, presenta unas caracterísiticas que lo hacen muy interesante y de gran utilidad para diferentes proyectos de investigación.

Para implementar el sensor de profundidad, Kinect cuenta con un láser de infrarrojos y un detector de infrarrojos.

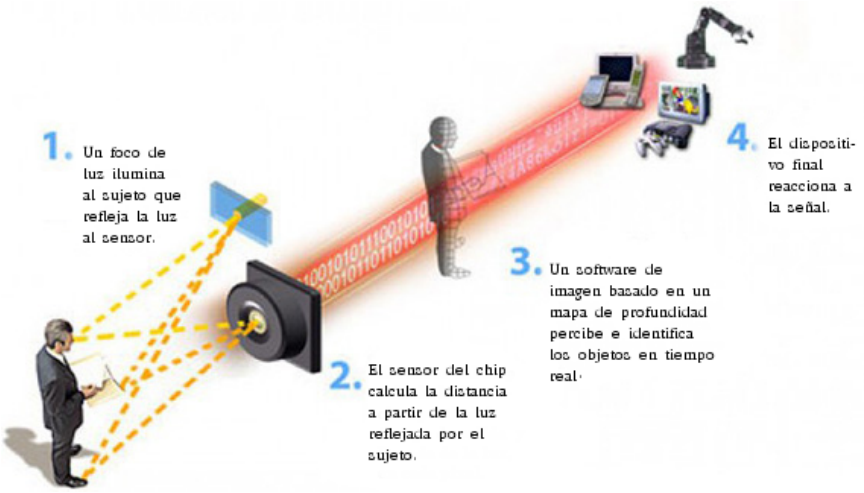


Figura 1: Esquema de funcionamiento de una cámara de profundidad.

Para conocer la profundidad a la que se encuentran los objetos presentes en la escena, el dispositivo proyecta una matriz de puntos de luz infrarroja sobre el espacio. Estos se proyectan en los objetos, y sus proyecciones son capturadas por el sensor infrarrojo y a través de unos algoritmos de triangulación se

calculan las diferentes medidas de profundidad. En la figura 1 se puede ver un esquema de como funcionan los sensore de profundidad a grandes rasgos.

Una vez comentado el sensor de profundidad empleado, cabe destacar que el trabajo se ha codificado utilizando el lenguaje de programación C++ empleando las librerias `OpenNI`, `OpenGL` y `OpenCV`.

A continuación se procede a presentar las partes en las que ha sido estructurado el trabajo.

El proyecto cuenta con dos partes diferenciadas aunque estrechamente unidas:

- Por un lado, para la obtención de las medidas de distancia existentes entre la cámara y la escena, se utilizan las muestras de profundidad capturas por Kinect.

  Estas muestras son codificadas y plasmadas en dos dimensiones en relación a su magnitud. De esta forma, las medidas cuya magnitud sea menor se verán representadas con colores más claros; y las mayores tomarán una gama de color más oscura. Esto permite dotar de realismo a la imagen y poder apreciar mejor las diferentes profundidades existentes en la escena.

  Con este tratamiento se obtiene la representación de un mapa de profundidad en el que se puede observar claramente el entorno de trabajo que se encuentra delante de la cámara.

  Aunque Kinect también cuenta con una cámara RGB, se ha decidido plasmar la visión que tiene el sensor a través de las muestras de profundidad haciéndola así independiente de las condiciones lumínicas, ya que al ser profundidad funciona perfectamente en ausencia de luz visible.

  Una vez obtenido el array de muestras, estas se codifican y representan gracias a `OpenCV`.

- En esta segunda parte, en base a lo anteriormente comentado, se produce la integración de presencia humana virtual dentro de un espacio virtual simulado.

  Para desarrollar esto, se ha utilizado `OpenNI` que junto con `NITE`, middleware de código abierto desarrollado por PrimeSense, es posible detectar la existencia de personas, su postura y su posterior seguimiento a lo largo de la escena.

  Una vez es detectada la persona, se representan las principales articulaciones de la misma (cabeza, cuello, hombros, codos, manos, tórax, caderas, rodillas y pies) dentro del espacio virtual empleando la librería de representación gráfica `OpenGL`.

Figura 2: Ejemplo de integración de la presencia de un usuario dentro del espacio virtual.

En la figura 2 se pueden ver las dos ventanas generadas durante la ejecución de la aplicación, una por cada una de las partes mencionadas anteriormente.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1 Presentación

Desde sus orígenes, el ser humano ha ido desarrollando numerosas máquinas y mecanismos para mejorar su calidad de vida. La forma en la que el hombre ha interactuado con estos dispositivos ha ido evolucionando al mismo ritmo y de forma paralela a la correspondiente tecnología de cada época.

Actualmente vivimos en una sociedad en la que constantemente se están emitiendo señales y estímulos. Y de esto nace la necesidad de querer medir todo ello, de forma que esta información desestructurada procedente de diversas fuentes sea moldeada y transformada en datos útiles.

Existen múltiples sensores capaces de registrar hasta el más mínimo detalle del entorno que nos rodea.

Gracias a la era de la digitalización y a los avances de la tecnología, todas esas medidas son mucho más fiables y precisas.

Hoy en día, históricamente hablando, podríamos situarnos en lo que podría ser el orígen de la era tecnológica. Y al igual que las primeras máquinas se accionaban con interruptores y palancas, los ordenadores más modernos también se accionan con pequeños interruptores alojados en el teclado. Pero si observamos el ritmo al que avanzan estas tecnologías, somos conscientes de que cada vez más, se está orientando todo al mundo inalámbrico en el que no existen los cables y todo está conectado.

Es por ello que nace la necesidad de buscar nuevas formas de comunicarse con el entorno y poder manejar las nuevas máquinas del futuro. Y tal y como apuntan muchos de los expertos, el futuro está en la biométrica.

Los gestos son uno de los candidatos mejor posicionados como sustitutivo de la interación que se realizaba con las máquinas del pasado. Esto permite que cualquier persona pueda ser capaz de iteraccionar con las máquinas, independientemente de su condición física.

Actualmente existen controladores gestuales en los que la comunicación se hace a través de la interpretación de los gestos que realiza el indivíduo o incluso oculares en los que una persona con discapacidad funcional es capaz de comunicarse con un ordenador a través del movimiento de sus ojos.

En este marco de la comunicación sin cables es donde se encuadran los sensores de profundidad. Estos dispositivos puede obtener información de la distancia existente entre el dispositivo y los elementos del entorno.

Todo esto es utilizado para detectar objetos y/o personas, así como captar gestos que puedan hacer los individuos y poder así interaccionar con el entorno, tal como se ha comentado anteriormente.

1.2 Objetivos del trabajo

El objetivo principal que persigue este trabajo de fin de grado es el de la integración de un entorno de trabajo virtual con uno real, en el que a través del dispositivo Kinect se tomen muestras de profundidad existentes entre la cámara y la escena.

A través de la librería `OpenNI` se calculan las posiciones de los puntos más característicos de una persona de forma que al representarlos se obtiene un 'esqueleto' que emula la presencia humana dentro de la escena simulada. Aparte, las medidas de profundidad obtenidas también son representadas de forma que se pueda realizar una comparativa del entorno real de trabajo y el simulado mediante el ordenador.

El trabajo cuenta con una serie de objetivos específicos los cuales serán detallados a continuación:

1. **Estudio de la tecnología de las cámaras de profundidad.**

   Se deberá realizar un estudio de las tecnologías existentes en las cámaras de profundidad y su funcionamiento para su comprensión e integración dentro del trabajo.

   Cabe destacar que existen principalmente dos tipos, los cuales serán detallados posteriormente: cámaras de luz estructurada y cámaras de tiempo de vuelo.

2. **Analizar el código disponible del entorno virtual.**

   Será necesario análizar el código cedido por el tutor del que ha partido el trabajo.

   Se trata de una aplicación capaz de realizar la simulación del entorno de trabajo virtual. Dicho código se empleó en proyectos anteriores por lo que cuenta con numerosas funciones que será necesario analizar y decidir cuales pueden ser de utilidad para su posterior adaptación

3. **Analizar el código disponible del detector de pose.**

   Antes de iniciar el desarrollo del trabajo propiamente dicho, será necesario realizar un estudio con su posterio análisis de `OpenNI`.

   `OpenNI`, es un framework de código abierto que provee APIs para el desarrollo de aplicaciones que utilicen interacción natural para la interacción con el usuario. Es decir, iteración basada en los sentidos humanos.

   Una de las APIs desarrollada por `OpenNI` es '`User Tracker`'. Esta aplicación es capaz de realizar y representar un seguimiento de la pose humana dentro de la escena captada por el dispositivo Kinect.

4. **Comprobar las funciones del código para su adaptación e integración.**

   En primer lugar será necesaria la instalación de `OpenNI` así como del middleware `NITE (Natural Iteraction Technology for End-user)` para poder utilizar Kinect.

   Una vez realizado el proceso de instalación y comprobado que Kinect está tomando datos del entorno de forma correcta, se procede al análisis de la funcionalidad de la API '`User Tracker`'.

   De este modo se parte de una funcionalidad ya conocida y facilita la integración con el espacio virtual.

5. **Integrar mecanismos de interacción con el entorno de trabajo virtual a partir de los datos proporcionados por el sistema Kinect.**

   El objetivo final y a su vez el principal, es el de integrar la detección de pose obtenida a través de `OpenNI` dentro del espacio virtual, es decir, representar la presencia humana captada por Kinect dentro del entorno de trabajo simulado.

1.3 Motivación

A la hora de la elección del trabajo de fin de grado me decanté por este tema ya que me llamó la atención la idea de virtualizar un entorno y poder integrar dentro de él las personas detectadas por una cámara.

A lo largo del Grado se estudia principalmente hardware y es por eso, que decidí abordar este proyecto como una forma de ganar conocimiento en el ámbito de la visión artificial y el procesamiento algoritmico.

Otra de las cosas que despertó mi interés era la posibilidad de poder utilizar el dispositivo Kinect fuera del ámbito de los videojuegos y conocer el gran potencial que tiene este dispositivo.

1.4 Estructura del trabajo

El trabajo cuenta con una estructura dividida en dos grandes bloques bien diferenciados: En primer lugar tiene cabida una parte puramente documental, en la que se realiza un estudio de la cámara de profundidad empleada así como las librerías de código utilizadas. Y una segunda parte experimental en la que se codifica el trabajo y se realizan diferentes pruebas donde se comprueba su correcto funcionamiento.

Dentro de estos bloques, la memoria está organizada en una serie de capítulos tal como se detalla a continuación:

1. **Aspectos teóricos**: en este capítulo se detallará el funcionamiento de los diferentes tipos de cámaras de profundidad que existen; se estudiarán las librerías de libre distribución `Libfreenect`, `OpenNI`, `OpenGL` y `OpenCV` con las que se obtendrán, tratarán y representarán las muestras tomadas con Kinect.

2. **Implementación y pruebas**: en este segundo capítulo se procederá a la explicación del trabajo desarrollado así como la exposición de los datos obtenidos durante la ejecución del programa.

3. **Conclusiones y líneas futuras**: aquí se citarán una serie de conclusiones extraídas a lo largo de todo el proceso, así como unas posibles mejoras o modificaciones que pudieran ser desarrolladas en un futuro.

4. **Manual de usuario**: aquí se desarrollará un manual en el que se detallarán los pasos a seguir por el usuario, así como ciertos aspectos a tener en cuenta a la hora de poder ejecutar la aplicación de forma correcta. Esto se encuentra dividido en dos bloques: Manual del espacio virtual y Manual de funcionamiento.

5. **Procedimientos de instalación**: en esta penúltima sección del trabajo tendrá cabida lo referente al proceso de instalación de las diferentes librerías utilizadas en el desarrollo del trabajo.

6. **Herramientas**: por último, en este apéndice se plasmarán las diferentes herramientas utilizadas durante el desarrollo del trabajo.

Capítulo 2

Aspectos teóricos

2.1 Introducción

El principal objetivo de este trabajo es la integración de la pose de uno o varios usuarios obtenida mediante la cámara Kinect dentro de un espacio virtual tridimensional, reflejándose en este las diferentes interacciones que realice el usuario en la escena captada por el dispositivo.

En los siguientes apartados se detallarán los diferentes aspectos teóricos analizados a la hora del desarrollo del trabajo: Kinect, las librerías empleadas y la relación de los sistemas de referencias existentes entre la 'visión del sensor' y la del entorno virtual.

2.2 Cámaras de profundidad

En la actualidad, se están utilizando mucho las denominadas cámaras de profundidad. Estos dispositivos son capaces de obtener información sobre la medida de profundidad o distancia existente entre la cámara y la escena.

Dentro de las cámaras de profundidad se puede distinguir, principalmente entre dos tipos, en función del método empleado para la obtención de la información: cámaras de profundidad basadas en luz estructurada y cámaras basadas en tiempo de vuelo [1].

- **Cámaras de profundidad basadas en luz estructurada**: estos dispositivos están basados en la proyección de un patrón de luz infrarrojo en forma de red o puntos. Una vez proyectado el patrón, la medida de distancia se obtiene a partir de la deformación del mismo al alcanzar los objetos de la escena.

- **Cámaras de profundidad basadas en tiempo de vuelo**: en este caso, las cámaras obtienen la medida de la distancia en función del tiempo que tarda una onda infrarroja emitida por los focos de la cámara en viajar hasta un punto y volver al sensor. La medida de ese tiempo que tarda la señal en ir y venir, se determina a partir de la diferencia de fase entre la onda enviada y la onda reflejada. Esto permite obtener una medida de profundidad para cada píxel.

  Estos sensores presentan una mayor velocidad de medida y mayor precisión con respecto a los de luz estructurada.

2.2.1 Kinect

Kinect es la cámara de profundidad más conocida, asequible y accesible del mercado al ser desarrollada por Microsoft. Buscando en la omnipresente Wikipedia, podemos encontrar que Kinect se define como [2]: *'Un controlador de juego libre y entretenimiento desarrollado por Microsoft para la videoconsola Xbox 360. Kinect permite a los usuarios controlar e interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuego tradicional, mediante una interfaz natural de usuario que reconoce gestos, comandos de voz y objetos e imágenes'.*

Fue lanzada a finales de 2010 de la mano de Microsoft, aunque en un primer momento se le dió el nombre de 'Project Natal', más tarde pasó a llamarse Kinect.

El hardware fue desarrollado por la empresa PrimeSense y Microsoft aportó los conocimientos obtenidos durante 20 años de investigaciones detrás de algoritmos de reconocimientos de personas y gestos.

Sus características y prestaciones pronto llamaron la atención de la comunidad hasta que Adafruit lanzó el concurso 'The Open Kinect Project'.

Este concurso consistía en la creación de un controlador o driver de código abierto para Kinect de forma que pudiera funcionar con un ordenador. El ganador fue Héctor Martín Cantero, un jóven madrileño estudiante de informática que consiguió desarrollar un controlador que responde a movimientos en profundidad y da imágenes en RGB en una ventana de OpenGL.

Gracias a esto, Microsoft se cercioró del gran potencial presente en su dispositivo y poco después lanzó la primera versión del 'SDK Kinect' [3], la cual consiste en una librería que facilita la interacción con el dispositivo Kinect. Básicamente, es una librería que obtiene información sobre los esqueletos y las articulaciones de un usuario.

El sensor Kinect es una barra horizontal de 23 cm de longitud conectada a una pequeña base circular con un eje de articulación de rótula, y está diseñado para ser colocada de forma longitudinal por encima o por debajo de la pantalla. En la figura 2.1 se puede ver el aspecto del dispositivo Kinect.

Las principales características hardware del dispositivo son:

- Sensor de profundidad de 320x240 de 16 bits.
- Cámara RGB VGA de 640x480 de 32 bits.
- Array de 4 micrófonos con cancelación de eco de 16 bits con una frecuencia de muestreo de 16 kHz.
- Campo de visión horizontal de 57º.
- Campo de visión vertical de 43º.
- Rango del tilt físico de ± 27º.
- Rango del sensor de profundidad de entre 1.2m y 3,5m.
- Puede detectar hasta 6 personas, obteniendo 20 puntos de cada uno.

Dentro de las cámaras de profundidad, Kinect es del tipo de luz estructurada.

Para obtener las medidas, el dispositivo cuenta con una cámara de infrarrojos y un proyector de luz infrarroja.

Este proyector se encargará de iluminar el entorno con un patrón de puntos con una estructura conocida. El patrón se proyectará sobre la escena, provocando que los diferentes puntos infrarrojos se deformen al alcanzar los objetos y cuerpos de la imagen tal como se puede ver en la imagen 2.2.

Figura 2.1: Aspecto de la cámara Kinect

Para obtener las muestras y conocer la profundidad, el sensor infrarrojo observa cada uno de los puntos del patrón proyectados en la escena y lo compara con la que sabe que es la posición de equilibrio. Una vez comparado mediante unos algoritmos de triangulación, es capaz de calcular la profundidad de cada uno de esos puntos.

Haciendo esto para todos los puntos del patrón proyectado, se obtiene una imagen de profundidad que describe el espacio en el que se encuentra la cámara.

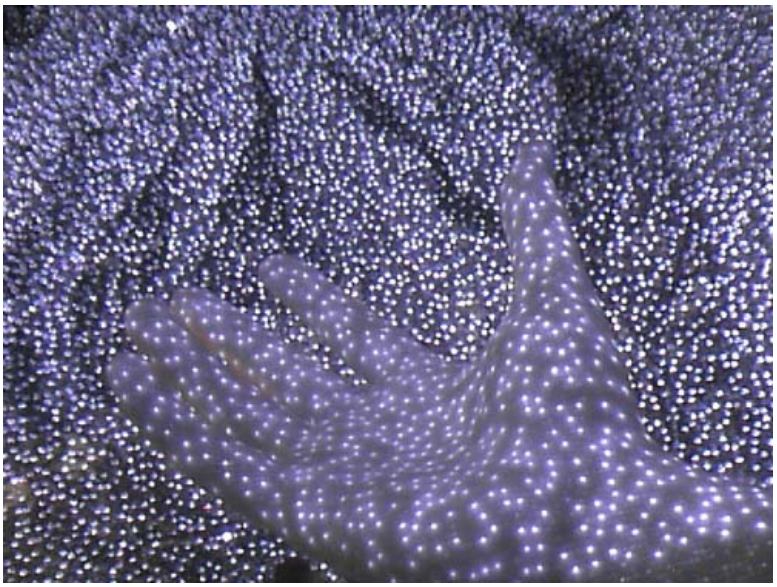


Figura 2.2: Patrón de luz infrarroja emitido por Kinect y proyectado sobre una mano.

Aparte de sus buenas prestaciones, como todo, Kinect también tiene una serie de limitaciones que hacen que la profundidad de ciertas zonas de la escena no se pueda calcular o no se haga de forma correcta. El origen de estas limitaciones se debe tanto a la forma en la que está construido el dispositivo, como a la naturaleza presente en la escena.

Seguidamente se exponen dichas limitaciones:

- El patrón de puntos infrarrojos no cubre de forma continua la superficie de los objetos, lo que implica que algunos píxels tienen que ser obtenidos a través de interpolación en la imagen de profundidad. Esto conlleva un margen de error que es mayor cuanto más alejado se encuentre el objeto ya que los puntos están más separados. Esto también puede provocar la aparición de sombras en los bordes de los objetos ya que algunas veces se tomará la estimación del objeto más cercano y otras las del más lejano.

- Si los objetos se encuentran a demasiada distancia del dispositivo, no podrá ser calculada la distancia a la que se encuentran, ya que la luz infrarroja se atenúa en el trayecto provocando que el sensor no pueda detectar los puntos. Por el contrario, si los objetos se encuentran demasiado próximos al dispositivo, los puntos infrarrojos no se proyectan de forma correcta para la detección del sensor.
- Si el haz de luz infrarroja se encuentra con un objeto que presente algún tipo de concavidad, se pueden producir reflexiones dobles provocando solapamiento de los puntos de luz y haciéndolos irreconocibles para el sensor.

Todos estos puntos en los que no se puede calcular la profundidad, se les asigna valor 0 obteniendo zonas oscuras al representar el mapa de profundidad.

En la figura 2.3 se puede ver un ejemplo de representación de mapa de profundidad en el que aparecen zonas oscuras debido a las limitaciones mencionadas anteriormente.

Figura 2.3: Ejemplo de representación de mapa de profundidad en el que se pueden ver zonas oscuras debido a las limitaciones de Kinect.

2.3 Librerías y drivers

Para poder utilizar Kinect en el ordenador es necesario instalar una serie de drivers y librerías.

En este caso, se ha utilizado `Libfreenect` para realizar una primera toma de contacto con la cámara y `OpenNI` junto con su middleware `NITE` para desarrollar el trabajo expuesto.

Aparte, también se han utilizado las librerías `OpenGL` y `OpenCV` a la hora de realizar la simulación del espacio de trabajo y la representación del mapa de profundidad capturado.

2.3.1 `Libfreenect`

`Libfreenect` es un driver desarrollado por la comunidad 'OpenKinect' [4]. Consiste en una comunidad abierta en la que participan personas desarrollando librerías de código abierto que permiten utilizar Kinect en Windows, Linux y Mac.

El principal objetivo de la comunidad es el desarrollo de la librería `Libfreenect`.

`Libfreenect` es un driver con el que podemos obtener datos de la cámara RGB y/o de profundidad, modificar la orientación de las cámaras a través del motor de la base, utilizar el acelerómetro, activar los leds de señalización y tener acceso al array de micrófonos.

En la figura 2.4 se puede ver un ejemplo de utilización del driver `Libfreenect` para representar tanto el mapa de profundidad como las imágenes obtenidas a través de la cámara RGB del dispositivo Kinect.

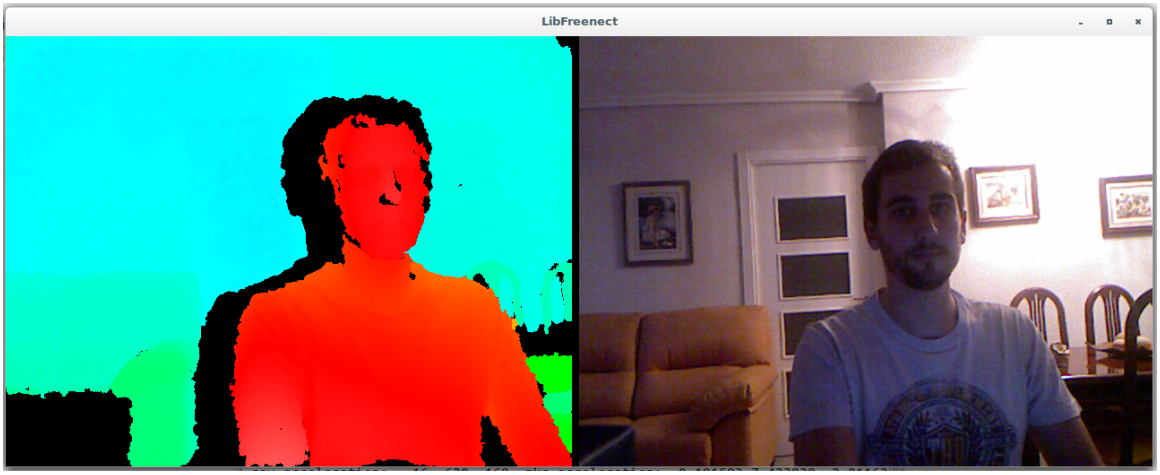


Figura 2.4: Ejemplo de representación de datos obtenidos con la cámara RGB y representación del mapa de profundidad a través de la librería `Libfreenect`.

Aunque con `Libfreenect` se tiene acceso a todos los elementos de Kinect, el trabajo no se ha desarrollado utilizando estos drivers. La razón es la existencia de `OpenNI`, una librería más completa que posibilita el desarrollo del trabajo de una forma óptima y en especial al middleware `NITE` que integra.

2.3.2 OpenNI

OpenNI [5], acrónimo de Open Natural Interaction, es definido por Wikipedia [6] como *'una organización sin ánimo de lucro impulsada por la industria centrada en la certificación y mejora de la interoperabilidad de la interfaz natural de usuario y la interfaz orgánica de usuario para dispositivos de interacción natural, las aplicaciones que usan estos dispositivos y el middleware que facilita el acceso y uso de tales dispositivos'*.

Con iteracción natural se refiere al concepto donde la iteración entre el usuario y el dispositivo se realiza basándose en los órganos de los sentidos humanos.

La organización fue creada a finales del 2010. Uno de sus principales miembros era PrimeSense, empresa a la que se debe el desarrollo del hardaware de Kinect. Un poco más tarde, esta empresa liberó sus propios drivers de código abierto junto con un middleware de detección de movimiento denominado `NITE`.

En 2014 PrimeSense fue comprado por Apple por lo que la página oficial de la organización fue cerrada. También se conoce como `OpenNI` al framework y al conjunto de módulos y drivers desarrollados por la organización para ser utilizados para las cámaras de PrimeSense. En este punto surgió SensorKinect, se trata de la adaptación de los drivers desarrollados por OpenNI para el dispositivo Kinect.

En la figura 2.5 se puede ver como `OpenNI` se puede dividir en tres capas:

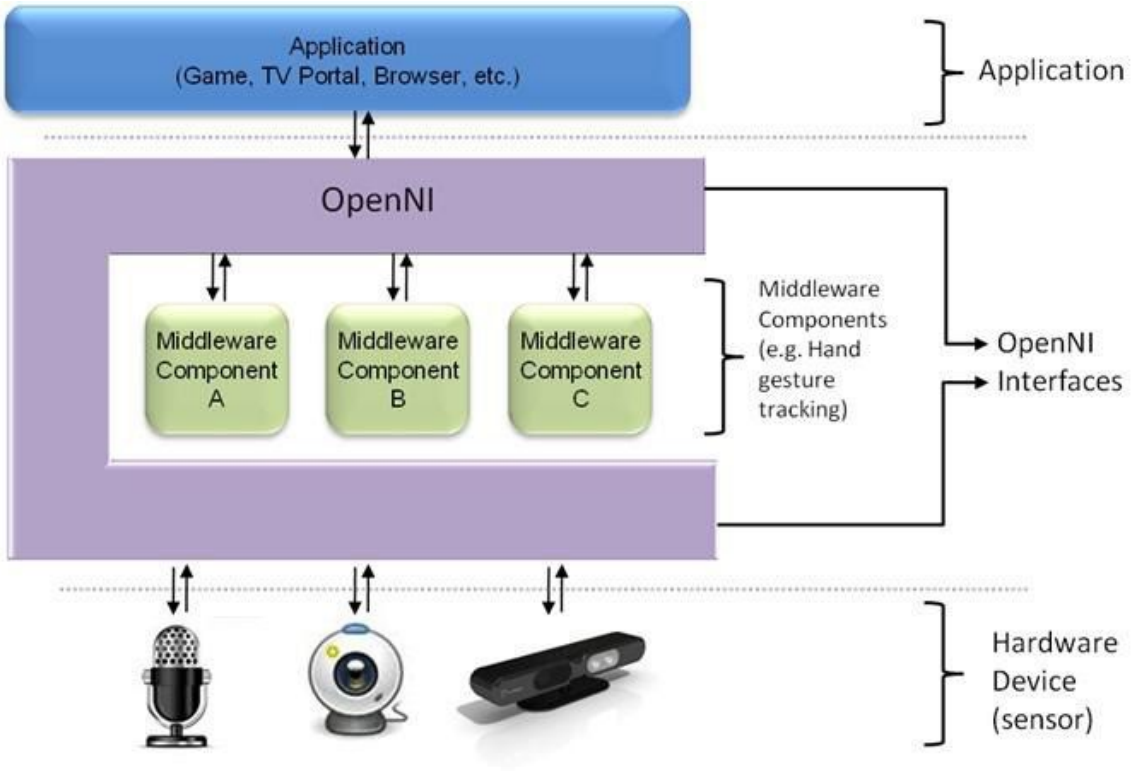


Figura 2.5: Esquema OpenNI

- La capa inferior se corresponde con los dispositivos hardware que forman Kinect de los que se puede extraer información: la cámara RGB, la cámara de profundidad o el array de micrófonos.
- La capa central corresponde a `OpenNI`, que provee las interfaces de comunicación capaces de comunicarse entre los sensores y los componentes middleware.
- Por último, la capa superior hace referencia a las aplicaciones que emplean las librerías de `OpenNI`.

Las principales ventajas que tiene `OpenNI` frente a `Libfreenect` son:

- Al estar desarrollado por varias compañías posibilita el poder utilizar el código en diferentes cámaras, haciéndolo versátil y portable.
- Aunque Apple cerró la página oficial de la organización, la comunidad ha encontrado la forma de seguir con su actividad y continuar con el desarrollo de librerías y actualizaciones haciendo que sea muy fácil encontrar información. Basta con adentrarse un poco en la web para encontrar tutoriales y manuales sobre el funcionamiento de la misma.
- La principal ventaja frente a su alternativa `Libfreenect` es la existencia de `NITE`.

`NITE (Natural Interaction Technology for End-user)` [7] es un middleware de código abierto desarrollado por PrimeSense que implementa varias características como análisis completo y tracking esqueletal, análisis de la posición y tracking de las manos o reconocimiento de gestos. En este caso, se utilizará la detección de presencia humana y su seguimiento a través de la escena.

Primeramente, el reconocimiento de personas se realiza a través de una pose de calibración y posteriormente, el tracking es realizado a través de una serie de puntos (`OpenNI` los llama 'joints') característicos

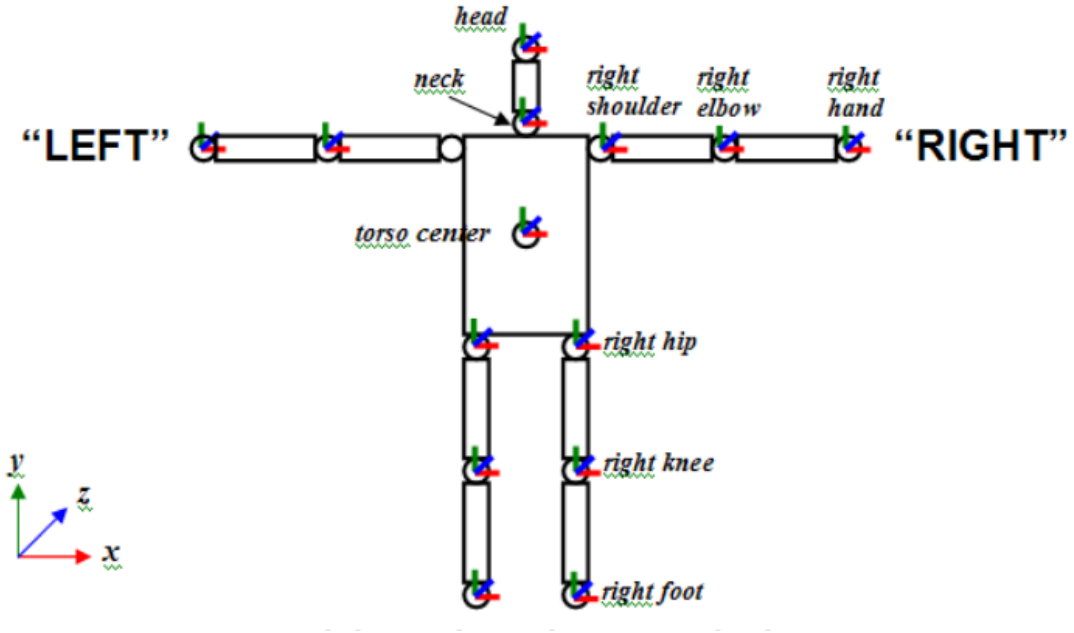


Figura 2.6: Esquema de los 'joints' que puede capturar OpenNI.

de la fisiología humana. En la figura 2.6 se puede ver un esquema de los diferentes puntos detectados por NITE.

La existencia de esta herramienta ya desarrollada, facilita enormemente el trabajo de integración de presencia humana dentro del espacio virtual.

2.3.3 OpenGL

Tal como dice la wikipedia [8], OpenGL (Open Graphics Library) *'es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada por Silicon Graphis Inc en 1992 y se usa ampliamente en CAD, realidad virtual, representación científica, visualización de información, simulación de vuelo y desarrollo de videojuegos'.*

En definitiva, OpenGL es un software que facilita la interacción con el hardware gráfico de la máquina [9] [10].

Consta de diferentes funciones que permiten desarrollar aplicaciones interactivas en las que intervienen tanto gráficos en dos dimensiones como en tres.

Está desarrollado de forma totalmente independiente del hardware por lo que puede implementarse en diversas plataformas así como diferentes sistemas operativos.

La herramienta no incluye comandos para modelar objetos complejos, todo se realiza a través de primitivas geométricas.

En la figura 2.7 se puede ver un esquema de cómo esta organizado OpenGL, y las diferentes capas que lo componen.

Las operaciones que se pueden hacer con OpenGL son:

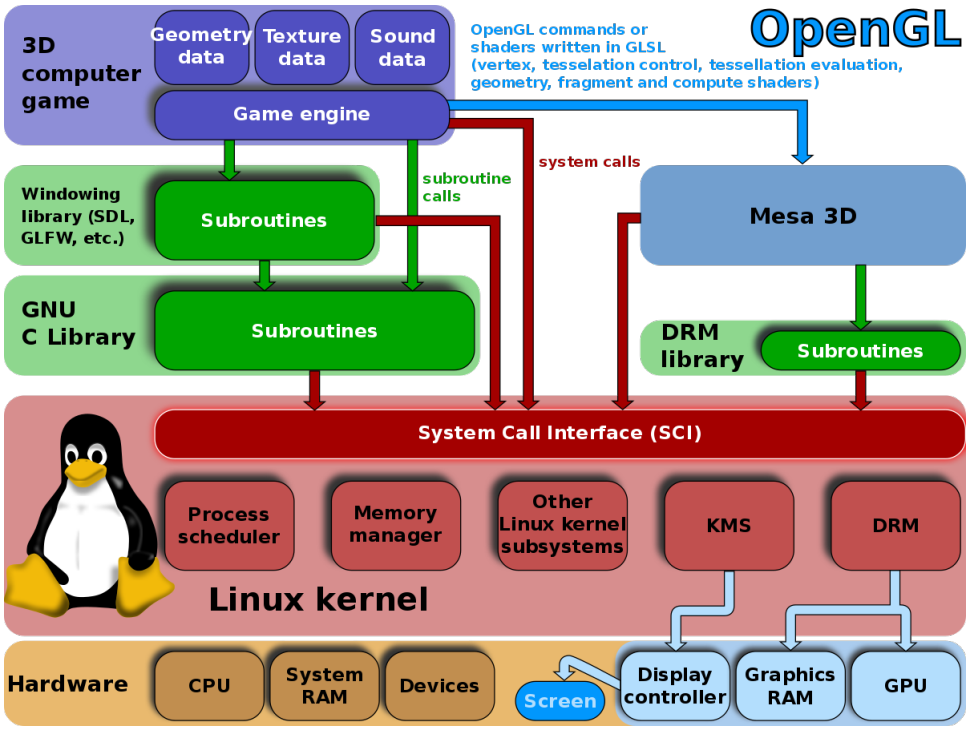


Figura 2.7: Esquema de las diferentes capas que forman `OpenGL`.

- Modelar figuras partiendo de las primitivas básicas tales como puntos, líneas y polígonos, creando descripciones geométricas de los objetos.
- Situar los objetos en el espacio tridimensional de la escena y seleccionar el punto de vista desde el cual queremos observarla.
- Calcular el color de todos los objetos. Esto puede hacerse asignando explícitamente el color en cada píxel o colocando una textura sobre los objetos.
- Convertir la descripción matemática de los objetos y la información asociada de los colores en forma de imagen virtual 3D.

Para desarrollar una aplicación utilizando la librería `OpenGL`, debemos tener en cuenta una estructura en la que primeramente se deberán inicializar varios estados que controlan el proceso de rendering y posteriormente especificar los objetos que deben visualizarse describiendo su geometría y propiedad.

2.3.4 OpenCV

`OpenCV (Open Source Computer Vision Library)` es una librería de libre distribución desarrollada por Intel. Ha sido codificada en lenguaje C/C++ y se utiliza para tratamiento de imágenes, destinada principalmente a aplicaciones de visión por computador en tiempo real [11].

Dentro del rango de áreas de aplicación destacan: interacción hombre-máquina, segmentación y reconocimiento de objetos, reconocimiento de gestos, seguimiento del moviento, estructura del movimiento y robots móviles.

Se trata de una librería multiplataforma, por lo que está disponible tanto para Linux y Windows como para Mac.

La librería `OpenCV` es una API de unas 300 funciones que se caracteriza por lo siguiente:

- Se trata de un software libre que puede ser utilizado tanto para uso comercial como no comercial.
- No utiliza librerías numéricas externas, aunque puede hacer uso de algunas de ellas, si están disponibles, en tiempo de ejecución.
- Es compatible con otras librerías desarrolladas por Intel para mejorar el rendimiento, en caso de estar disponibles en el sistema.
- Dispone de interfaces para algunos otros lenguajes y entorno.

Por todas estas características, se ha seleccionado esta librería para realizar la representación del mapa de profundidad tal como se detallará mas adelante.

2.4 Sistema de referencia

Uno de los puntos más importantes de este trabajo es el de mapear correctamente las posiciones de las 'joints' obtenidas a través de Kinect dentro del espacio virtual.

Esta tarea no es tan trivial como parece, pues existen dos sistemas de referencias distintos. La habitación simulada presenta su centro de coordenadas en una esquina y de Kinect toma como centro ella misma.

Para poder realizar la integración de la cámara dentro del espacio virtual será necesario realizar un cambio de sistemas de referencias [12] [13]. Existen dos formas de hacer este tipo de adaptaciones: transformaciones lineales y afines.

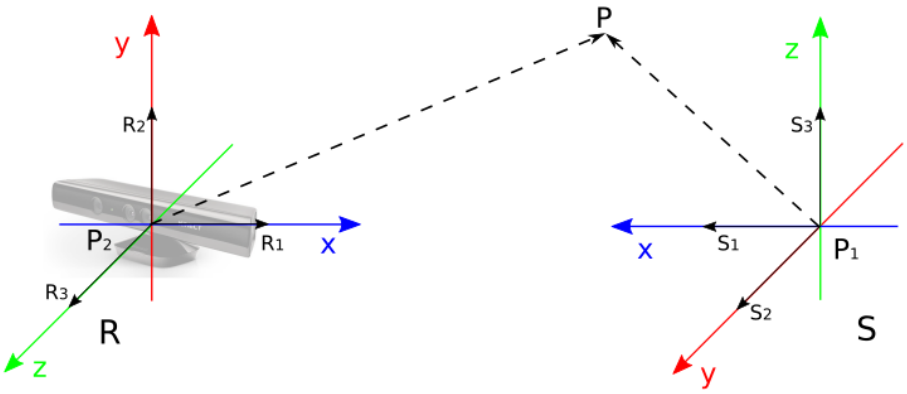


Figura 2.8: Sistemas de referencias

Sean 2 sistemas de referencia S=$\{P_1, [S_1,S_2,S_3]\}$, sistema de referencia del espacio virtual, y R=$\{P_2, [R_1,R_2,R_3]\}$, sistema de referencia asociado a la cámara Kinect, con bases asociadas ortogonales y sea Q $\in R^3$ un punto arbitrario. En la figura 2.8 se puede ver la composición de ambos sistemas de referencias.

El punto P tendrá unas coordenadas α_1, α_2, α_3 en el primer sistema de referencia y otras β_1, β_2, β_3 en el segundo.

$$\begin{cases} \text{P - } P_1 = \alpha_1 \cdot S_1 + \alpha_2 \cdot S_2 + \alpha_1 \cdot S_3 \\ \text{P - } P_2 = \beta_1 \cdot R_1 + \beta_2 \cdot R_2 + \beta_1 \cdot R_3 \end{cases} \tag{2.1}$$

El problema que se plantea es el de obtener α_1, α_2, α_3 en función de β_1, β_2, β_3.

Esta cuestión se ha resuelto aplicando una transformación lineal tal como se detalla a continuación, siendo M la matriz de transformación y T la matriz de traslación:

$$P' = M \times P + T \tag{2.2}$$

$$\begin{pmatrix} P'_x \\ P'_y \\ P'_z \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \times \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} + \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix} \tag{2.3}$$

Es importante destacar que el sistema de referencia del espacio virtual siempre va a ser el mismo y que está definido por los vectores S={(1,0,0), (0,1,0), (0,0,1)} siendo su centro el punto P_1=(0,0,0).

Para construir la matriz de cambio de base, se buscan las coordenadas en la base S de los vectores de la base R.

- Coordenadas de R_1=(R_{1x}, R_{1y}, R_{1z}) en la base S:

$$R_1 = (R_{1x}, R_{1y}, R_{1z}) = M_{11} \cdot (1,0,0) + M_{21} \cdot (0,1,0) + M_{31} \cdot (0,0,1) \Rightarrow \begin{cases} M_{11} = R_{1x} \\ M_{21} = R_{1y} \\ M_{31} = R_{1z} \end{cases} \tag{2.4}$$

- Coordenadas de R_2=(R_{2x}, R_{2y}, R_{2z}) en la base S:

$$R_2 = (R_{2x}, R_{2y}, R_{2z}) = M_{12} \cdot (1,0,0) + M_{22} \cdot (0,1,0) + M_{32} \cdot (0,0,1) \Rightarrow \begin{cases} M_{12} = R_{2x} \\ M_{22} = R_{2y} \\ M_{32} = R_{2z} \end{cases} \tag{2.5}$$

- Coordenadas de R_3=(R_{3x}, R_{3y}, R_{3z}) en la base S:

$$R_3 = (R_{3x}, R_{3y}, R_{3z}) = M_{13} \cdot (1,0,0) + M_{23} \cdot (0,1,0) + M_{33} \cdot (0,0,1) \Rightarrow \begin{cases} M_{13} = R_{3x} \\ M_{23} = R_{3y} \\ M_{33} = R_{3z} \end{cases} \tag{2.6}$$

El valor de los vectores que formen la base R dependerá de la orientación que tenga la cámara dentro del lugar de trabajo. La matriz de traslación T, hace referencia al punto en el que se coloque la Kinect en el sistema de referencia del espacio virtual.

Sustituyendo los valores calculados anteriormente se obtiene la matriz de transformación del sistema de referencia de Kinect al sistema de referencia del espacio virtual:

$$\begin{pmatrix} P'_x \\ P'_y \\ P'_z \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \times \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} + \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix} = \begin{pmatrix} R_{1x} & R_{2x} & R_{3x} \\ R_{1y} & R_{2y} & R_{3y} \\ R_{1z} & R_{2z} & R_{3z} \end{pmatrix} \times \begin{pmatrix} P_x \\ P_y \\ P_z \end{pmatrix} + \begin{pmatrix} T_1 \\ T_2 \\ T_3 \end{pmatrix} \tag{2.7}$$

Capítulo 3

Implementación y pruebas

3.1 Introducción

En este tercer capítulo se detallará el proceso llevado a cabo para realizar la implementación del sistema de integración de presencia humana en el espacio virtual.

En primer lugar, se detallarán las funciones de callback utilizadas por `OpenNI` para conseguir detectar a las personas y su posterior seguimiento. También se explicará el proceso de simulación del sistema virtual y el de representación del sistema de profundidad captado por la cámara.

Más tarde se detallará la librería '`skeletonLib.h`' que se ha desarrollado en la que se realiza la integración de presencia humana en el entorno virtual.

Por último, se expondrán una serie de pruebas llevadas a cabo para garantizar el correcto funcionamiento del sistema.

3.2 Detección de presencia humana a través de **`OpenNI`** y **`NITE`**

El primer paso que hay que realizar para la integración de presencia humana en el espacio virtual es el de encontrar esa presencia. Para ello se utiliza la librería `OpenNI` y en especial el middleware `NITE`.

Esta librería trabaja utilizando nodos de producción, que son conjuntos de componentes que tienen un rol productivo en el proceso de creación de datos.

Cada nodo de producción encapsula la funcionalidad que se relaciona con la generación de un tipo específico de datos y puede proveer este dato a la aplicación o a otro nodo de producción. Todo esto se realiza a través de una serie de funciones de callback.

Una función de callback es una función que es llamada tantas veces como se produzca un evento descrito previamente a lo largo de la ejecución del código. Esto permite desarrollar capas de abstracción de código a bajo nivel que pueden ser llamadas desde una subrutina definida en una capa de mayor nivel.

Por ejemplo, uno de los múltiples usos que tienen es para señalización de errores: cuando se detecta que algo esta fallando, se ejecuta una función definida previamente para este tipo de error.

El primer paso que hay que hacer para poder utilizar todas las funciones de `OpenNI` que se comentarán a continuación, es definir un '`context`'. Se trata de un objeto en el que se especifican los parámetros de configuración tales como el campo de visión o la resolución del dispositivo.

Se parte del nodo de producción 'depth generator' en el que se lee periódicamente los datos de profundidad obtenidos y se genera el mapa de profundidad.

Existe un segundo nodo de producción llamado 'user generator' que se encarga de leer periódicamente los datos del mapa de profundidad generados por el primer nodo. Este los analiza y determina si se ha detectado algo que pudiera tener la apariencia de un ser humano.

Una vez que un usuario ha sido detectado correctamente a través de la pose de calibración, se le asigna un número identificativo y se realiza el seguimiento a lo largo de la escena detectada.

En la figura 3.1 se muestra un diagrama del proceso que sigue el programa a lo largo de las llamadas a las funciones de callback de OpenNI.

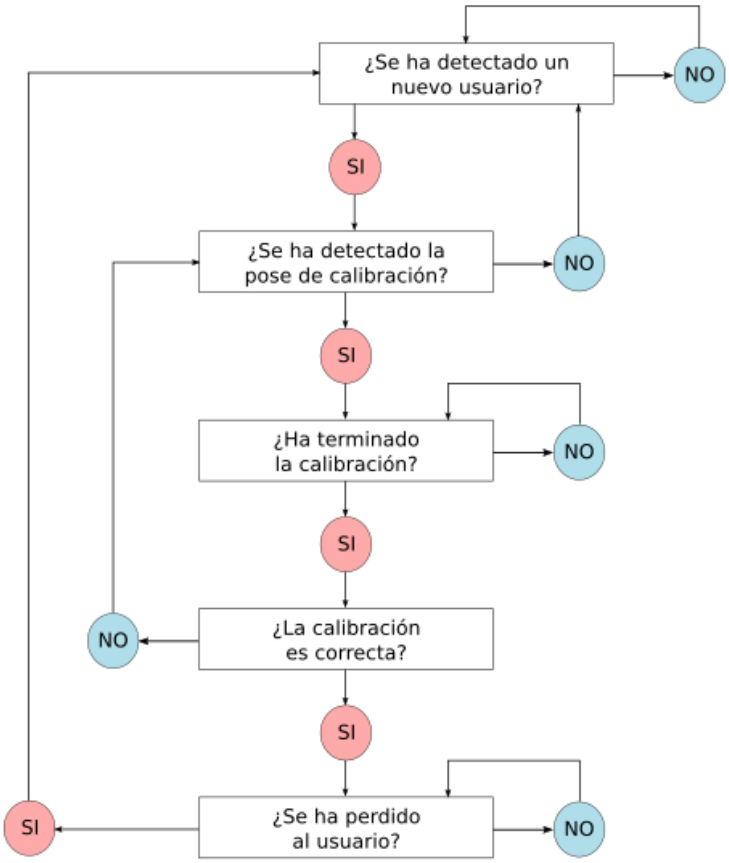


Figura 3.1: Diagrama de flujos de las funciones de callback de OpenNI.

Del diagrama anterior nos hacemos una idea de cómo se van haciendo diferentes llamadas a funciones de callback para realizar la detección del usuario, la calibración y su posterior seguimiento.

1. Se parte de un estado de reposo en el que se continúa a menos que se detecte a un usuario nuevo a través de la función 'User_NewUser'.

2. Una vez detectado el nuevo usuario, se procede a detectar la pose de calibración a través de la función 'UserPose_PoseDetected'. Esta pose consiste en levantar las manos flexionando los codos 90º de forma que los brazos formen una U con los hombros. Si se detecta esta pose, se pasa al siguiente estado. En caso contrario se regresa al estado inicial de reposo.

3. En este tercer estado se evalúa el proceso de la calibración del usuario, permaneciendo en él hasta completar la calibración. Se inicia con la llamada a la función 'UserCalibration_CalibrationStart'.

4. Cuando se termina el proceso de calibrado a través de la llamada a la función 'UserCalibration_CalibrationComplete', se pasa a este cuarto estado en el que se analiza si el proceso anterior se ha realizado de forma correcta. En caso afirmativo, se procede a realizar el seguimiento del usuario a través de la escena. De lo contrario, se regresa al segundo estado en el que se detecta la pose de calibración. De esta forma se posibilita volver a realizar el proceso de calibración si este no se hubiera realizado de forma correcta.

5. En este último estado, se evalúa continuamente si se ha perdido al usuario o se continúa realizando el seguimiento a lo largo de la escena. Si se hubiera perdido al usuario, se ejecutaría la función 'User_LostUser' regresando al estado inicial a la espera de volver a detectar un nuevo usuario. En caso contrario, se continua con el seguimiento del mismo.

En la fase de seguimiento se obtiene la posición de cabeza, cuello, hombros, codos, manos, tórax, caderas, rodillas y pies del usuario referenciadas al sistema de coordenadas de kinect. Es decir, con respecto del dispositivo.

Dentro de las funciones de callback mencionadas anteriormente y de otra que se comentará posteriormente encargada de la parte de visualización, se utilizan una serie de clases incluidas en la librería OpenNI [14] para la obtención de los datos a través de la cámara. Las principales son:

- 'WaitAndUpdateAll': dentro del 'context', esta clase se utiliza para actualizar todos los nodos generadores y se espera hasta que se obtienen nuevos datos.
- 'GetUsers': con esta clase se obtiene el número de usuarios a los que se les está haciendo el seguimiento.
- 'GetPoseDetectionCap': dentro del nodo de producción 'DepthGenerator', se utiliza esta clase para obtener un objeto que sea capaz de acceder a las funcionalidades de la detección de pose.
- 'StartPoseDetection': se utiliza para iniciar la detección de pose del usuario.
- 'GetSkeletonCap': dentro del nodo de producción 'UserGenerator', esta clase obtiene un objeto con la capacidad de acceder a las funcionalidades del esqueleto.
- 'IsTracking': partiendo del objeto obtenido con la clase anterior, este método retorna un booleano indicando si se esta realizando el seguimiento a un usuario específico.
- 'RequestCalibration': es utilizada cuando se necesita realizar la calibración del usuario.
- 'StartTracking': se emplea para inicializar el seguimiento del usuario a lo largo de la escena.
- 'GetSkeletonJointPosition': esta función retorna la posición de las 'joints' detectadas por la cámara. Cabe destacar que estas posiciones estarán referenciadas al sistema de coordenadas de Kienect.
- 'GetMetaData': es utilizada para obtener las medidas de profundidad obtenidas por la cámara.
- 'GetCom': se emplea esta función para obtener el centro de masas de un usuario. Este dato es importante a la hora de insertar los textos informativos referentes a cada usuario dentro de la representación del mapa de profundidad.

- 'ConvertRealWorldToProjective': partiendo del nodo de producción 'DepthGenerator', se emplea esta función para obtener las coordenadas del mundo real a partir de las de la proyección.

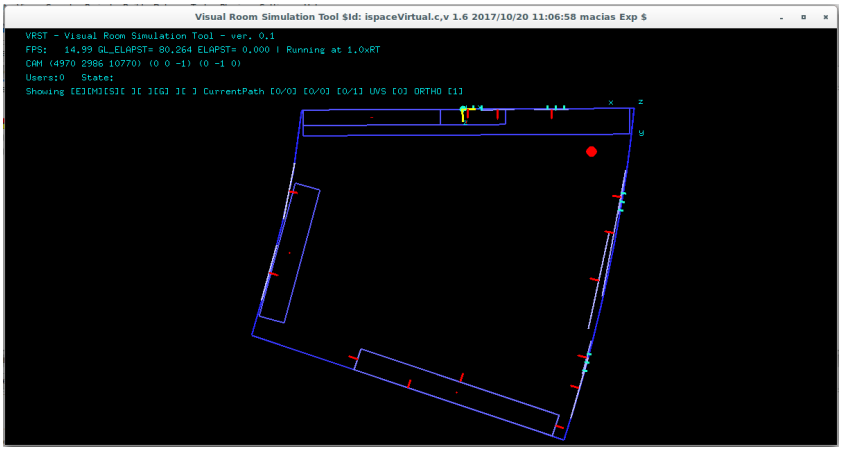
Cabe destacar que OpenNI permite realizar un seguimiento de hasta 6 personas simultaneamente.

Tal como se ha comentado anteriormente, será necesario realizar una serie de transformaciones para conseguir mapear las coordenadas de las 'joints' dentro del entorno simulado con respecto al sistema de referencias que tiene éste y no con respecto al de Kinect. De no ser así, se verá el 'esqueleto' formado por las 'joints' en un sitio y orientación aleatoria.

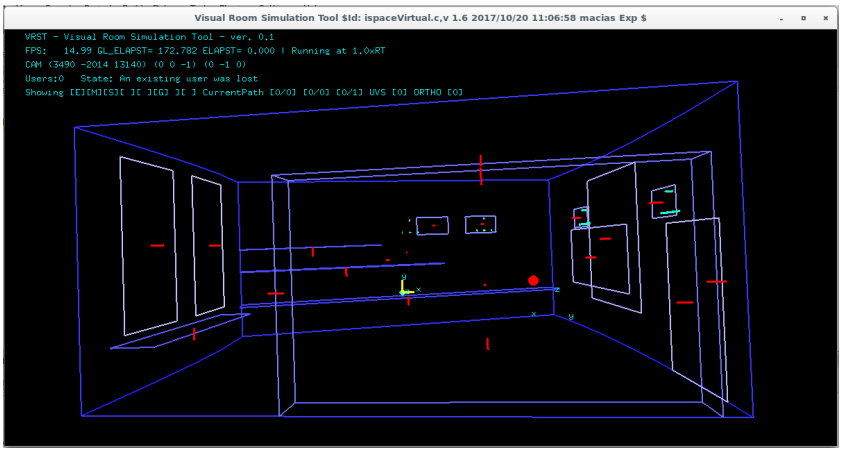
3.3 Simulación del espacio virtual

Uno de los elementos fundamentales de este trabajo es el entorno virtual en el que se tiene que integrar la presencia humana detectada por la Kinect. Como se ha mencionado anteriormente, el código que simula espacio virtual a través de la librería OpenGL ha sido cedido por el tutor Javier Macías Guarasa.

Esta simulación trata de emular el 'espacio geintra', es decir, la sala física número 201 de la zona Oeste del Edificio Politécnico de la Universidad de Alcalá de Henares.



(a) Espacio virtual en 2D.



(b) Espacio virtual en 3D.

Figura 3.2: Simulación del espacio de trabajo en dos y tres dimensiones respectivamente.

Como se puede ver en la imagen 3.2, se trata de una habitación con una forma trapezoidal en la que se han incluído una serie de detalles como una puerta, dos ventanas, dos mesas laterales, dos baldas colgadas de la pared y un armario.

El código de simulación posibilita la simulación del espacio virtual tanto en dos dimensiones como en tres. Permite cambios de vistas y proyecciones para mejorar la visualización en diferentes situaciones, así como acercarse o alejarse para poder observar con más detalle la simulación.

Dentro de la ventana de simulación, se puede ver tanto la habitación virtual como una serie de mensajes informativos sobre el estado de la simulación en la esquina superior derecha.

Con el fin de obtener más detalles acerca del estado de la integración de personas humanas dentro de este espacio simulado, se ha añadido una línea más con dos campos informativos:

- Users: aquí vendrá reflejado el número actual de usuarios detectados a los que se les está haciendo el seguimiento a lo largo de la escena. De tal modo que cuando aparezca un nuevo usuario en la escena y se realice la calibración de forma correcta, este contador aumentará y cuando se pierda a un usuario descenderá.
- State: este campo estará relleno con la información relevante al último usuario detectado. Se podrán distinguir los siguientes mensajes:
  - *'New user was detected'*: aparecerá en el momento en que se detecte la presencia de un nuevo usuario.
  - *'An existing user was lost'*: este mensaje indicará que un usuario al que se le estaba haciendo el seguimiento ha desaparecido del campo de detección de la cámara.
  - *'Detected a pose'*: indicando que un usuario nuevo detectado está poniendo la pose de calibración.
  - *'Started calibration'*: mensaje que indica que se ha iniciado la calibración de un nuevo usuario.
  - *'Calibration succeeded'*: se mostrará cuando termine satisfactoriamente la calibración del usuario.
  - *'Calibration failed'*: si en el momento de la calibración algo falla aparecerá este mensaje informativo en la pantalla.

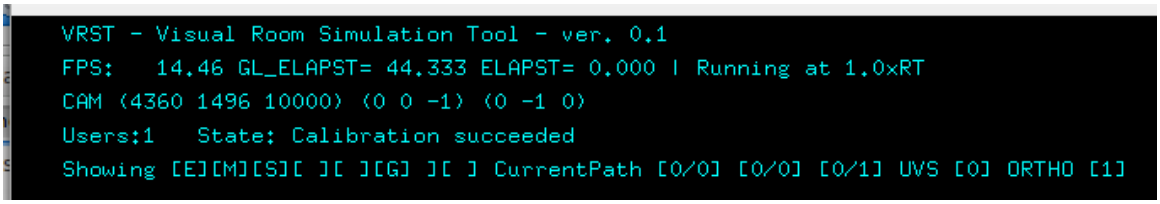


Figura 3.3: Ejemplo de mensajes informativos mostrados en la ventana del espacio simulado.

En la imagen 3.3 se puede ver un ejemplo de los mensajes informativos que aparecen en la ventana de simulación del espacio virtual. En este caso, se ha realizado correctamente la calibración a un usuario por lo que se estará llevando a cabo su seguimiento a lo largo de la escena.

Otro de los elementos relevantes que se representan dentro de la habitación simulada es la propia cámara.

Esta representación simbólica se hace mediante un círculo de color azul claro junto con los ejes del sistema de referencias de Kinect. De esta forma, se puede apreciar tanto la colocación como la orientación del dispositivo dentro de la escena.

Finalmente, el elemento principal en ésta integración de presencia humana es la propia presencia humana.

Para realizar esto, se representa un 'esqueleto' formado por los puntos de las 'joints' obtenidos por Kinect. De esta forma se puede obtener la posición exacta del usuario dentro del espacio virtual, ya que estos puntos se van actualizando.

El 'esqueleto' está formado por la unión entre las siguientes articulaciones:

- Cabeza.
- Cuello.
- Hombros.
- Codos.
- Manos.
- Tórax.
- Caderas.
- Rodillas.
- Pies.

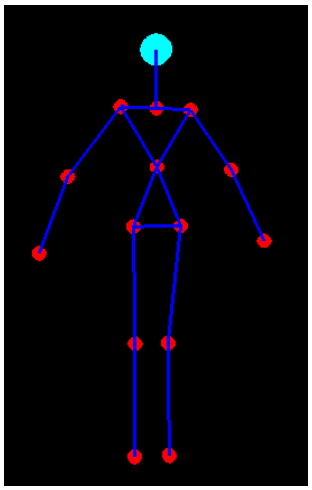


Figura 3.4: 'Esqueleto' formado por los joints detectados.

Tal como se puede ver en la imagen 3.4 las articulaciones están representadas mediante puntos rojos, la cabeza con un punto de mayor tamaño azul claro y las uniones mediante líneas de color azul oscuro.

Para realizar el proceso de simulación dentro del código, hay definida una función de callback llamada *'renderScene'* que se ocupa de todo el proceseso de rendering. Esta función es llamada continuamente y sus principales tareas son:

- Representación del entorno de trabajo simulado, incluyendo los objetos que se pueden añadir tal como se mostrará en el manual de usuario.
- Visualización de los mensajes informativos.
- Representación de la posición de los 'joints' que forman el usuario virtual.
- Representar la cámara Kinect junto con su sistema de referencias.

3.4 Representación del mapa de profundidad

La librería `OpenNI` además de dar información sobre la posición de las 'joints', también crea un mapa de profundidad a partir de las medidas obtenidas con Kinect.

Al ejecutar la aplicación, se ha decidido generar una segunda ventana en la que se representa dicho mapa de profundidad. Esto permite comprobar de un vistazo que la integración de presencia humana dentro del espacio virtual se está realizando correctamente, es decir, que lo que se está representando dentro de la habitación simulada corresponde con lo que está sucediendo en la realidad.

La representación del mapa de profundidad se ha hecho empleando la librería `OpenCV`. Para ello, se genera un objeto `Mat` con las mismas dimensiones como muestras de profundidad captura Kinect. De esta forma, cada píxel corresponderá con una muestra obtenida.

Con el fin de aportar mayor información, una vez detectado un usuario se calcula su centro de masa y se representa un mensaje en dicha posición. Esto permite que la cadena de caracteres acompañe al usuario a lo largo de la escena en caso de que este se desplazase. Los mensajes que se representan son:

- *'N - Calibrating'*, siendo N el número asociado al usuario en cuestión. Este mensaje aparece en el momento en que se detecta un nuevo usuario y se procede a realizar la calibración para su posterior reguimiento.
- *'N - Tracking'*, siendo N el número asociado al usuario en cuestión. Indicando que se ha efectuado la calibración correctamente y que se está realizando el seguimiento del usuario a través de la escena con su correspondiente integración en el espacio virtual.

Figura 3.5: Ejemplo de representación de mapa de profundidad en el que se está realizando el seguimiento a 2 usuarios.

En la figura 3.5 se puede ver la representación del mapa de profundidad en el que se está realizando el seguimiento a dos usuarios.

3.5 Librería `skeletonLib.h`

El proceso de integración de presencia humana dentro del entorno virtual se realiza empleando las librerías mencionadas anteriormente. Para que quede todo más agrupado y ordenado, se ha procedido a desarrollar

la librería 'skeletonLib.h' en la que se han codificado una serie de funciones empleando 'OpenNI' y 'OpenCV'.

Dentro de 'skeletonLib.h' se pueden encontrar las siguientes funciones:

- __initSkeletonLib: esta función se encarga de configurar la cámara para su correcto funcionamiento. Es aquí donde se define el 'context' , se inicializan los nodos de producción y se habilitan las funciones de callback.

  Se trata de una función que no necesita ningún parámetros y que retorna un código identificativo en caso de error.

- __convertSkeletonJoints: aquí se realiza el proceso de conversión de coordenadas entre el sistema de referencia de Kinect y el del espacio virtual.

  En este caso, tampoco hay que pasarle ningún parámetros a la función para que funcione correctamente ya que se emplean unas variables globales auxiliares.

- __getSkeletonJoints: en esta función se obtienen las posiciones de las articulaciones de los usuarios a partir de la función de OpenNI 'GetSkeletonJointPosition'.

  En este caso, si que es necesario pasarle un puntero a una estructura con todas las posiciones de los joints del esqueleto y el número identificativo del usuario al que se le está haciendo el seguimiento.

  Dentro de esta función existe un flag que posibilita elegir si realizar el cambio de sistema de referencia de la posición de las joints o no.

- __DrawDepthMap: esta función es capaz de representar el mapa de profundidad a partir de las medidas obtenidos por la cámara empleando la librería OpenCV.

Aparte, también se han integrado las diferentes funciones de callback de OpenNI detalladas anteriormente: 'User_NewUser', 'User_LostUser', 'UserPose_PoseDetected', 'UserCalibration_CalibrationStart', 'UserCalibration_CalibrationComplete'.

3.6 Pruebas de funcionamiento

En esta última sección del tercer capítulo, se detallarán las pruebas que se han realizado y se mostrará el funcionamiento del trabajo.

Con el fin de realizar estas pruebas, se ha colocado la cámara en tres posiciones diferentes dentro del espacio de trabajo. De esta forma, se puede comprobar que la aplicación es reutilizable y que se realiza correctamente el mapeo de las posiciones de los usuarios dentro del espacio virtual.

La sección se dividirá en tres partes, una por cada punto en el que se ha colocado el dispositivo Kinect:

- La cámara se ha posicionado en el eje X del sistema de referencias del espacio virtual, en el punto (4000, 0, 800).

  En la imagen B.1 se puede ver en la esquina superior derecha el sistema de referencias del espacio virtual. El dispositivo se ha colocado en el eje X cuyo sistema de referencias con respecto al de la habitación simulada es {(-1, 0, 0), (0, 0, 1), (0, 1, 0)}.

  Al ejecutar la simulación con los parámetros mencionados anteriormente, el resultado se observa en la figura 3.7.

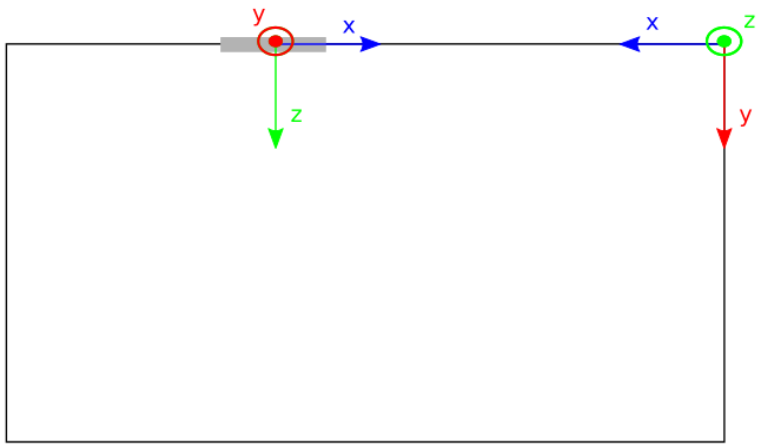


Figura 3.6: Esquema de los sistemas de referencias de la primera prueba realizada.

- En este caso, se ha decidido colocar el dispositivo en el eje Y del sistema de referencias de la habitación simulada en el punto (500, 3000, 800).

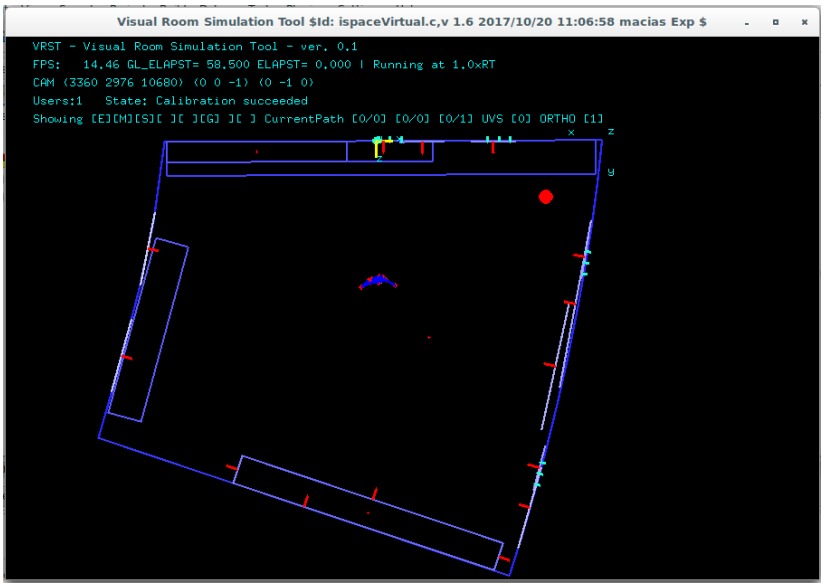
  Tal como se puede observar en el esquema 3.8, la cámara se ha colocado en el eje Y de la habitación, siendo su sistema de referencias con respecto al del simulado {(0, 1, 0), (0, 0, 1), (1, 0, 0)}.

  En la imagen 3.9 se puede ver el resultado obtenido durante la ejecución con los parámetros de posición y orientación de la cámara previamente citados.

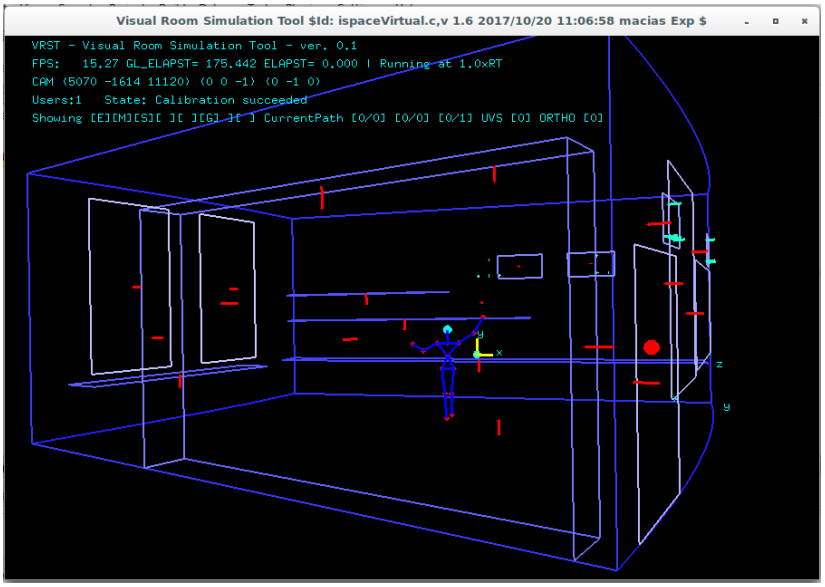
- En esta última prueba, se ha fijado la Kinect en una esquina de forma oblicua de tal manera que pueda captar todo lo que sucede dentro de la escena. Se ha colocado la cámara de forma que el eje longitudinal de ésta forme una ángulo de 30º con respecto al eje X del sistema de referencias del espacio virtual. En la figura 3.10 se puede apreciar de forma más clara la colocación del dispositivo dentro de la escena virtual.

  Se puede ver en este caso que al colocarse la cámara en una posición oblicua, será necesario aplicar razones trigonométricas para obtener su sistema de referencias. Quedando este de la siguiete forma: {(-cos (30º), -sen(30º) ,0), (0, 0, 1), (-cos(60º), sen(60º), 0)}.

  El resultado de la ejecución de la simulación se puede ver en la figura 3.11.



(a) Prueba con el espacio virtual en 2D.



(b) Prueba con el espacio virtual en 3D.

Figura 3.7: Prueba número uno de ejecución de la aplicación desarrollada.

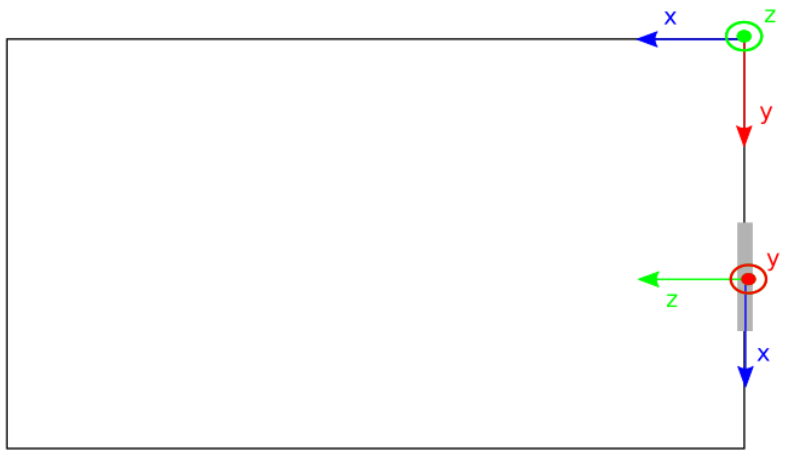
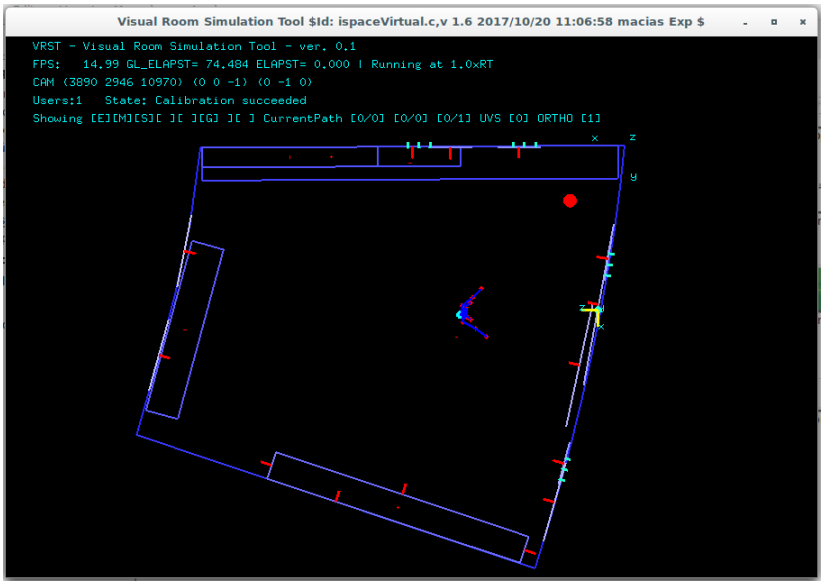
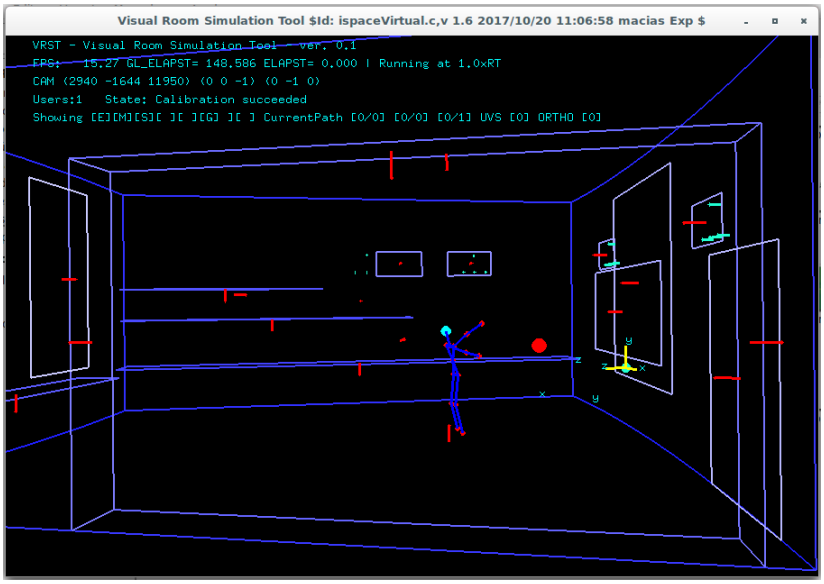


Figura 3.8: Esquema de los sistemas de referencias de la segunda prueba realizada.



(a) Prueba con el espacio virtual en 2D.



(b) Prueba con el espacio virtual en 3D.

Figura 3.9: Segunda prueba de ejecución del trabajo desarrollado.

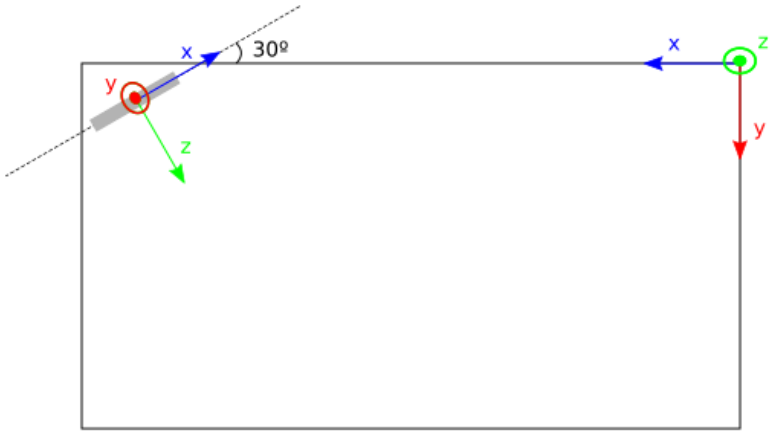
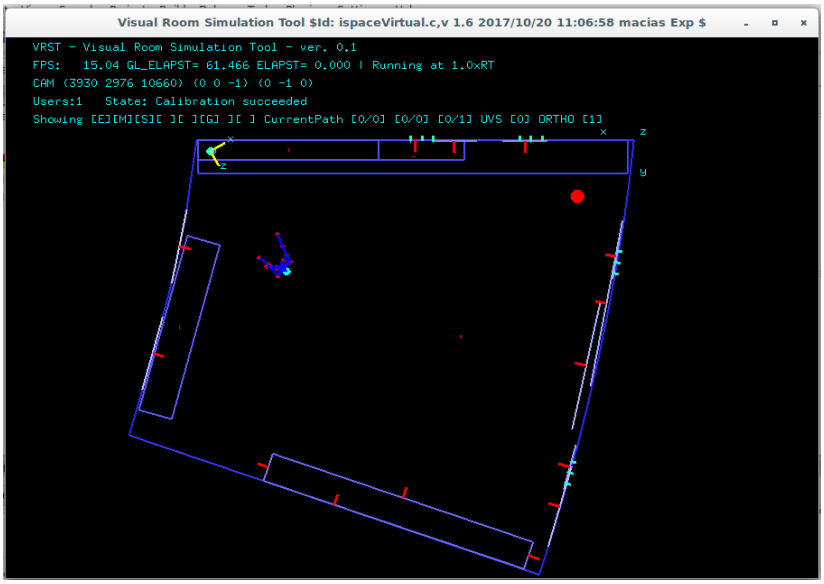
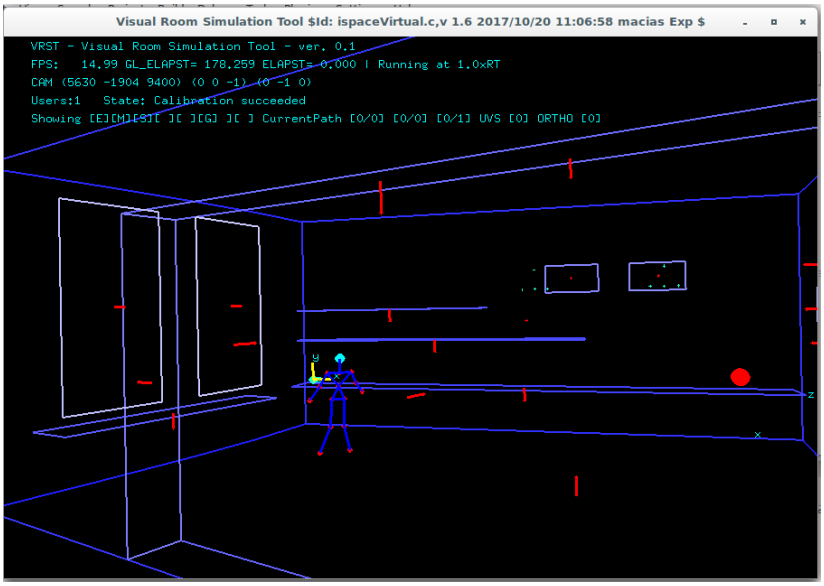


Figura 3.10: Esquema de los sistemas de referencias de la tercera prueba realizada.



(a) Prueba con el espacio virtual en 2D.



(b) Prueba con el espacio virtual en 3D.

Figura 3.11: Tercera prueba del desarrollo de la implementación de la integración de presencia humana dentro del espacio virtual.

Capítulo 4

Conclusiones y líneas futuras

4.1 Introducción

En este último capítulo de la memoria se indicarán las conclusiones obtenidas a partir de la realización del trabajo, así como futuras líneas de trabajo que puedan desarrollarse a partir de lo expuesto anteriormente.

4.2 Conclusiones

El objetivo principal de este desarrollo se ha centrado en la integración de presencia humana dentro de una simulación que emula un espacio de trabajo existente en la realidad. Parte de una simulación ya existente en la que se ha integrado la presencia de usuarios.

En primer lugar, se ha analizado el código de simulación cedido para comprender la estructura del mismo y poder decidir las funciones que pudieran ser de utilidad para el objetivo de este trabajo.

Más tarde, se ha realizado una investigación sobre las cámaras de profundidad y sus diferentes funcionamientos.

Esto implica la elección de la cámara Kinect por ser un dispositivo muy versátil, de fácil acceso y asequible económicamente.

Otro de los factores importantes a la hora de la elección de la cámara Kinect ha sido el respaldo de la comunidad de desarrolladores e investigadores que existen en Internet y aportan sus conocimientos desinteresadamente a través de la red.

Otro de los aspectos importantes en la realización del trabajo ha sido la elección de las librerías necesarias a la hora de desarrollar el código.

- Para la simulación del entorno de trabajo se utiliza `OpenGL` por herencia. Es decir, al ser un código cedido no se ha tocado nada con respecto a este punto.
- La detección de presencia humana se realiza mediante `OpenNI` por contar con el middleware `NITE`, el cual se adapta perfectamente a las necesidades demandadas por el trabajo.
- Por último la representación del mapa de profundidad se representa empleando la librería `OpenCV` por la imposibilidad de hacerlo correctamente mediante `OpenGL`.

A través de esta librería es mucho más sencillo que como se planteó en un primer momento mediante `OpenGL`, ya que no es necesario estar definiendo ventanas nuevas ni texturas adicionales. Esto también favorece una simulación más fluida y con menos carga a la parte gráfica del ordenador.

Una vez realizado el desarrollo de la aplicación, se han realizado las pruebas mencionadas anteriormente.

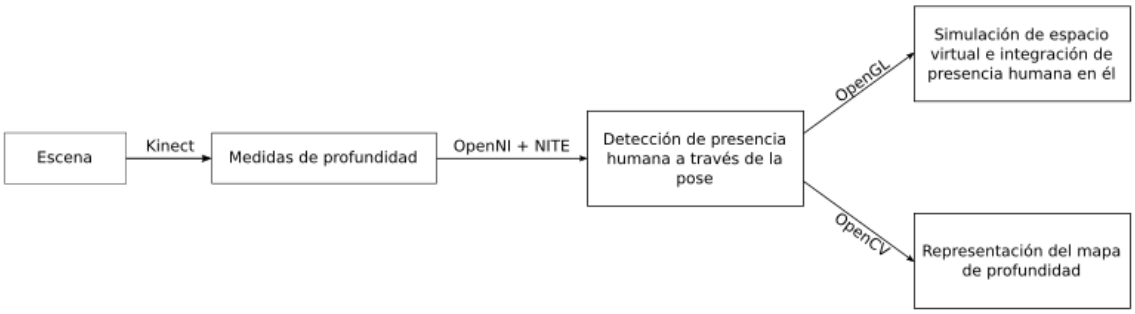


Figura 4.1: Diagrama de flujo del trabajo.

En la figura 4.1 se puede ver el diagrama de frujo que presenta el trabajo desde que se obtienen las muestras con Kinect, hasta que se representan el mapa de profundidad y la simulación del espacio virtual en dos ventanas independientes.

A partir de los resultados obtenidos, se puede afirmar que la integración de presencia humana dentro del espacio virtual se realiza de forma correcta tal como se pretendía al inicio del trabajo. Se ha aprovechado el potencial de Kinect haciendo la integración multiusuario, permitiendo realizar un seguimiento de hasta 6 usuarios simultáneamente.

4.3 Líneas futuras

A continuación se citarán algunas de las posibles mejoras y líneas de trabajo que podrían desarrollarse sobre lo expuesto anteriormente.

- Utilizar la segunda generación de Kinect: este dispositivo calcula la profundidad a través del tiempo de vuelo. Esto hace que sean más precisas las muestras y mejora las limitaciones expuestas anteriormente.
- Mejorar la detección de pérdida de un usuario: actualmente desde que desaparece un usuario de la escena hasta que realmente se detecta su pérdida transcurre un pequeño periodo de tiempo. Esto se debe a que `OpenNI` guarda la calibración de un usuario durante un tiempo por si éste vuelve al campo de visión de la cámara. Durante este tiempo, en el espacio virtual se representa al usuario aunque realmente no está.
- Más detalles en mapa de profundidad: otra cosa que se podría mejorar es el mapa de profundidad representado. Podrían incluirse más detalles como por ejemplo superponer el 'esqueleto' detectado por Kinect encima del correspondiente usuario.
- Añadir interacción virtual del usuario con elementos del entorno.

Bibliografía

[1] R. G. Jiménez, "Detección y conteo de personas, a partir de mapas de profundidad cenitales capturados con cámaras tof," Master's thesis, Escuela Politécnica Superior. Universidad de Alcalá. Spain, 2015.

[2] "Entrada de kinect en wikipedia." [Online]. Available: https://es.wikipedia.org/wiki/Kinect

[3] "Sitio microsoft donde se puede descargar el sdk de kinect para windows." [Online]. Available: https://www.microsoft.com/en-us/download/details.aspx?id=44561

[4] "Página oficial de la comunidad openkinect." [Online]. Available: https://openkinect.org/wiki/Main_Page

[5] "Página de la organización openni." [Online]. Available: http://openni.ru/

[6] "Entrada de openni en wikipedia." [Online]. Available: https://es.wikipedia.org/wiki/OpenNI

[7] PrimeSense, "Prime sensor nite 1.3 controls programmer's guide," 2010.

[8] "Entrada de opengl en wikipedia." [Online]. Available: https://es.wikipedia.org/wiki/OpenGL

[9] S. M. F. Jose Luis Bosque Orero, Ed., *Introducción a OpenGL*. S.L. Dykinson, 2007.

[10] J. A. P. G. Rafael Molina Carmona, "Apuntes de opengl."

[11] G. A. V. M. Arévalo, J. González, "La librería de visión artificial opencv aplicación a la docencia e investigación."

[12] "Página donde se explica el cambio entre sistemas de referencias." [Online]. Available: https://personal.us.es/jcortes/Material/Material_archivos/Articulos%20PDF/SistemRefer.pdf

[13] "Ejemplos del cambio entre sistemas de referencia." [Online]. Available: https://aga.frba.utn.edu.ar/matriz-de-cambio-de-base/

[14] "Página donde vienen detalladas las distintas clases y funciones de openni." [Online]. Available: http://kinectcar.ronsper.com/docs/openni/index.html

[15] "Página donde se explica el proceso de instalación de openni, primesense y nite." [Online]. Available: https://www.icyphy.org/accessors/wiki/ROS/InstallingThePrimeSenseKinectSensorOnUbuntu?from=Main.InstallingThePrimeSenseKinectSensorOnUbuntu

[16] "Tutorial de latex." [Online]. Available: https://www.fing.edu.uy/~canale/latex.pdf

Apéndice A

Proceso de instalación

En este apéndice se expondrá un manual detallado en el que se explicará el proceso de instalación que hay que realizar para poder utilizar la cámara Kinect y las diferentes librerías necesarias para que el dispositivo funcione de forma correcta [15].

La instalación se ha realizado introduciendo los siguientes comandos en una terminal siguiendo el siguiente orden:

1. **Instalación de librerías**:

   Para que la instalación se realice de forma correcta y que todo funcione bien, primero hay que instalar una serie de librerías y de herramientas que permitan agilizar este proceso.

   ```
   sudo apt-get install freeglut3-dev pkg-config build-essential
   libxmu-dev libxi-dev libusb-1.0-0-dev doxygen graphviz mono-complete
   ```

2. **Instalación de `OpenNI`:**

   El paso anterior es necesario, pero ahora a partir de aquí es cuando realmente empieza el proceso de instalación de los drivers de Kinect.

   Se recomienda crear una carpeta dónde tener todo el código de instalación y los diferentes drivers.

   En primer lugar, es necesario descargar el código de `OpenNI` de un repositorio github.

   ```
   git clone https://github.com/OpenNI/OpenNI.git
   ```

   Una vez descargado el código, accedemos a la siguiente ruta y compilamos el código generando un script.

   ```
   cd Openni/Platform/Linux/CreateRedist
   ```

   ```
   ./RedistMaker
   ```

   Cuando el proceso anterior haya finalizado, se habrá creado una carpeta llamada 'Redist' y dentro de ésta otra llamada '`OpenNI-Bin-Dev-Linux-x64-XXXX`'. `XXXX` hace referencia a la versión del código descargado.

   Accedemos a la carpeta mencionada y ejecutamos el script generado.

   ```
   sudo ./install.sh
   ```

3. **Instalación del driver `PrimeSense`:**

   La instalación de este driver, seguirá la misma metodología que para el caso de `OpenNI`.

Primeramente se descargará el código de un repositorio github en la carpeta de trabajo, se compilará el código generando un script y por último se ejecutará dicho script.

```
git clone https://github.com/ph4m/SensorKinect.git
cd SensorKinect/Platform/Linux/CreateRedist
./RedistMaker
cd  /kinect/SensorKinect/Platform/Linux/Redist/Sensor-Bin-Linux-x64-XXXX/
sudo ./install.sh
```

4. **Instalación del middleware NITE:**

   Para poder realizar la detección de presencia humana y tomar las medidas de profundidad del cuerpo, es fundamental tener instalado el middleware NITE.

   En este caso el proceso de instalación es sencillo, no es necesario clonar ningún repositorio, basta con descargarse el código directamente desde internet dentro de la carpeta de trabajo.

   Desde la página de OpenNI puede descargarse el código, habiendo un histórico con diferentes versiones.

   ```
   http://openni.ru/openni-sdk/openni-sdk-history-2/index.html
   ```

   Una vez descargado y descomprimido el software, se accede a la carpeta generada y se procede a la instalación del mismo.

   ```
   ./install.sh
   ```

   Con todo esto, ya debería ser suficiente para poner en marcha la cámara y poder obtener datos.

   Para comprobar si el proceso de instalación se ha realizado con éxito, basta con ejecutar alguno de los demos ubicados en la ruta Openni/Platform/Linux/Bin/x64-Release.

5. **Instalación de la librería Opencv:**

   La librería OpenCV es la más sencilla de instalar, pues se puede hacer a través del gestor de paquetes 'aptitude'.

   Basta con introducir el siguiente comando por el terminal:

   ```
   aptitude install libopencv-dev
   ```

Apéndice B

Manual de usuario

En este último apéndice se detallará un manual de usuario dividido en dos partes, por un lado se citarán los modos de funcionamiento del espacio virtual y sus comandos, y por otro lado se explicará al usuario como configurar la posición y la orientación de la cámara dentro del entorno simulado.

B.1 Manual del espacio virtual

Dentro del espacio virtual, existen dos modos de funcionamiento distintos: modo normal y modo especial.

Por defecto, la simulación arranca en el modo normal y para cambiar al especial hay que presionar la tecla '8'.

Cuando la simulación esté dentro del modo especial, aparecerá en la pantalla el siguiente mensaje: *'Teclado especial activado, para volver pulse la tecla '8'.*

Dentro del modo normal se pueden realizar casi todas las funcionalidades del programa excepto las relacionadas con la iteracción de objetos, siendo ésta la única diferencia entre los modos.

A continuación se detallan los comandos permitidos en cada modo de funcionamiento:

- **Modo normal:**
  - Intercambiar entre la simulación en 2D y 3D: '`Mayús + o`'
  - Rotar sobre el eje X: '`Mayús + x`'
  - Activar/Desactivar rotación continua en el eje X: '`Alt + x`'
  - Rotar sobre el eje Y: '`Mayús + y`'
  - Activar/Desactivar rotación continua en el eje Y: '`Alt + y`'
  - Rotar sobre el eje Z: '`Mayús + z`'
  - Activar/Desactivar rotación continua en el eje Z: '`Alt + z`'
  - Activar/Desactivar objetos: '`6`'
  - Activar/Desactivar relleno de objetos: '`7`'
  - Alejar la cámara: '`b`'
  - Acercar la cámara: '`Mayús + b`'
  - Invertir la imagen actual: '`Mayús + v`'

  - Volver a la vista inicial: 'Mayús + c'
  - Ocultar/Mostrar los arrays de micrófonos: 'Mayús + m'
  - Ocultar/Mostrar la estructura del espacio virtual: 'Mayús + e'
  - Desplazar 10 centímetros al robot en sentido positivo del eje X: '2'
  - Desplazar 10 centímetros al robot en sentido negativo del eje X: '3'
  - Desplazar 10 centímetros al robot en sentido positivo del eje Y: '4'
  - Desplazar 10 centímetros al robot en sentido negativo del eje Y: '5'
  - Iniciar movimiento de la pelota: 'Espacio'
  - Cambiar al modo especial: '8'

- **Modo especial:**
  - Activar/Desactivar relleno de objetos: '7'
  - Abrir/Cerrar la puerta: 'p'
  - Abrir/Cerrar la ventana: 'v'
  - Subir/Bajar las persianas: 'b'
  - Encender/Apagar la luz del techo: 'l'
  - Encender/Apagar la televisión: 't'
  - Encender/Apagar la radio: 'r'
  - Encender/Apagar la lámpara 1: '1'
  - Encender/Apagar la lámpara 2: '2'
  - Girar el robot en sentido antihorario: 'o'
  - Girar el robot en sentido horario: 'Mayús + o'
  - Cambiar al modo normal: '8'

B.2 Manual de funcionamiento

Antes de poner en marcha la aplicación es importante definir dónde va a estar colocada la cámara y cuál va a ser su orientación.

Una vez decidido esto, se procede a realizar la configuración de estos parámetros dentro de la aplicación siguiendo los siguientes pasos:

1. Lo primero que hay que tener en cuenta es la colocación del dispositivo.

   Tomando como ejemplo el primer caso que se ha expuesto en la sección de pruebas de funcionamiento, se ha decidido colocar la cámara de forma horizontal apoyando toda la superficie de la base sobre una mesa. El sensor apunta en dirección perpendicular a la mesa, es decir, se encuentra con una inclinación de 0º.

   Se ha colocado la Kinect a lo largo del eje X del sistema de referencia del espacio virtual, haciendo coincidir el eje Z de la cámara con el eje Y del espacio simulado.

   En la imagen B.1 de puede ver dicha colocación de una forma más clara. El sistema de referencia de la esquina superior derecha es el referente al espacio virtual, y el de la izquierda el de la cámara Kinect.

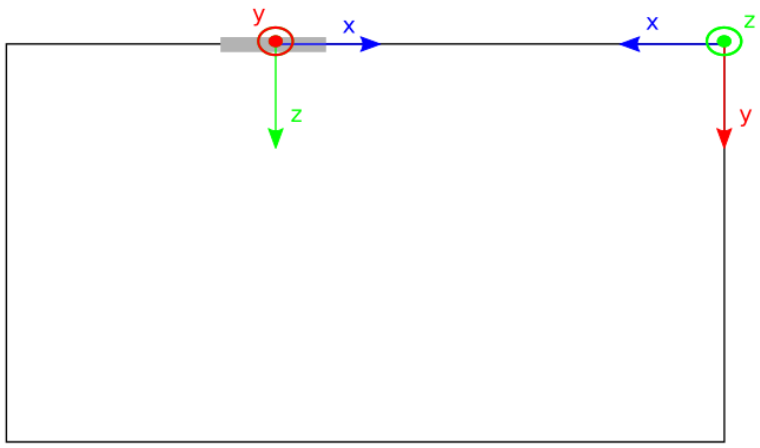


Figura B.1: Esquema de los sistemas de referencias.

2. Una vez colocado el dispositivo, es importante identificar correctamente los vectores asociados a la cámara con respecto al sistema de referencia del espacio virtual.

   En este caso, los vectores quedan:

$$x = (-1, 0, 0) \tag{B.1}$$

$$y = (0, 0, 1) \tag{B.2}$$

$$z = (0, 1, 0) \tag{B.3}$$

3. Una vez identificados los vectores de la cámara hay que introducirlos en la aplicación.

   Este proceso se hace en la parte inicial del fichero `ispaceVirtual.c` en la que se han definido tres arrays de tres posiciones cada uno (u1, u2 y u3). Siendo cada uno un vector con sus correspondientes ejes.

   Para introducir la posición del dispositivo dentro del espacio virtual se han creado tres variables (`pos_kinect_x`, `pos_kinect_y` y `pos_kinect_z`) una por cada coordenada. En este caso,se ha decidido colocar la cámara en el punto (4000, 0, 800) del entorno simulado.

   En la imagen B.2 se puede ver la declaración de las variables y como ha sido configurado para el ejemplo mencionado

```
int pos_kinect_x=4000;
int pos_kinect_y=0;
int pos_kinect_z=800;
//--------- VECTORES QUE FORMAN EL SISTEMA DE REFENCIAS DE KINECT ---------
// los angulos, si los hubiera, hay que indicarlos en radianes!!!
float u1[]={-1,0,0};
float u2[]={0,0,1};
float u3[]={0,1,0};
//-------------------------------------------------------------------------
```

Figura B.2: Conjunto de variables definidas en el código para definir la posición y la orientación de Kinect dentro del espacio virtual.

Apéndice C

Herramientas y recursos

Las herramientas necesarias para la elaboración del proyecto han sido:

- PC compatible
- Sistema operativo GNU/Linux
- Entorno de desarrollo Emacs
- Entorno de desarrollo Code::Blocks IDE
- Procesador de textos LaTeX[16]
- Editor de textos Texmaker
- Control de versiones CVS
- Compilador C/C++ gcc
- Gestor de compilaciones make

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá