ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA Y SISTEMAS DE TELECOMUNICACIÓN

PROYECTO FIN DE GRADO

TÍTULO: Reconocimiento de movimientos faciales mediante Kinect v2

AUTOR: Daniel Sánchez-Rico Carrero

TUTOR: Martina Eckert

TITULACIÓN: Grado en Ingeniería de Sonido e Imagen

DEPARTAMENTO: Departamento de Teoría de la Señal y Comunicaciones

VºBº

Miembros del Tribunal Calificador:

PRESIDENTE: Mª Luisa Martin Ruiz

VOCAL: Martina Eckert

SECRETARIO: Enrique Rendón Angulo

Fecha de lectura:22 de julio de 2016

Calificación:

El Secretario,

Agradecimientos

Agradezco a todas las personas que me han ayudado a hacer posible el desarrollo de este Proyecto Fin de Grado, especialmente a la tutora del mismo, Martina Eckert, por haberme dado la oportunidad de trabajar con ella.

Igualmente estoy muy agradecido a los amigos y compañeros que he conocido a lo largo de estos cinco años de Grado Universitario, por los grandes momentos que hemos compartido y el apoyo que he recibido por su parte cuando las cosas no iban bien.

Por último, agradecer a mis padres el haber apostado siempre por mi educación por encima de todo. Estando su cariño y apoyo moral siempre presentes a lo largo de todos mis años de formación académica.

Resumen

Hoy en día no se puede concebir la vida cotidiana sin Tecnologías de la Información y Comunicación (TIC). Estas tecnologías ofrecen mecanismos que contribuyen a la mejora de la calidad de vida de todos, teniendo especial relevancia en determinados grupos de la población, como puede ser el caso de los discapacitados a nivel físico. La supervisión de personal cualificado no siempre está a la disposición de todos los pacientes, por lo que el desarrollo de un sistema que guíe sus ejercicios de rehabilitación contribuirá al bien común.

En este Proyecto Fin de Grado se implementa una herramienta modular y ampliable que permite incorporar la cámara Kinect para Xbox One de reconocimiento de movimientos corporales y faciales, a juegos desarrollados en Blender. Con el objetivo futuro de su incorporación a terapias de rehabilitación del rostro. Tras realizar un estudio sobre el estado actual de investigación en el reconocimiento facial y analizar la amplia gama de posibilidades que ofrece el sensor, se desarrolla la base de comunicación entre el mismo y Blender. Finalmente se idean varios videojuegos básicos en los que a través de la detección de determinadas expresiones faciales se consigue el movimiento de diversos objetos en la escena virtual.

Después de una evaluación de estos métodos en un grupo de niños, se extraen conclusiones que se deducen del trabajo realizado, proponiendo una serie de mejoras necesarias de cara a una investigación futura.

Abstract

Nowadays, it seems that daily life cannot be understood without the Information and Communication Technologies (ICT). These technologies contribute to improve our quality of life, focusing on particular population groups, as it may be the case of people with physical disabilities. The supervision by qualified people is not always available for all the patients, so the development of a system which may lead and supervise their rehab exercises will benefit society.

This Bachelor's Thesis aims to design a modular and expandable tool which incorporates the Kinect camera for Xbox One to those videogames created by Blender in order to recognize body and face movements. After carrying out a study about the state of art on face recognition, besides analyzing the broad range of possibilities that the sensor offers, it has been developed a communications bridge between Blender and the sensor. Finally some basic videogames in which it is possible to move several objects thanks to the detection of different face expressions are developed.

Through the physical evaluation of these methods children, some conclusions have been obtained, which have led to a list of improvements for future researches.

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

La ciencia y la tecnología siempre han evolucionado de forma conjunta al hombre. El desarrollo de una conlleva o potencia el desarrollo de la otra, es por ello que siempre las hallamos conectadas. La motivación de una innovadora forma de comunicación entre ellos ha propiciado la aparición de la Interfaz Natural de Usuario, un tipo de interfaz en la que se interactúa con el sistema o aplicación sin utilizar mandos de control o dispositivos de entrada.

Una de las actividades humanas que está más en consonancia con las Tecnologías de la Información y la Comunicación es la Medicina. Cada vez es mayor el número de usuarios e instituciones que se han incorporado a la búsqueda de diferentes medios que permitan mejoras en el tratamiento y calidad de vida de los pacientes.

Esta es la línea en la que trabaja este Proyecto Fin de Grado, se pretende desarrollar una aplicación de reconocimiento de movimientos faciales para su integración en un videojuego que haga las veces de terapia de rehabilitación. De forma que se facilite la labor tanto de los especialistas como de los usuarios.

El dispositivo de detección de movimientos que se ha empleado es la nueva Kinect para Xbox One de Microsoft, que junto con el motor de edición de videojuegos Blender van a constituir las herramientas principales del sistema. Se necesita una implementación que constituya la base de comunicación entre ambas y permita que trabajen simultáneamente.

La estructura de la memoria será la siguiente: Primero, se presenta un capítulo de análisis del estado actual de la investigación, en el que se introducen las técnicas de detección facial más empleadas a día de hoy, un estudio minucioso del dispositivo Kinect y las librerías de desarrollo que se ofrecen. A continuación, se muestran los objetivos generales y específicos que se pretenden alcanzar con el sistema propuesto. Le sigue el capítulo en el que se analiza con todo detalle el desarrollo y funcionamiento completo del proyecto. También se incluyen los datos y resultados de las pruebas que se realizaron en un Centro de Educación Especial en casos reales de niños discapacitados. Para finalizar con las conclusiones y futuras líneas de trabajo e investigación.

2. Análisis del estado actual de investigación

2.1. Reconocimiento facial

Por el mero hecho de ser seres humanos ya tenemos determinados rasgos morfológicos que nos distinguen del resto de seres vivos. Además existen una serie de rasgos más específicos como la forma de nuestra cara, la geometría del cuerpo, la retina, el iris y la huella dactilar tanto de los pies como de las manos que constituyen el patrón único de cada individuo. La biometría es el campo de la ciencia que se encarga de estudiar estas diferencias biológicas. El propio término deriva de la unión de las palabras griegas "bio" que significa vida y "metría" que significa medida o medición.

A lo largo de la historia se desarrollaron diferentes tesis y estudios sobre la clasificación de huellas dactilares, pero no fue hasta finales del siglo XIX cuando se encuentra utilidad práctica a esta rama de la ciencia con el sistema antropométrico desarrollado por el criminólogo francés Alphonse Bertillon para identificar criminales. Destacar también al oftalmólogo Frank Burch que en 1936 desarrolló la idea de usar patrones de iris como método de identificación [1].

La Real Academia de la Lengua Española (RAE) da la siguiente definición de biometría:

> *"Estudio mensurativo o estadístico de los fenómenos o procesos biológicos."*

Ante la creciente necesidad y rápido avance que los sistemas de reconocimiento biométricos han tomado recientemente puede parecer que esta definición es imprecisa, por lo que está tomando más relevancia la siguiente definición

> *"La biometría es el estudio de métodos automáticos para el reconocimiento único de humanos basados en uno o más rasgos conductuales o físicos intrínsecos."*

En la biometría se distinguen dos grupos de registros biométricos, los fisiológicos o morfológicos y los conductuales.

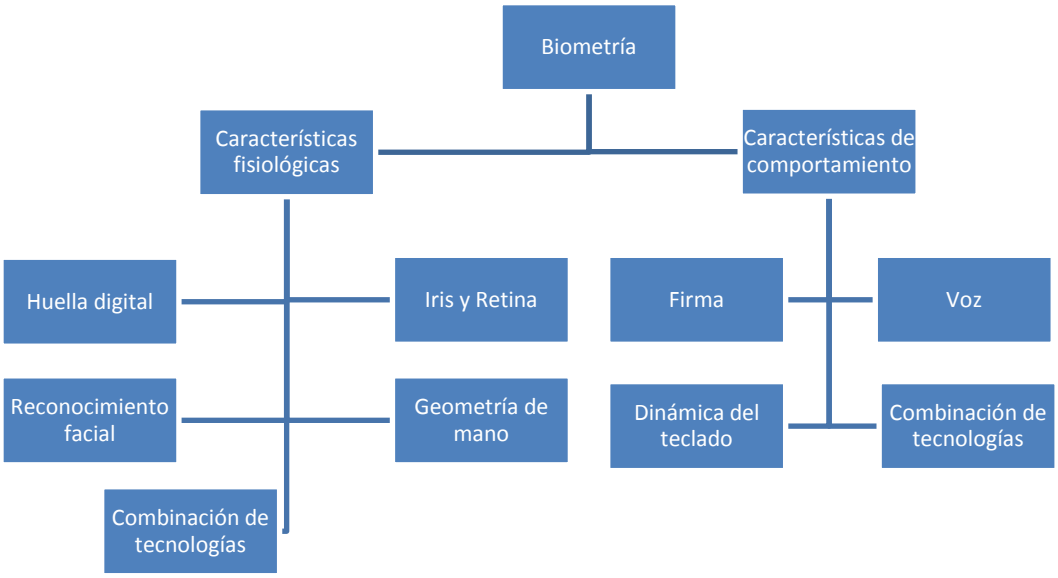


Figura 1. Clasificación recursos biométricos

Los registros morfológicos son aquellas características físicas innatas e inalterables que todo ser humano posee, algunas de las más utilizadas suelen ser las huellas dactilares, el iris del ojo y la geometría de la mano o el pie. Por otro lado están los registros conductuales que están

ligados a determinadas conductas del ser humano, como las pulsaciones del teclado, trazado de la firma e incluso características de la voz.

Es por todos sabido que en los últimos años se han dado grandes pasos en los sistemas de reconocimiento, ya sea utilizando rasgos de la cara, del iris del ojo o por medio de los tradicionales análisis de la voz y de huellas dactilares. Pero el trabajo que se plantea en este proyecto está enfocado al reconocimiento facial, por lo que de aquí en adelante nos centraremos en esta parte de la biometría.

Los métodos de reconocimiento facial que se han desarrollado hasta el momento se pueden dividir en dos grandes grupos, los fotométricos y los geométricos.

2.1.1.Métodos fotométricos

Tienen en cuenta la totalidad de la imagen facial, son métodos que se basan en la correlación estadística. Las características se extraen mediante en filtrado de la imagen o de una región de la misma. El proceso de reconocimiento consiste en tomar una imagen de dos dimensiones, *m* filas y *n* columnas, y transformarla en un vector unitario. Después se le substrae la imagen promedio y se proyecta el vector resultante en un subespacio de menor dimensión. Esta proyección es comparada con la proyección de un conjunto de imágenes de una base. La clase del vector más similar es el resultado del proceso de reconocimiento.

En los años 70 Goldstein, Harmon, &Lesk [2], usaron 21 marcadores subjetivos específicos tales como el color del cabello y grosor de labios para automatizar el reconocimiento facial. Conforme a que el interés en reconocimiento facial continuó, fueron desarrollados muchos algoritmos diferentes de los cuales destacan tres.

Análisis de Componentes Principales(*Principal ComponentAnalysis*, *PCA*). Es comúnmente considerado uno de los algoritmos que ofrece un mayor rendimiento. Con él se permite representar una imagen de una cara usando una base vectorial de menor dimensión que se ha conseguido a partir de muchas observaciones de diferentes caras. Su funcionamiento se basa en la proyección de imágenes faciales sobre un espacio vectorial que recibe el nombre de "facciones", el cual engloba las variaciones significativas entre las imágenes faciales conocidas. Las facciones significativas se llaman **Eigenfaces**, ya que son los componentes principales del conjunto de caras. La proyección caracteriza la imagen facial de un individuo como la suma de los diferentes pesos de todas las facciones y, de la misma manera, para reconocer una imagen facial determinada sólo hará falta comparar estos pesos con aquellos de los individuos ya conocidos [3].

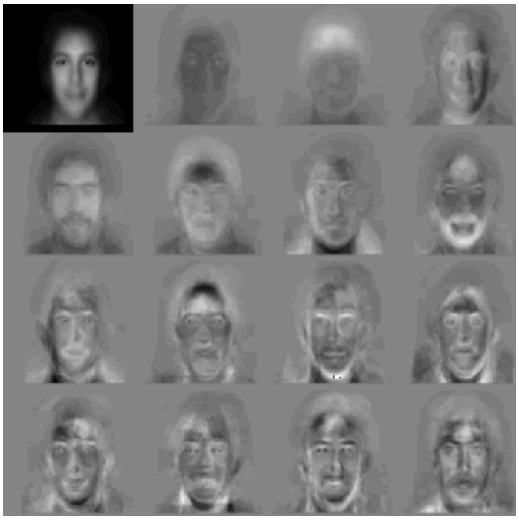


Figura 2. Eigenfaces estándar de diferentes individuos [3]

Análisis Lineal Discriminante (*Linear Discriminant Analysis, LDA*). Se trata de una aproximación estadística basada en la clasificación de muestras de clases desconocidas por medio de ejemplos de entrenamiento de clases conocidas. Es decir, permite utilizar la información entre miembros de la misma clase (imágenes de la misma persona) con la que generar un conjunto de vectores de características donde la variación entre las distintas caras se enfaticen mientras que los cambios producidos por la iluminación, expresión facial y orientación de la cara no. Consigue maximizar la varianza entre clases (entre usuarios) y minimizar la varianza de cada clase (de cada usuario) [4], [5].

Figura 3. Ejemplo de seis clases usando LDA [5]

Análisis Discriminante Lineal de Fisher (*Fisher Discriminant Analysis, FDL*). Sus términos son a menudo usados para expresar la misma idea que el LDA, pero realmente el discriminante es ligeramente distinto ya que no se hace la suposición de que existe una distribución normal entre las clases o covarianzas iguales entre las mismas, como sucedía en LDA. Este método no deja de ser una medida de la señal de ruido para la clasificación entre clases.

Figura 4. Comparación entre LDA (a) y FDL (b) [4]

2.1.2. Métodos geométricos

Se rigen por el análisis y comparación de diferentes características geométricas de las caras. Existen dos divisiones, la basada en los vectores característicos extraídos del perfil, y la basada en los extraídos a partir de una vista frontal.

Modelo de Apariencia Activa (*Active AppearanceModel, AAM*). Este método adapta imágenes de superficies que contienen deformaciones no rígidas o cambios de apariencia a un modelo estadístico de forma, que se ha creado previamente en una fase de entrenamiento utilizando un conjunto de imágenes con el que obtener un modelo estadístico de la forma y apariencia del objeto de interés, en nuestro caso una cara. El modelo estadístico se describe mediante un conjunto de *N* puntos característicos (los parámetros de forma) situados estratégicamente en

la imagen para conformar una malla triangular en 2D que representa la forma del rostro, como se puede ver en la Figura 5 [6].

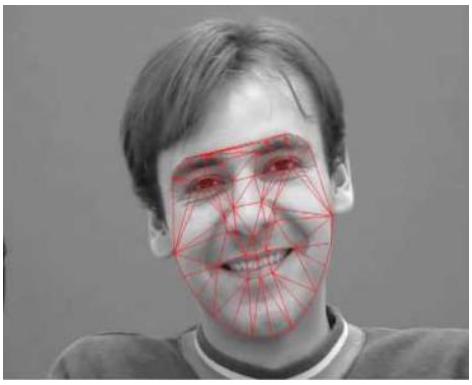


Figura 5. Malla 2D triangular del modelo AAM [6]

Correspondencia entre agrupaciones de grafos elásticos (*ElasticBunchGraphMatching, EBGM*). Construye rasgos locales de la cara utilizando algunos puntos de interés, no la totalidad de la misma. Este algoritmo trabaja en dos etapas: primero se ajusta groso modo un grafo cuyos nodos coinciden con los puntos principales de la cara de un individuo (ojos, nariz y boca) por medio de un modelo estadístico y se normaliza la imagen para ubicar la posición de los ojos en unas coordenadas predeterminadas. Cada nodo del grafo es caracterizado utilizando un banco de filtros de Gabor, se trata de filtros especiales Paso-Banda que ofrecen una descripción de la información en frecuencia para una región específica de la imagen. Después quedará por lo tanto cada nodo determinado por sus coordenadas y por la convolución de la región de la imagen en que se encuentre dicho nodo con un banco de filtros de Gabor con diferentes frecuencias y orientaciones. Al entrar una nueva imagen para clasificar se comparan los descriptores de los nodos del grafo obtenido con los grafos almacenados previamente, dependiendo de la distancia encontrada se ratifica la similitud de las caras [7].

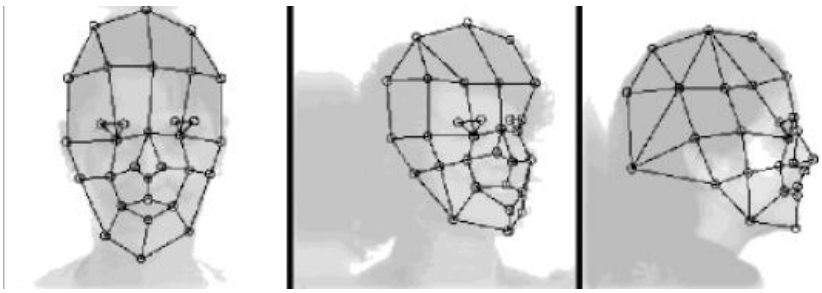


Figura 6. Grafos trazados desde distintos ángulos de la misma persona del modelo EBGM [7]

Modelo de forma activa (*Active ShapeModel, ASM*). Hace uso de un modelo de estimación inicial previo de lo que se espera encontrar en la nueva imagen, y trata de buscar la mejor posición de coincidencia del modelo y los datos de la imagen de entrada. Se compone de tres elementos: el modelo de contorno, que asegura que la segmentación produce contornos válidos para una segmentación facial; modelo de apariencia, que se encarga de que la segmentación localice la cara en la posición donde la estructura de la imagen alrededor y en el interior del contorno sea similar a la estructura de las imágenes de entrenamiento archivadas; y el algoritmo de búsqueda, por el que el contorno facial se ajusta mediante un proceso iterativo, en el que en cada iteración las segmentaciones se mueven *N* posiciones en la dirección perpendicular al contorno para colocarse en la posición que ofrece menor distancia a la realidad.

Inicial | Después de 2 interacciones | Después de 6 interacciones | Después de 18 interacciones

Figura 7. Proceso del modelo ASM [7]

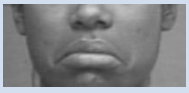
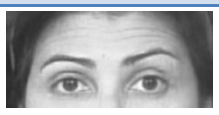
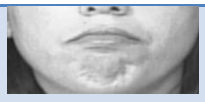
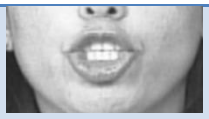
Una vez estudiados los diferentes modelos de reconocimiento facial desarrollados hasta el momento, es interesante profundizar ahora en el reconocimiento de movimientos faciales cuando expresamos una emoción determinada. Charles Robert Darwin (1809-1882) parece ser el primer antecedente del estudio e investigación de la expresión de las emociones, asumiendo la existencia de una serie de emociones básicas, innatas y de carácter universal, presentes en todos los seres humanos, es decir, los movimientos de los músculos faciales no varían entre individuos y razas.

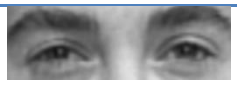
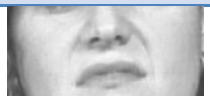
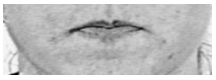
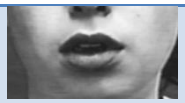
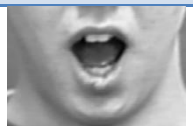
2.1.3. Detección de emociones faciales

El sistema más popular en la medición de expresiones faciales es el **Método de Codificación de Movimiento Facial (*Facial ActionCodingSystem, FACS*)** desarrollado por Ekman y Friesen en 1978. Hablamos de un sistema descriptivo basado en la anatomía de los músculos de la cara, por el que se pueden registrar los cambios faciales debidos al movimiento de músculos individuales del rostro. Se trata de asociar el movimiento de los músculos que se producen en la cara con la propia expresión que provocan. Este sistema permite descomponer cada emoción en una serie de Unidades de Acción (*ActionUnits, AUs*), cada uno de estos AUs se pueden referir a la acción de un único musculo o al de un conjunto de ellos y son 46 AUs las que están descritas en el método [8].

En la Tabla 1 se representan las AUs más utilizadas.

Tabla 1. *Action Units* según las FACS [9].

AUs	Imagen representativa	Definición	AUs	Imagen representativa	Definición
AU1		Interior de las cejas elevado	AU15		Comisura de los labios hacia abajo
AU2		Exterior de las cejas elevado	AU16		Labio inferior hacia abajo
AU4		Cejas bajadas	AU17		Barbilla elevada
AU5		Párpado superior elevado	AU20		Labios estrechados y estirados en horizontal
AU6		Mejillas elevadas	AU22		Labios crateriformes

AU7		Párpados estrechados	AU23		Labios tirantes
AU9		Nariz arrugada	AU24		Labios presionados
AU10		Labio superior elevado	AU25		Labios separados
AU11		Nasolabial pronunciado	AU26		Boca entreabierta
AU12		Comisuras de la boca estiradas y elevadas	AU27		Boca abierta

Las AUs también indican el grado de intensidad en que se dan, escalado de la letra A a la E:

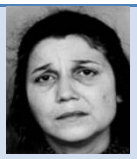
Tabla 2. Grados de intensidad definidos en FACS [10].

Intensidad	Descripción
A	El movimiento es un poco apreciable
B	Se ve el movimiento pero es muy débil
C	El movimiento se ve claramente
D	El movimiento es severo
E	El movimiento es extrema

A través del FACS y sus AUs se ha realizado un listado de los patrones faciales típicos basado en cada una de las seis emociones básicas (Tabla 3). Aunque con el paso de los años se han ido realizando pruebas en situaciones reales y los resultados no son tan claros como los que planteaba Ekman, sigue siendo un descubrimiento importante.

Tabla 3. Correspondencia AUs para cada expresión según las FACS [11], [12].

Expresión	Descripción	AU's	
Fear	Interior y exterior de cejas elevado, párpado superior elevado, labios estrechados o boca entreabierta	AU1, AU2, AU5, AU20, AU26	
Anger	Cejas bajadas, párpados estrechados, barbilla elevada y labios presionados	AU4,AU7,AU17,AU24	

Sadness	Interior de cejas elevado, párpados estrechados, comisura de los labios hacia abajo, barbillas elevada o labios presionados	AU1, AU6, AU15, AU17, AU24	
Disgust	Cejas bajadas, nariz arrugada y labio superior elevado	AU4, AU9,AU10	
Happy	Mejillas elevadas, comisuras de la boca estiradas y elevadas o con labios separados	AU6, AU12, AU25	
Surprise	Interior y exterior de cejas elevado, párpado superior elevado y boca entreabierto o abierta	AU1, AU2, AU5, AU26, AU27	

Es muy importante seleccionar los puntos de la cara más relevantes y que nos aporten la mayor información de movimiento posible, no va a dar la misma información un punto de la parte superior de la frente que otro situado alrededor de los labios que tienen mayor capacidad de acción. Uno de los inconvenientes de este método es que la cara tiene que estar totalmente de frente para que la detección sea óptima, si la cara está inclinada o parcialmente tapada se producen incoherencias.

Otro método de detección facial de emociones es **LBP (*Local BinaryPatterns*)**que se basa en el estudio de las texturas que aparecen en la imagen, etiquetando píxel a píxel según el nivel de umbralización teniendo en cuenta los píxeles vecinos mediante un número binario de ocho dígitos. Así es capaz de detectar información espacial por cambios de iluminación el movimiento del músculo de la cara [13].

Destacar también otro sistema que ha tenido bastante éxito y cada vez es más empleado en el ámbito del reconocimiento facial como es **SIFT (*ScaleInvariantFeatureTransform*)** [14]. Es un algoritmo usado para extraer características de las imágenes que puedan ser posteriormente usadas para reconocimiento de objetos o detección de movimientos en este caso. Trabaja con una serie de puntos de interés de los que parte para crear un vector de características utilizando la magnitud del gradiente en cada uno de ellos para captar posibles movimientos en la escena.

Existen otros muchos métodos de detección de movimiento en imágenes, pero los tres anteriormente explicados son los más óptimos para el reconocimiento de movimientos faciales.

2.2. Kinect

En esta sección se va a hablar sobre la Kinect de Xbox siendo la base de las herramientas que emplearemos para el desarrollo de este proyecto. Veremos la evolución que ha tenido a lo largo de los años, sus características y funcionalidades, para dar una idea general de las amplias posibilidades que puede llegar a ofrecer este dispositivo

2.2.1. Introducción

Kinect es un sistema de reconocimiento de movimientos que se define como "un controlador de juego libre y entretenimiento" creado por el brasileño Alex Kipman y desarrollado por Microsoft para la consola de videojuegos Xbox360, siendo compatible actualmente con PCs que corran Sistemas Operativos Windows 7, Windows 8 y Windows 10 [15].

El gran avance que supone este dispositivo es el poder que da a los usuarios de controlar y de interactuar con la consola sin necesidad de tener contacto físico con un controlador de videojuegos tradicional, mediante una interfaz natural de usuario que reconoce movimientos, gestos, comandos de voz, e incluso objetos e imágenes.

Los creadores no fueron conscientes de la repercusión que podría llegar a tener este dispositivo ya que el único motivo que perseguían era atraer a un público más amplio con la innovadora forma de jugar con tu propio cuerpo de una forma sencilla y que no llevaba demasiado tiempo aprender. Pronto especialistas de distintos campos profesionales empiezan a hacer uso de esta cámara para diversos fines, se llega a emplear para tratamientos de Alzheimer, rehabilitación de algún grado de discapacidad física, espejo virtual para tiendas de ropa, y hasta estudio del movimiento de glaciares.

Paralelamente a Kinect se desarrollaron otros dos dispositivos que trabajaban con la misma idea de juego por control de movimiento, pero que necesitan de un controlador:

Wiimote de la Wii de Nintendo (Figura 8), que detecta el movimiento gracias a un giroscopio y un acelerómetro en el mando, detectando cambios en la orientación y en la aceleración. Otro dispositivo que ofrece Nintendo para esta consola es el Wii Balance Board, básicamente es una tabla que se coloca en el suelo y detecta la presión que se ejerce sobre ella gracias a cuatro sensores que incorpora.

Figura 8. Videoconsola Wii con mando Wiimote y Wii Balance Board

PlaystationMove para la consola PlayStation de Sony (Figura 9), que utiliza sensores de movimiento y ubica la posición 3D del mando en el plano gracias a una cámara que reconoce el color de la esfera iluminada sobre el mismo. Sony también ha desarrollado quizás el que más se asemeje a Kinect, PlayStation Camera para PlayStation 4, ofreciendo la capacidad de seguimiento del esqueleto y de la cara, pero sin la posibilidad del reconocimiento de los puntos faciales que sí hace Kinect.

Figura 9. PlayStation Move y PlayStation Camera

2.2.2. Características técnicas y componentes

Actualmente se puede hablar de dos generaciones de Kinect, la primera desarrollada para Xbox 360 y la segunda para Xbox One, además cada una de ellas dispone de un adaptador que hace posible su conexión a PC. En este apartado se describen en detalle sus características.

2.2.2.1. Primera generación

El dispositivo Kinect para Xbox 360 tiene forma rectangular apoyada sobre una base articulada que permite modificar la orientación del dispositivo. Este sensor cuenta con una cámara RGB, una cámara de profundidad, un emisor láser de infrarrojos y un micrófono MultiArray.

Cámara RGB: es la encargada de capturar y transmitir los datos de vídeo a color. Tal y como su propio nombre indica detecta los colores rojo, verde y azul de la escena (más una componente de transparencia, α, para imágenes RGBa)para generar píxel a píxel los cuadros de imagen fijos de salida organizados en un vector. Cada componente de color del píxel tiene un valor decimal entre 0(negro) y 255(blanco) que corresponde a 1 byte. De forma que a la salida del sensor se obtiene un vector de bytes con la información de color de todos los píxeles de la escena. La organización de los píxeles en el vector corresponde a recorrer la imagen de arriba a abajo y de izquierda a derecha, como se muestra en la Figura 10.

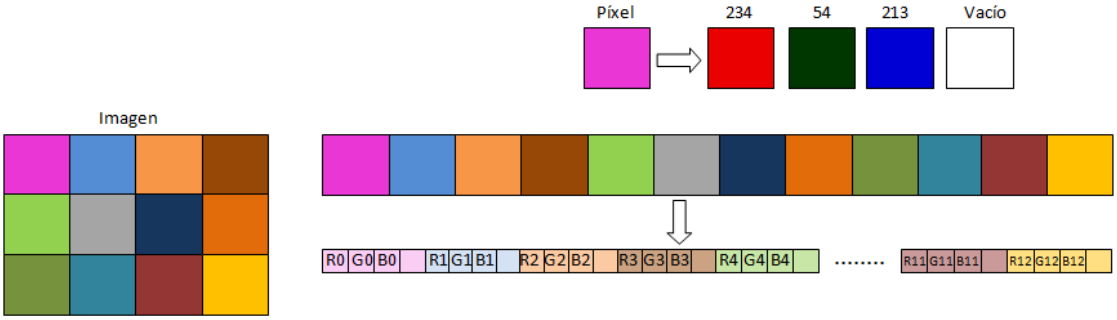


Figura 10. Composición del vector de salida del sensor en cada cuadro de imagen

Cámara de profundidad IR y emisor láser de infrarrojos IR: La acción conjunta de ambos hace posible la captación de imágenes de profundidad. A simple vista el emisor láser puede parecer una cámara más, pero es un proyector de luz infrarroja que se emite de forma constante. La cámara infrarroja recoge los haces IR que rebotan, obteniendo así la información de profundidad con la medida de la distancia entre el sensor y el punto del que vino el haz.

Micrófono MultiArray: consta de cuatro micrófonos distribuidos en línea (Figura 11), esto explica que el dispositivo sea alargado, que capturan la voz y además sitúan el origen y la dirección de la fuente sonora.

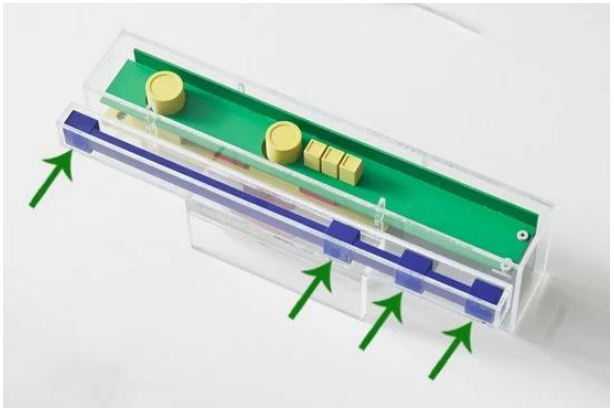


Figura 11. Distribución de los micrófonos en Kinect

Motor de inclinación: Entre la base y el cuerpo que aloja las cámaras y los micrófonos se encuentra un motor que inclina verticalmente el sensor hasta 27 grados.

También incorpora un LED que indica si los controladores de Kinect están correctamente instalados.

Figura 12.Cámara Kinect Xbox 360

Listado completo de especificaciones [16]:

- Sensores
  - Lentes de color y sensación de profundidad
  - Micrófono multimatriz
  - Ajuste de sensor con su motor de inclinación
  - Totalmente compatible con las consolas existentes de Xbox 360
- Campo de visión
  - Campo de visión horizontal: 57 grados
  - Campo de visión vertical: 43 grados
  - Rango de inclinación física: ± 27 grados
  - Rango de profundidad del sensor: 1,2 - 3,5 metros

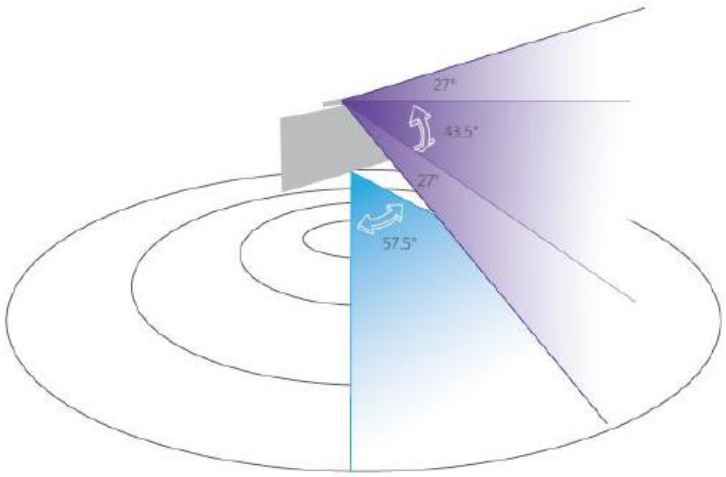


Figura 13. Ángulos de visión vertical, horizontal e inclinación [17]

Rangos de distancia de profundidad (modo predeterminado):

- Límites físicos: 0,8 a 4 m
- Límites prácticos: 1,2 a 3,5 m

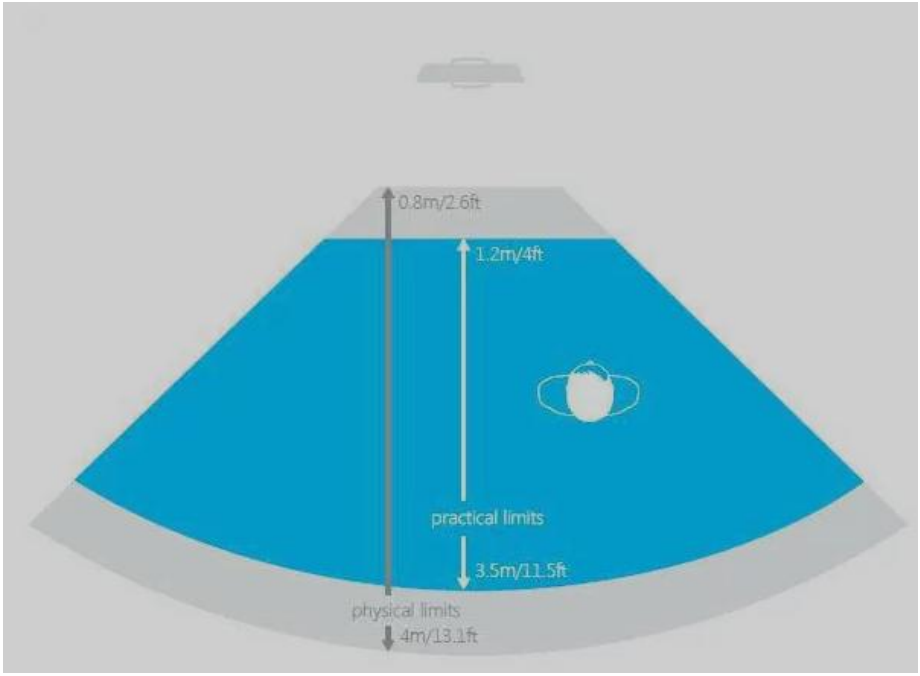


Figura 14. Rangos de distancia de profundidad modo predeterminado [17]

Rangos de distancia de profundidad (modo cerca):

- Límites físicos: 0,4 a 3 m
- Límites prácticos: 0,8 a 2,5 m

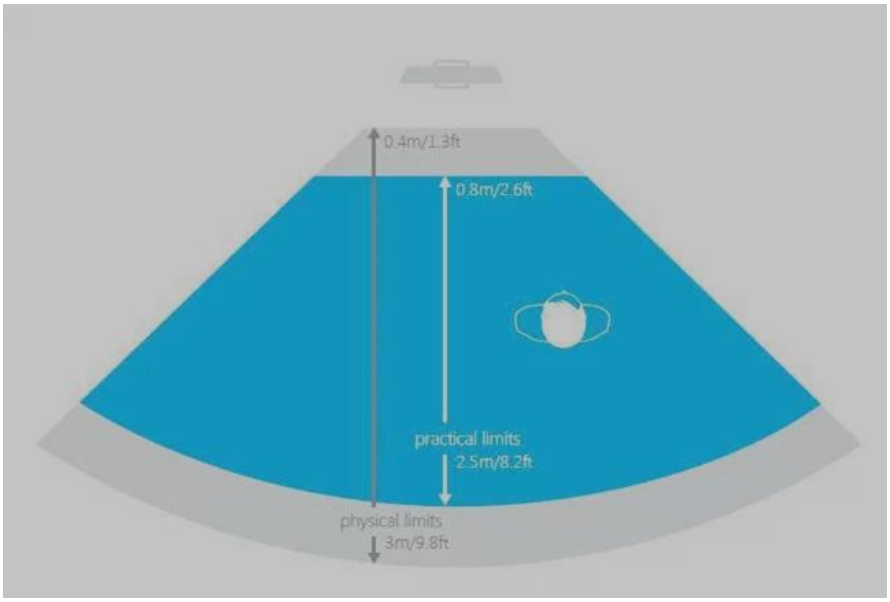


Figura 15. Rangos de distancia de profundidad modo predeterminado [17]

- *Data Stream* (Flujo de datos)
  - 320 × 240 a 16 bits de profundidad @ 30fps
  - 640 × 480 32-bit de color @30fps
  - Audio de 16-bit @ 16 kHz
- Sistema de Seguimiento
  - Rastrea hasta 6 personas, incluyendo 2 jugadores activos
  - Rastrea 20 articulaciones por jugador activo (1 o 2 jugadores) y la posición de hasta 6 jugadores
  - Capacidad para mapear jugadores activos en *Live Avatars*

- Sistema de audio
  - Chat en vivo y voz dentro del juego (requiere *Xbox Live Gold*)
  - Sistema de cancelación de eco que aumenta la entrada de voz
  - Reconocimiento de voz múltiple

2.2.2.2. Segunda generación

Posteriormente, en el año 2013 se lanzó la nueva consola Xbox One de Microsoft (Figura 16), junto a una segunda versión del sensor de movimientos (Kinect 2.0), que ofrece una serie de mejoras y nuevas posibilidades de desarrollo respecto a la primera generación.

Figura 16. Cámara Kinect Xbox One

Resolución mayor: 1920x1080 Full HD. A mayor resolución mayor detalle y mayor calidad de la imagen, permite detectar con más precisión todo el entorno, influyendo esto directamente en la capacidad de diferenciar la orientación del cuerpo, el movimiento independiente de los dedos de la mano y hasta la captación de gestos de la cara.

Campo de visón mayor: aumenta su campo de visión en un 60% respecto al primer Kinect. El ángulo de captación horizontal pasa de 57º en la primera generación a 70º en la segunda, y el de captación vertical de los 43º a los 60º (Figura 17). Al tener un campo de visión mayor se da la posibilidad de incluir un mayor número de jugadores en la escena, hasta seis simultáneamente.

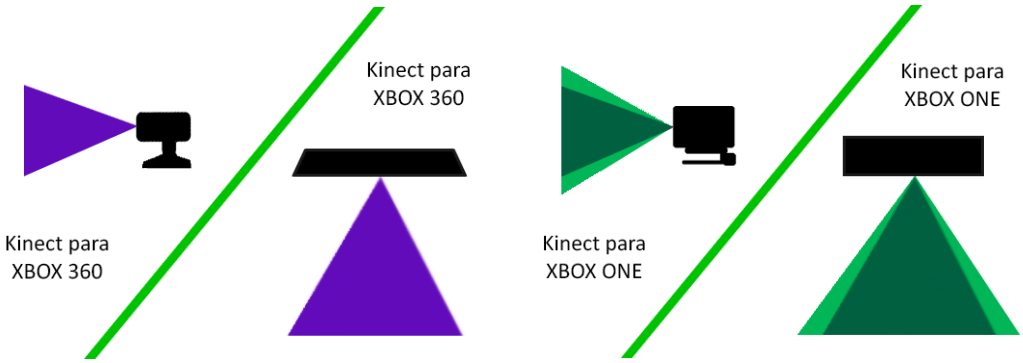


Figura 17. Comparación del campo de visión [18]

Rango de profundidad detectado: el rango de profundidad pasa a ser de 0,5 metros a 4,5 metros, como se observa en la Figura 18.

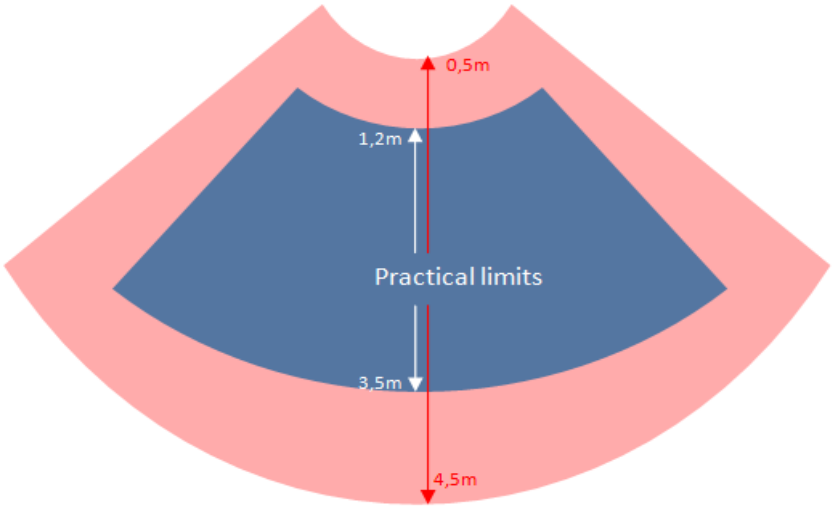


Figura 18. Rango de profundidad (Rojo Kinect v2, azul Kinect v1)

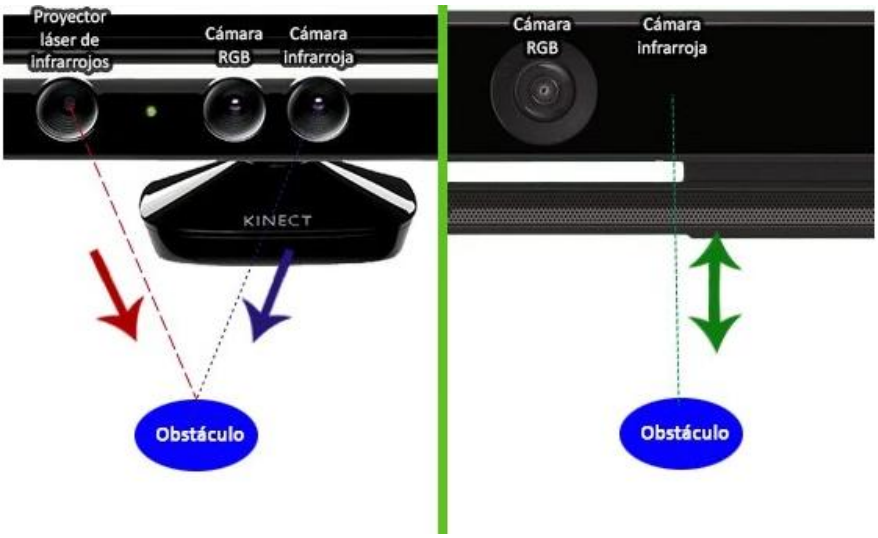


Figura 19. Detección de profundidad (Izquierda Kinect v1, derecha Kinect v2) [18]

Como se puede comprobar, el concepto de detección de profundidad es diferente en esta nueva versión. Una única cámara de infrarrojos moderna, capta únicamente al individuo por medio de un láser. Este láser vuelve al lugar de origen, es una ida y vuelta continua (Figura 19).

Captación de movimientos a oscuras: aunque la escena no tenga ningún tipo de iluminación, Kinect 2 es capaz de trabajar con normalidad.

Detección de esqueleto y cara mejorada: es capaz de detectar seis esqueletos completos con 25 articulaciones cada uno (Figura 20), y hasta 1347 puntos del rostro (la Kinect de primera generación sólo detectaba 20 articulaciones y 121 puntos faciales).

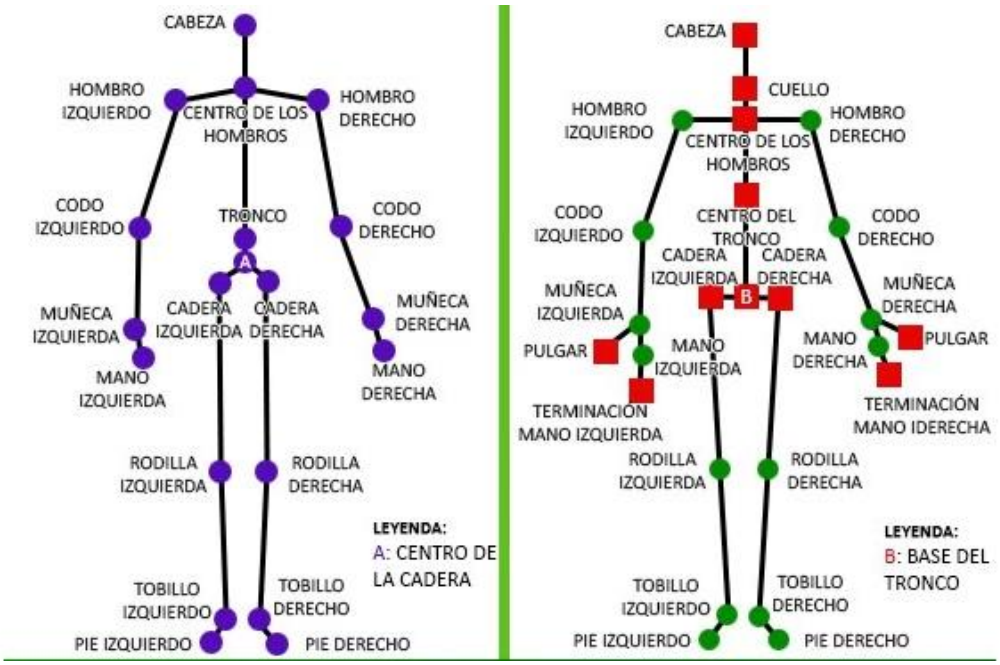


Figura 20. Detección de esqueleto (Izquierda Kinect v1, derecha Kinect v2) [18]

Además gracias a funciones del SDK 2.0, se pueden detectar expresiones faciales y estados de ánimo. Sabe exactamente en cada instante si se tiene boca abierta, boca cerrada, si se habla, si se ríe, etc. (Figura 21).

Figura 21. Expresiones detectadas por el SDK 2.0

Captación de sonidos mejorada: se ha conseguido eliminar el ruido ambiente mediante filtrado en frecuencia, lo que permite captar con mayor precisión las instrucciones de voz.

Conexión USB 3.0: esto repercute en una mayor velocidad de comunicación entre el dispositivo y el PC, los datos fluyen más rápido y disminuye la latencia del sensor pasando de 90 a 60 ms.

Permite captar y calcular la fuerza de nuestros músculos y medir el ritmo cardiaco: por medio de la interacción entre la cámara RGB y el sensor IR, se consigue ver el cambio de

color que se produce en la piel según esté bombeando el corazón, lo que le proporciona en tiempo real el pulso cardiaco del jugador.

Figura 22. Detección del punto de apoyo del jugador [18]

2.2.3. Librerías para desarrollo

Software Development Kit (SDK) de Windows, *Open Natural Interaction* (OPENNI) y *OpenKinect*, son las tres plataformas de desarrollo de aplicaciones para Kinect que se vienen utilizando recientemente.

Software Development Kit (SDK) de Microsoft

Como su propio nombre indica se trata de un kit de herramientas que ofrece Microsoft para el desarrollo de aplicaciones únicamente en el dispositivo Kinect escritas en lenguajes de programación C++, C# o *VisualBasic*, aunque no todas las funciones están disponibles en los tres lenguajes. Incluye los drivers necesarios para que Windows reconozca el dispositivo y poder trabajar con él, además contiene una gran variedad de APIs (*ApplicationProgramming Interface*) fáciles de emplear y con una documentación muy extensa. Cabe destacar que es una plataforma en constante evolución con importantes novedades y total compatibilidad con las versiones anteriores.

Open Natural Interaction (OPENNI)

Conocida como la alternativa *OpenSource* para interactuar con Kinect se caracteriza por su amplia adaptabilidad para conexiones middleware entre el sensor y otros dispositivos. Un ejemplo sería NI-Mate que extrae los datos del Kinect y los retransmite utilizando el protocolo OSC a los distintos programas con los que es compatible. No deja de ser un espacio de trabajo que provee APIs para el desarrollo de aplicaciones que utilicen interacción natural con el usuario. La principal ventaja es que es multiplataforma, pudiendo ser utilizada en Windows, Linux o Mac OS X. Al igual que el SDK de Microsoft posee una documentación bastante extensa y detallada, pero sólo en lenguajes de programación C++ Y C.

OpenKinect

Lo conforman una comunidad de personas interesadas en obtener el máximo provecho del increíble hardware de Xbox Kinect en nuestros PCs, movidos por este interés existen multitud de *wrappers* y lenguajes de programación que soporta. Evidentemente es una plataforma *OpenSource* pero con una librería de muy bajo nivel y pocas funcionalidades.

2.3. Herramientas de comunicación entre Kinect y otras plataformas

A día de hoy existen varias herramientas de conexión entre dispositivos Kinect y otras plataformas según la librería de desarrollo implementada.

La organización OpenNi tiene kits de desarrollo que permiten acceder específicamente a los servicios provistos por el Kinect y otros dispositivos de interacción natural. Adicionalmente esta empresa desarrolla el *middleware* NITE el cual no es de código abierto y permite acceder a funcionalidades avanzadas como seguimiento del esqueleto en tiempo real y reconocimiento de gestos.

ROS (*Robot OperatingSystem*) es un marco de trabajo para el desarrollo de software en robots que da soporte a Robots con inteligencia artificial. Desarrollaron un middleware que conseguía la comunicación del robot con el Kinect de primera generación, para detectar movimientos y obstáculos que puedan encontrarse estos robots en su trayectoria [19].

BlenderLoopStation o Bloop ,una solución gratuita que consiste en un *addon* para Blender que permite el control del movimiento de un personaje captado por la Kinect v1, utilizando simplemente las manos del usuario [20].

Destacar NI-Mate basado en OpenNi que ha sido citado anteriormente, tiene como aplicación principal un emisor que obtiene los datos del Kinect y los retransmite utilizando los protocolos OSC o MIDI. Existen *addons* que permiten instalar el receptor para esta transmisión en distintas plataformas software.

3. Descripción de la herramienta propuesta

El objetivo global de este Proyecto Fin de Grado es crear la base de comunicación entre el sensor Kinect 2.0 y el motor de modelado y creación de videojuegos Blender, para el desarrollo de un juego serio de rehabilitación orientado a movimientos faciales.

Actualmente un grupo de investigación del CITSEM (Centro de Investigación en Tecnologías Software y Sistemas Multitarea para la Sostenibilidad) está en desarrollo de un videojuego serio de rehabilitación corporal sobre Kinect 1, más centrado en el movimiento de articulaciones y extremidades. Incluso cuenta con la posibilidad de la inclusión de las gafas virtuales (*OculusRift*), cuyo objetivo es crear una sensación de inmersión mayor y aprovechar el sensor de giro de la cabeza para juegos. Este proyecto sería una importante contribución a esa investigación para poder dar el paso a la utilización de la segunda versión de Kinect, e incluir ejercicios faciales.

3.1. Objetivos específicos

Para realizar un sistema software capaz de reconocer rasgos y movimientos faciales en un videojuego será necesario integrar un dispositivo capaz de recoger información sobre los movimientos del usuario del mismo. Ahí es donde entra en juego el sensor Kinect 2.0, a partir del cual se podrán extrapolar las coordenadas de los distintos puntos de interés del cuerpo, en este caso de la cara.

Un vez obtenida la información necesaria sobre el esqueleto y la cara, dicha información será procesada y enviada al motor Blender. Finalmente dependiendo de la aplicación que se vaya a dar al juego, se tendrá en cuenta la información obtenida del esqueleto, de la cara o del conjunto.

Para la obtención de este resultado se pretenden llevar a cabo las siguientes implementaciones.

3.1.1. Creación de un *middleware* + *plugin* de Blender compatible con Kinect 2.0

Al ser Kinect 2.0 el dispositivo elegido para recopilar la información del cuerpo del usuario a tiempo real, es necesario analizar el SDK 2 compatible con este dispositivo y las respectivas librerías disponibles para conocer el tipo de datos que éstas proporcionan y en qué formato, ya que es la información que habrá que transmitir mediante el *middleware*.

Como punto de partida se va tomar el *middleware* desarrollado para el juego serio de rehabilitación del grupo de investigación del CITSEM que ya se ha comentado. Por ello es preciso hacer un estudio del mismo.

3.1.1.1. *Middleware* + *plugin* de Blender compatible con Kinect 1

El *middleware* para Kinect 1 desarrollado por Ignacio Gómez es el que se va a tomar como referencia para realizar la comunicación con Kinect v2 y Blender. Dos son las razones por las que existiendo ya NI-Mate se opta por implementar un middleware nuevo: por un lado, se buscaba una aplicación de comunicación de software libre y éste requiere la compra de licencia. Y por otro lado conviene disponer de un software basado en el SDK de Microsoft porque es la librería de desarrollo puntera en el ámbito Kinect [22].

La aplicación principal trabaja como un emisor que extrae los datos del sensor Kinect y los retransmite por protocolo OSC a un receptor, en este caso Blender. Ésta está basada en la aplicación de software libre KinectOSC que se encuentra disponibles en el SDK 1.8 de Microsoft

en lenguaje C#. Su misión es enviar paquetes de longitud variable a un puerto seleccionable por el usuario.

Protocolo OpenSound Control (OSC)

OSC es un protocolo comúnmente utilizado para la comunicación ordenadores, sintetizadores musicales y otros dispositivos multimedia. Aparece como reemplazo a MIDI siendo muy superior en características y capacidades.

Las características principales de este protocolo son las siguientes:

- Expansible y dinámico.
- Esquema de nombres simbólicos tipo URL.
- Patrones de coincidencia (*Patternmatching*) que permite la comunicación simultánea con varios dispositivos a través de un único mensaje.
- Marcas de tiempo(*Timelags)*, de alta resolución.

La base de la comunicación OSC se encuentra en los mensajes que pueden ser de dos tipos: mensaje único o paquete de mensajes (*bundle*), el paquete de mensajes es un contenedor que puede alojar uno o varios mensajes únicos (Figura 23).

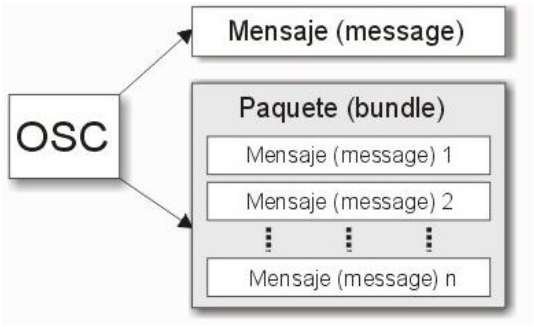


Figura 23. Tipos de mensaje OSC [21]

La sintaxis básica de un paquete OSC incluye 8 bytes que indican el tipo de mensaje, en este caso un *bundle*, le siguen 8 bytes de código de tiempo, 4 bytes que indican la longitud del campo de datos y después los bytes necesarios de datos propiamente dichos (Figura 24).

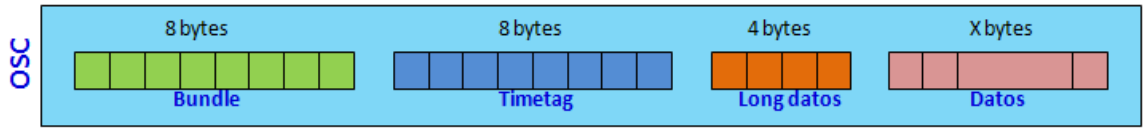


Figura 24. Empaquetado de un mensaje OSC

Para el etiquetado de cada una de los datos a enviar obtenidos por la Kinect, se emplea la estructura de paquete mostrado en la Figura 25.

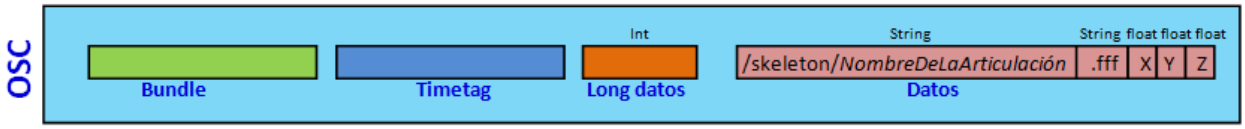


Figura 25. Empaquetado del mensaje OSC utilizado en el middleware

Indicando con un *string* a qué articulación corresponden las coordenadas enviadas. El segundo *string*indica que el siguiente dato va a ser tres *float* seguidos, las coordenadas de la articulación en cuestión.

Para establecer la comunicación entre el *middleware* y Blender es necesario el uso de cuatro puertos por los que circulan diferentes paquetes, *MiddlePort*, *BlenderPort*, *Player1Port* y *Player2Port*. El protocolo de trasporte sobre el que va a estar soportado es UDP, que es no orientado a la conexión, por lo que no se necesita un establecimiento previo de conexión para enviar un datagrama. Quedando el esquema de comunicación mostrado en la Figura 26.

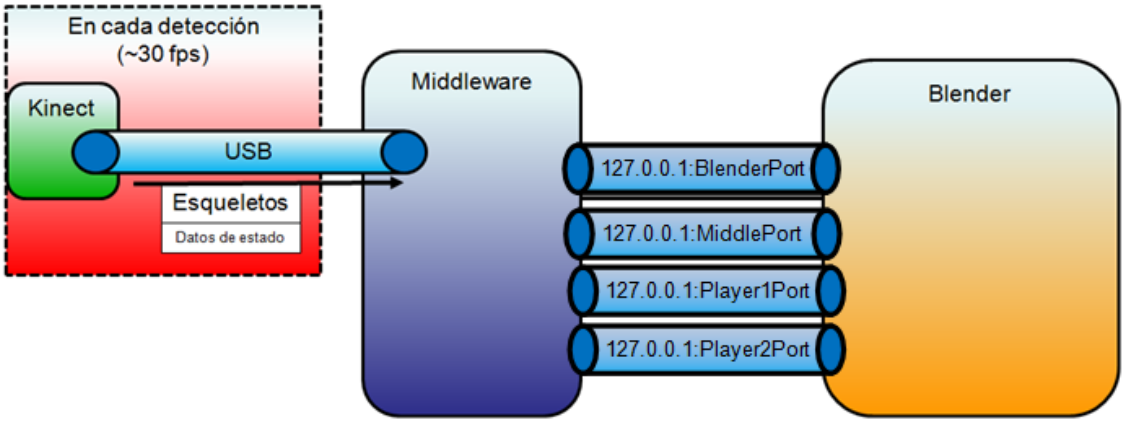


Figura 26. Esquema de comunicación middleware Kinect v1. [22]

Kinect transmite datos de posición de las articulaciones periódicamente al *middleware*. Al iniciar el receptor en Blender (Figura 27), éste envía un mensaje de configuración al puerto *MiddlePort* con el número específico de los puertos que se van a utilizar y la presencia o no del jugador 2. El *middleware* se configura según esta información y envía una respuesta con la altura y la inclinación del Kinect al puerto *BlenderPort* para que Blender reajuste el centro de coordenadas.

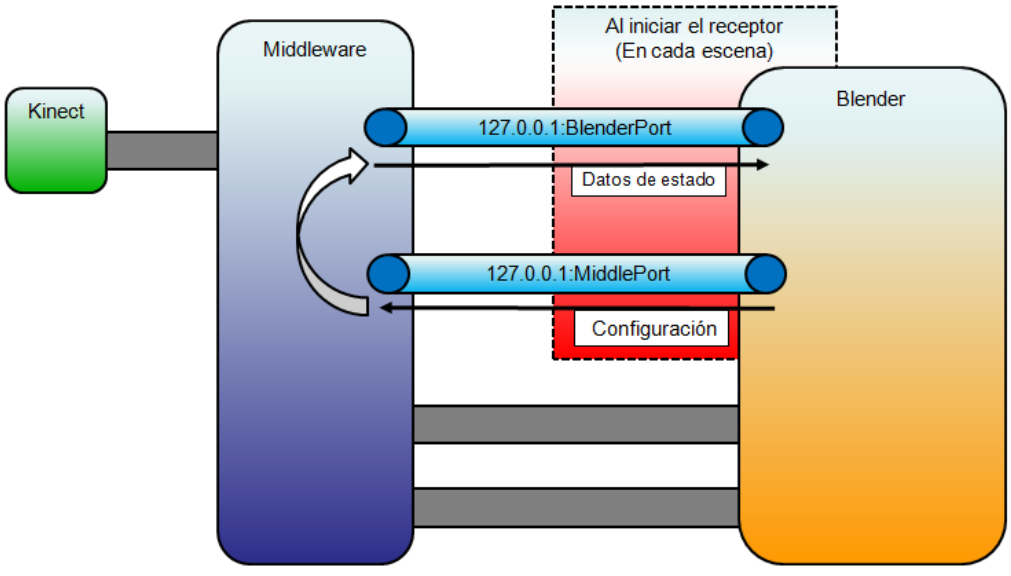


Figura 27. Intercambio de mensajes al iniciar el receptor en Blender [22]

Cada dos cuadros del juego (Figura 28) Blender envía al puerto *MiddlePort* una petición de datos y el *middleware* responde enviando a *PlayerPort1* o *PlayerPort2* los datos de posición que correspondan.

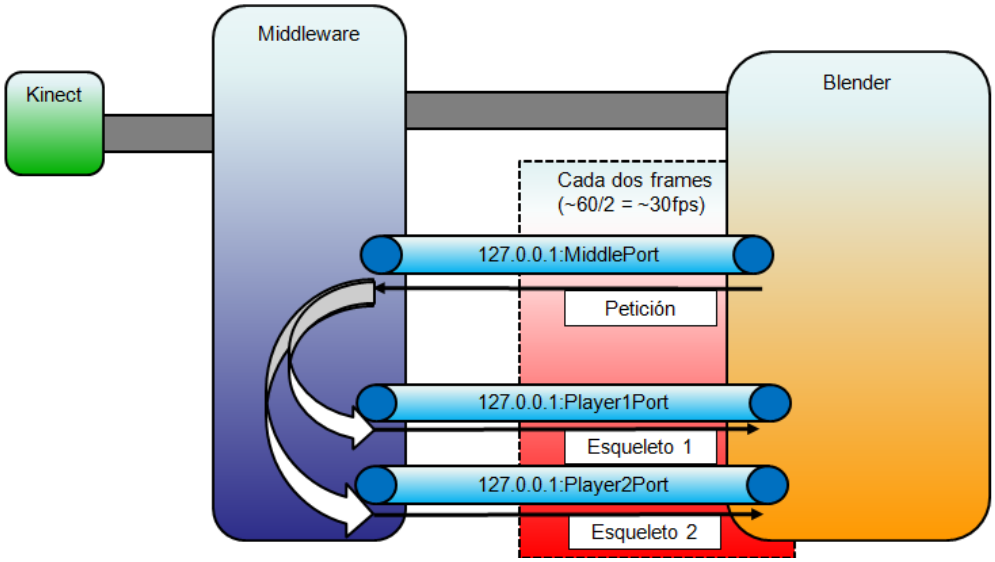


Figura 28. Intercambio de mensajes cada dos *cuadros* del juego [22]

El *middleware* siempre tiene abierto su puerto de escucha (*middelPort*) que lanza un evento cuando recibe un paquete de 17 bytes de longitud. Lee los primeros 4 bytes y actúa según su valor:

- Inicia el procedimiento de ajuste *middleware*, si el valor es "1234567890" (petición de configuración).

Figura 29. Mensaje de configuración

- Inicia el procedimiento de envío de coordenadas si el valor es "876543210" (petición de datos).

Figura 30. Mensaje de petición

El *addon* del código receptor está en lenguaje Python debido a que es el lenguaje en el que trabaja Blender. Blender tiene una interfaz para crear scripts en lenguaje Python, cuyo código se puede proyectar directamente en la ventana 3D o incorporarlo a la lógica del propio juego.

En este *addon* se crea el receptor propiamente dicho, es el llamado "EsqueletoKinect" que se introduce en el entorno y representa esquemáticamente el esqueleto del jugador. Consta de 20 objetos del tipo *Empty*, que reproducen la posición de las 20 articulaciones que da Kinect 1 y objetos *Bones*, que simulan los huesos que unen las mismas. Se elige el tipo de objeto físico *Empty*porque en la imagen final producida por Blender no aparecen (no se ven, pero siguen estando ahí), ya que lo que interesa que se vea sean los huesos a los que están unidas.

El *addon* asigna automáticamente a cada *bone* una restricción de posición, que le obliga a seguir el movimiento de su articulación origen, y una restricción de apuntamiento hacia la siguiente articulación final.

Está configurado para establecer el número de recepciones necesarias por segundo. Al ser la velocidad de Kinect 30 cuadros por segundo, y en cada cuadro se necesita enviar 20 articulaciones se tienen por lo tanto 600 envíos por segundo.

Además se genera un pulso periódico para que cada dosfotogramas del juego realice una llamada a la función de recepción del *addon*con la que se consigue ajustar la tasa de transmisión de cuadro del *GameEngine* al del sensor Kinect que trabaja a una velocidad de 30 fps. Consiguiendo juegos de 60 fps que dan un pulso de 30fps.

3.1.1.2. Diferencias y nuevos objetivos del middleware propuesto

Teniendo en cuenta estas consideraciones, será necesario reajustar este middleware para el tipo de datos de posición que ofrece el SDK2.0 con el que trabaja Kinect 2. En este proyecto se trabaja con la posición de los puntos faciales y no se precisa la unión entre ellos como ocurre con las articulaciones, que realmente son los nexos de unión para orientar los huesos.

La potencia del motor Python permite multitud de posibilidades a la hora de manejar Blender, ya que se puede acceder a prácticamente toda la estructura interna del mismo, por lo que no sólo se pueden dar órdenes a los objetos virtuales del juego sino que se puede añadir al software todas las funcionalidades que se desee. Se pretende obtener una interfaz sencilla, manejable y bastante representativa para facilitar la labor del desarrollador de videojuegos, y que a la vez sea modular para facilitar la inclusión de nuevas funcionalidades en futuras versiones. Se piensa introducir la captación del esqueleto entero para tener la posibilidad de combinar ejercicios faciales y corporales en futuras versiones del *addon*.

3.1.2.Creación de una aplicación de reconocimiento de determinados patrones faciales para efectuar acciones en Blender.

El segundo objetivo es el desarrollo de algún juego en el que se pueda aplicar el reconocimiento de determinados patrones faciales que ofrece Kinect para efectuar alguna acción dentro de Blender. Estos patrones pueden ser abrir o cerrar boca, dar beso, guiñar ojo derecho o izquierdo y sonreír entre otros.

Una vez implementada la aplicación que reconozca estas expresiones, se modelarán en Blender distintos videojuegos en los que la información de estos patrones forma parte de la lógica. Se piensa en una cara que imita la expresión del jugador o distintos ejercicios en los que desplazar un objeto con movimientos de la boca y los ojos, que incluya un reto de puntuación.

Para ello será necesario analizar las librerías que nos ofrece el SDK2 de Microsoft sobre reconocimiento facial y coger los conceptos más importantes que interesen para el desarrollo del proyecto.

Una de las herramientas más potentes que brinda el SDK2 son los componentes que se pueden agregar a los desarrollos para dar diferentes utilidades. Los que proporcionan acceso al seguimiento de la cara son las que se van a emplear en la construcción de este proyecto y serán dos: *Microsoft.Kinect.FaceTracking* y *Microsoft.Kinect.HighDefinitionFaceTracking*.

3.1.2.1. Microsoft.Kinect.FaceTracking

Esta API permite al desarrollador crear aplicaciones en las que se pueden detectar rostros humanos en tiempo real. Su motor analiza los datos obtenidos de la cámara Kinect, rastrea por la escena caras y determinadas expresiones faciales y hace que esta información esté disponible para ser usada en una aplicación en tiempo real.

Permite detectar en 2D los cinco puntos de la cara que aportan más información sobre el reconocimiento de expresiones, ojo derecho e izquierdo, nariz y extremos de la boca. Cabe la posibilidad de que aparezca un rectángulo que encierra al rostro y hace su seguimiento.

Además ofrece la detección de expresiones faciales predeterminadas [23]:

- Felicidad
- Ojo derecho o izquierdo abierto
- Sorpresa
- Boca hablando
- Presencia de gafas
- Mirada hacia o alejada de la cámara

Por cada una de estas expresiones se pueden dar hasta cuatro resultados en forma de número entero que indican su estado(sí, no, quizás o desconocido).

3.1.2.2. Microsoft.Kinect.HighDefinitionFaceTracking

La API de seguimiento de la cara en alta definición permite crear una conexión emocional más fuerte entre los jugadores y el juego. Ésta librería procesa los datos de posición en 3D y permite la detección de hasta 1347 puntos del rostro con los que se forma una máscara de 2630 triángulos. Gracias a esta mayor resolución se puede ahondar aún más en la detección de AUs en torno a emociones.

Esta librería trabaja internamente con el Método de Codificación de Movimiento Facial(FACS*)*, mencionado en 2.1. Se emplean distintas Unidades de Acción, AUs, y cuenta con la detección de 17 AUs que se expresan mediante un peso numérico que varía entre 0 y 1 (Tabla 4).

Tabla 4. AUs que proporciona *Microsoft.Kinect.HighDefinitionFaceTracking* [24]

AU	Descripción
AU0	FaceShapeAnimations_JawOpen
AU1	FaceShapeAnimations_LipPucker
AU2	FaceShapeAnimations_JawSlideRight
AU3	FaceShapeAnimations_LipStretcherRight
AU4	FaceShapeAnimations_LipStretcherLeft
AU5	FaceShapeAnimations_LipCornerPullerLeft
AU6	FaceShapeAnimations_LipCornerPullerRight
AU7	FaceShapeAnimations_LipCornerDepressorLeft
AU8	FaceShapeAnimations_LipCornerDepressorRight
AU9	FaceShapeAnimations_LeftcheekPuff
AU10	FaceShapeAnimations_RightcheekPuff
AU11	FaceShapeAnimations_LefteyeClosed
AU13	FaceShapeAnimations_RighteyebrowLowere
AU14	FaceShapeAnimations_LefteyebrowLowerer
AU15	FaceShapeAnimations_LowerlipDepressorLeft
AU16	FaceShapeAnimations_LowerlipDepressorRight

4. Desarrollo del proyecto

A lo largo de esta sección se va a describir la metodología de trabajo que se ha seguido para realizar el proyecto.

En una primera fase se analizaron las herramientas necesarias para trabajar con Kinect V2 y su SDK compatible, además de las directrices a seguir para desarrollar aplicaciones con la misma. Al ser el objetivo de este proyecto la detección facial, también se investigó sobre las técnicas más relevantes a día de hoy en este campo. Una vez completada la fase de estudio teórica, se pasó a la implementación de un middleware con Blender adaptado a esta segunda generación de Kinect. Debido a que el sistema va a dar soporte a la creación de una serie de videojuegos enfocados al reconocimiento de emociones faciales se van a emplear las posibilidades ofrecidas por el SDK 2 de Microsoft que se analizaron en la fase de teoría.

4.1. Herramientas de trabajo

En este bloque se describe todo lo necesario para una primera toma de contacto con Kinect con el fin de obtener la información necesaria para trabajar con ella. Comparando las librerías de desarrollo se opta por trabajar con el SDK de Microsoft, en su versión 2.0, por la amplia lista de herramientas y ejemplos que ofrece con documentación detallada. Al tratarse de una herramienta de Microsoft el código puede ser desarrollado en C++ o C#, y se debe trabajar con su entorno de programación Microsoft Visual Studio 2015

En el Anexo 1, se detalla la configuración del SDK 2.0 del dispositivo Kinect para poder trabajar con él en cualquier PC.

4.1.1. Preparación del entorno de desarrollo

Visual Studio es un entorno de programación de Microsoft que permite crear una gran variedad de aplicaciones. Está disponible en diferentes versiones desde la web de Microsoft, *Community*, *Enterprise* y *Code*, en todas ellas se puede editar y depurar código. En este proyecto se ha utilizado *Visual Studio Community* en su versión 2015. Este entorno incluye herramientas para el desarrollo de aplicaciones de escritorio de Windows como es el caso, pero también para *iOS* y *Android*.

Teniendo ya el entorno de programación, el siguiente paso fue decidir el lenguaje de programación C# frente a C++. En el segundo, la edición de la interfaz gráfica es más engorrosa, ya que se hace todo con comandos, sin embargo Visual C# crea un archivo XAML que controla el diseño de la interfaz simplemente arrastrando una serie de elementos predefinidos (botones, cuadros de texto, *check-box*...) y colocándolos donde convenga. Además se pueden editar desde código XAML estos elementos para conseguir la interacción de la interfaz con el fichero de la aplicación principal codificado en el propio lenguaje C#. Esta cómoda estructura permite una separación entre la parte lógica y la visual del programa.C# no deja de ser un lenguaje de programación orientado a objetos que utiliza en gran medida la sintaxis de C, y que evidentemente recoge muchos conceptos de lenguajes como JAVA.

Además muchos de los ejemplos que se proporcionan en el SDK browser que hacen referencia a la detección facial, y han servido de guía para el desarrollo de este Proyecto, están codificados en lenguaje C#.

Al abrir Visual Studio para crear un nuevo proyecto, el entorno C# ofrece dos posibilidades de crear un programa con entorno gráfico GUI, *Windows FormsApplicatio*n *WPF Application* (más novedoso). Los programas creados con *WinForms* estarán basados en las librerías estándar de Windows, esto implica que los controles que se utilicen van a tener el mismo aspecto que cualquier aplicación de Windows, mientras que WPF en general no depende de

ningún objeto estándar de Windows y además reacciona a estados como tener el ratón encima de un elemento de control, o haberlo pulsado sobre él. La principal diferencia es el rendimiento gráfico, mientras que *WinForms* emplea antiguos métodos de renderización, *WPF* tiene acceso a los aceleradores gráficos por hardware.

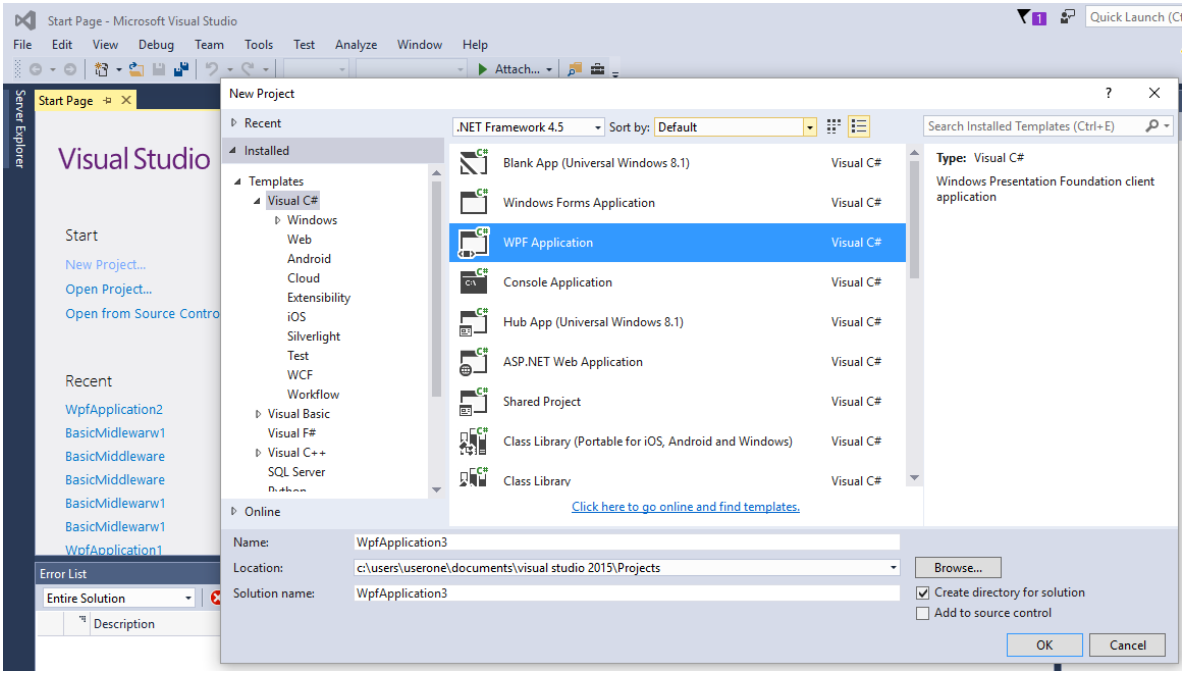


Figura 31. Creación de un nuevo proyecto *WPF Application*

Se selecciona WPF Application de C# y se pone nombre al proyecto, a continuación aparecen ya la ventanas de edición para trabajar que se han comentado (Figura 32 y 33).

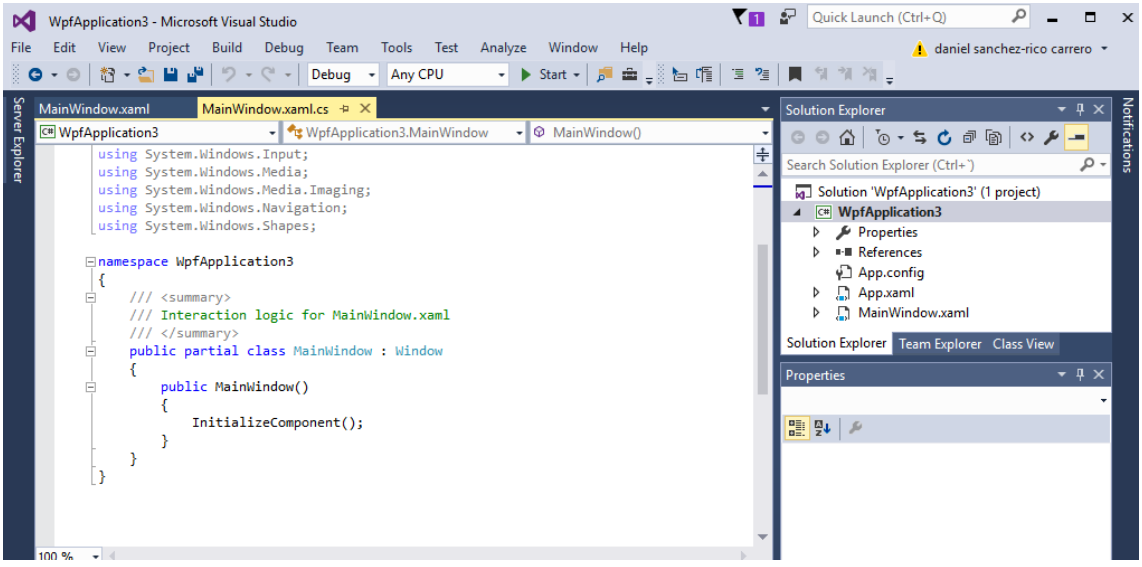


Figura 32. Edición programa principal en lenguaje C#

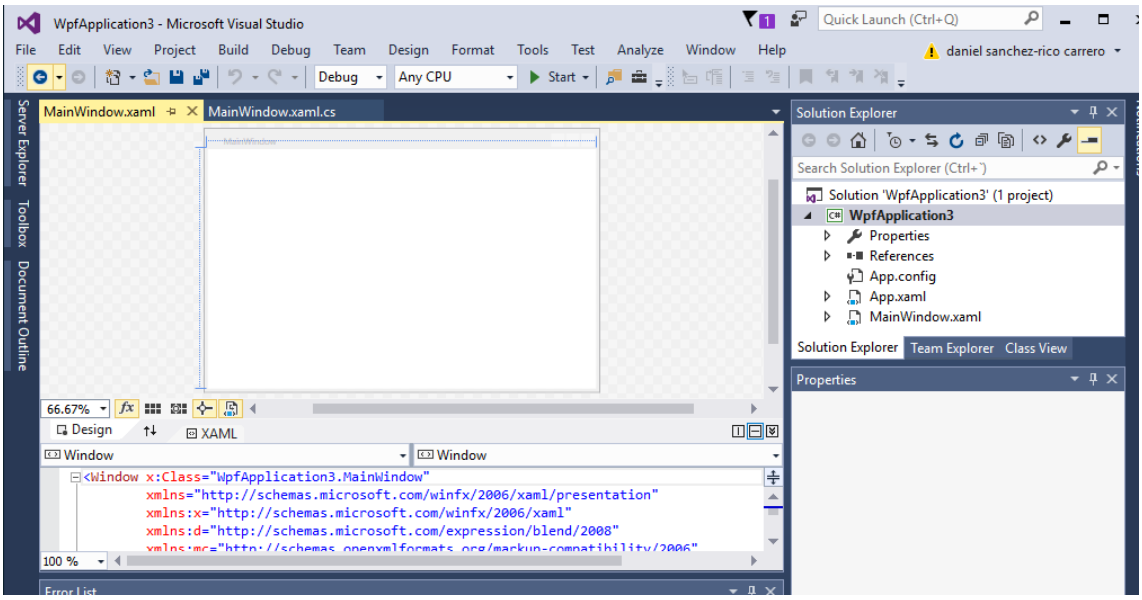


Figura 33. Edición interfaz XAML

Es necesario agregar la referencia SDK al proyecto, para poder hacer uso de sus librerías. Visual Studio facilita enormemente esta tarea a través del menú explorador de proyectos de la derecha. De igual manera se procedería para añadir cualquier otra referencia que se necesite (Figura 34).

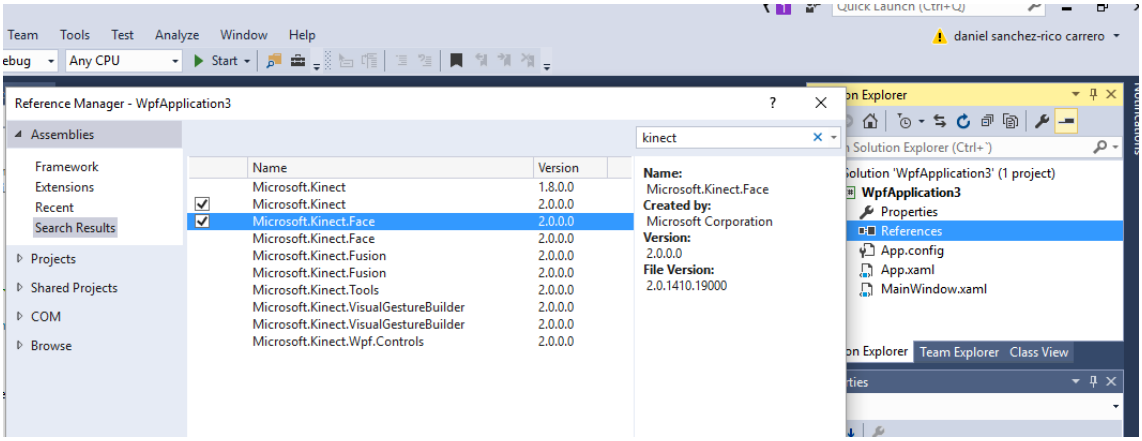


Figura 34.Agregar referencias al proyecto

4.1.2. SDK browser v2.0.

Esta herramienta permite al desarrollador probar diferentes aplicaciones que ejemplifican el uso de las distintas clases y métodos que ofrece la librería. Los ejemplos están organizados en varias secciones: *All*, C++, C#, *Tools* y *WinRT*, y en todos ellos se muestran distintas posibilidades de trabajar con *Camera*, *Depth*, *IR*, *Body* y *Face*. El espacio *Camera*, ofrece la retransmisión de la escena en RGB; *Depth* e *IR* es espacio de profundidad para trabajar en bajas condiciones de iluminación; *Body* y *Face*, muestran las distintas opciones del reconocimiento corporal y facial. Ofrece la posibilidad de analizar el código de los propios ejemplos con todo detalle, con lo que se extrae la información necesaria sobre el empleo de los métodos que el desarrollador tenga pensado utilizar.

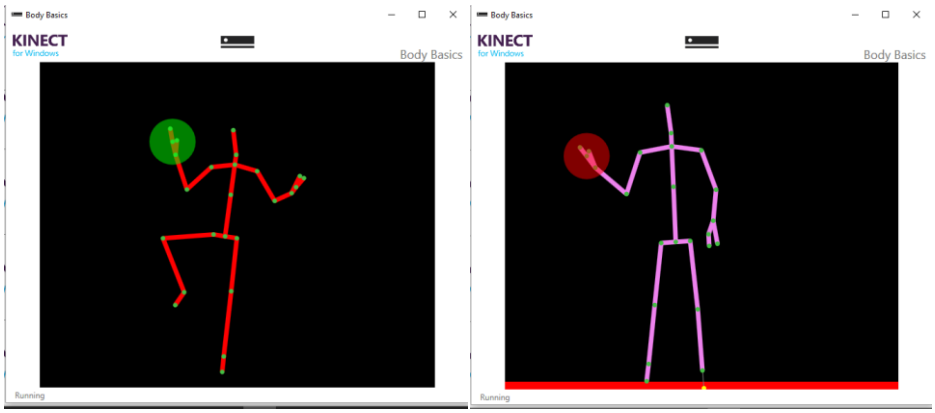


Figura 35. Ejemplo detección esqueleto de *Body Basics*

En la Figura 35 se muestra uno de estos ejemplos, *BodyBasics*, en el que se detecta y hace el seguimiento del esqueleto del jugador. Distingue si el puño está abierto o cerrado y si el esqueleto sale del campo de visión, lo indica con una línea roja en el borde implicado.

El ejemplo *FaceBasics* que se muestra en la Figura 36, proporciona algunas de las funcionalidades que se desean implementar en este Proyecto. Es la clase *FaceFrameResults* la que se emplea, ofreciendo la posibilidad de detectar diferentes emociones del rostro.

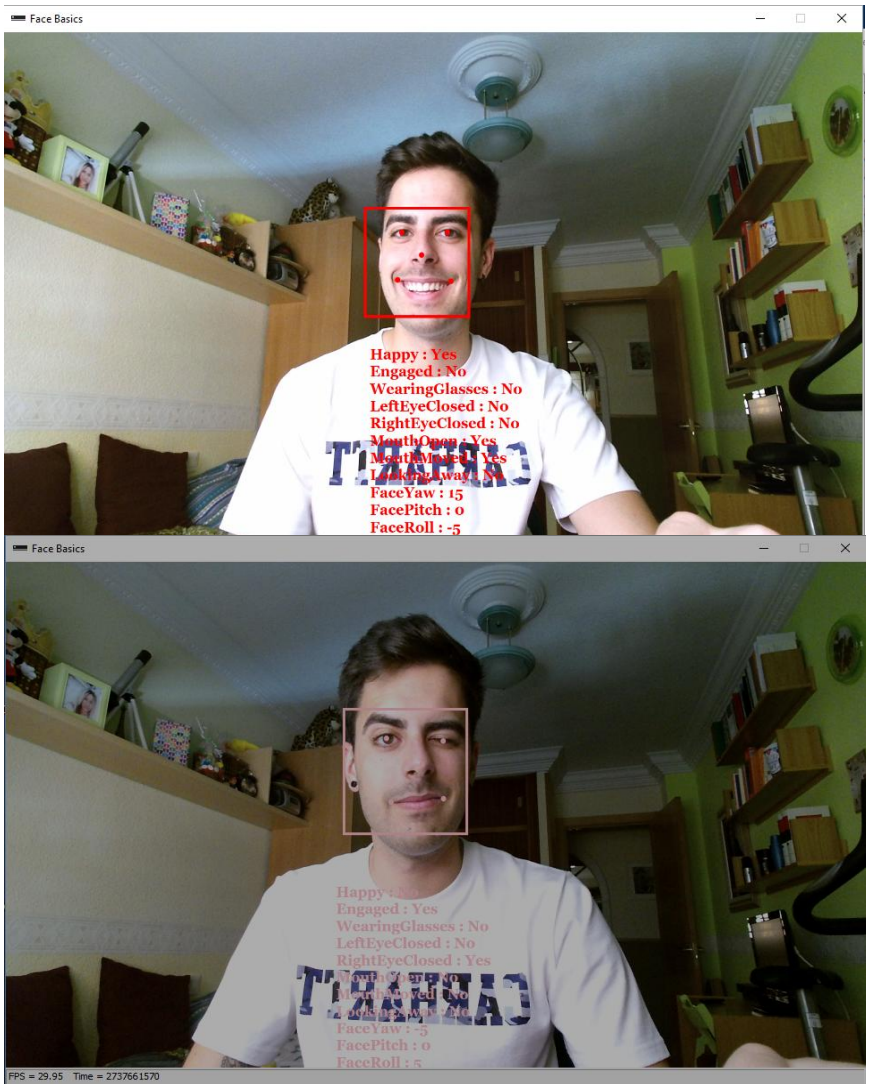


Figura 36. Ejemplo detección facial de *FaceBasics*

La utilización de la clase *FaceFrameResults*, obtiene además la posición de los cinco puntos principales de la cara (ojos, nariz y extremos laterales de la boca). Incluso es capaz de detectar la presencia de gafas o no en el jugador.

Otro de los ejemplos que han servido de referencia ha sido *HDFaceBasics*(Figura 37), que ofrece una representación de la cara mucho más realista en 3D. De la API *HD Face*, *FaceModel* es la clase empleada con la que se genera un modelo de rostro con más de 1300 puntos.

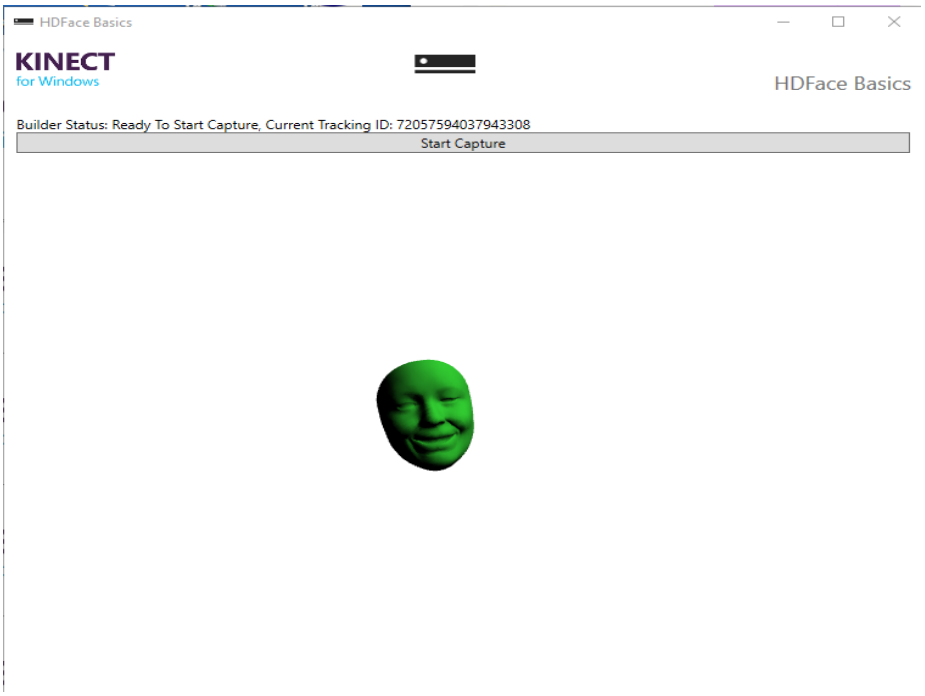


Figura 37. Ejemplo detección facial de *HDFaceBasics*

En la mayoría de las clases empleadas en estos ejemplos existe también el método que trabaja la detección en representación de profundidad en lugar de RGB (Figura 38).

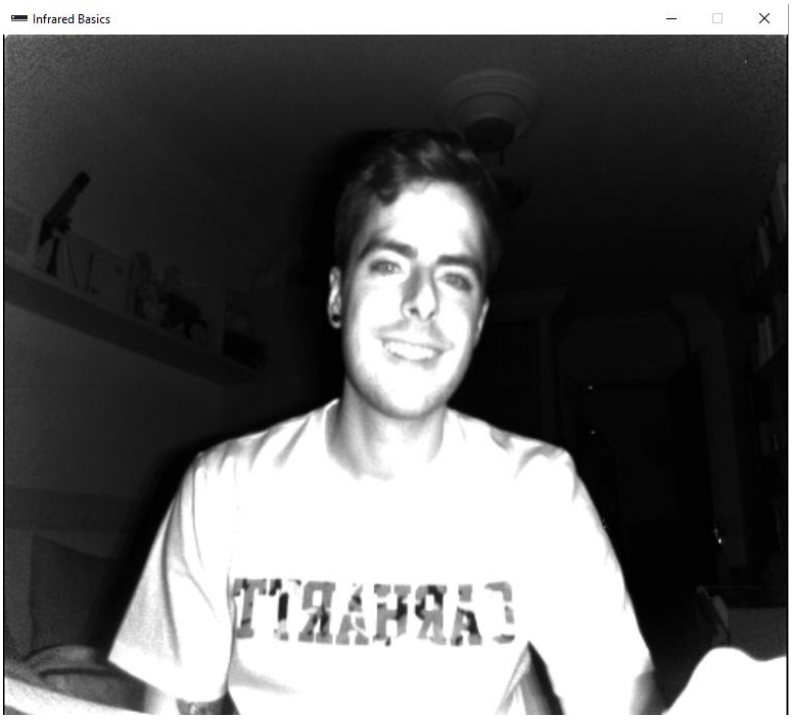


Figura 38. Ejemplo *Infrared Basics*

Esto posibilita que se puedan desarrollar aplicaciones que funcionen en condiciones de luz baja o incluso nula.

4.1.3. Python en Blender

La base constitutiva de Blender es el lenguaje Python, lo que permite que sea un programa personalizable en lo que respecta a su control. Se trata de un lenguaje de programación de sintaxis compacta y funcional orientado a objetos. Permite el acceso a sensores, actuadores y controladores de Blender para cambiarlos a conveniencia del desarrollador, acceder a las diferentes escenas y modificar todas las propiedades de los objetos pertenecientes a cada una de ellas, todo mediante código con extensión *.PY*.

La interfaz proporciona una ventana de edición de scripts en Python, "*Text Editor*", que permite proyectar directamente el código en la ventana *3D view* o incorporarlo a la lógica del juego. *BlenderGame Engine* es el motor de juegos de Blender, su funcionamiento se basa en asociar información a los objetos de la ventana 3D: variables, acciones, condiciones y operadores lógicos. Por lo que se pueden llamar a funciones de Python que estén incluidas en el proyecto *.blend*. Para que el script programado pueda operar en *Blender Game Engine* es necesario que trabaje como controlador en los bloques pertenecientes a la lógica del juego.

Para que se pueda acceder a las funciones correspondientes al *Blender Game Engine*, es necesario importar su correspondiente módulo, *BGE*. En la API de Blender se pueden encontrar todos los comandos que se requieran con ejemplos ilustrativos, siendo de gran ayuda para el desarrollador. [25]

4.2. Implementación del Middleware

Puesto que es la Kinect quien va a extraer la información de la detección del usuario periódicamente (a 30 fps), la aplicación desarrollada en Visual Studio con lenguaje C# va a hacer las veces de emisor, y el *addon* de Blender las de receptor.

La información que va a proporcionar la Kinect va a ser de dos tipos: posición espacial de los puntos de la cara y resultado de una serie de propiedades del rostro (ojos abiertos/cerrados, boca feliz/sorprendida).

Nada más iniciar el receptor de Blender, se envía un mensaje de configuración que contiene información de configuración (puertos que se usarán y presencia o no del jugador 2) a un puerto que se llamará *MiddlePort* para que el middleware se reajuste. A continuación cada dos cuadros del juego, el receptor de Blender vuelve a mandar un mensaje al *MiddlePort*, pero esta vez es de petición de datos.

Como respuesta, el *middleware* envía los últimos datos proporcionados por Kinect (que tenía almacenados en el *buffer*), al puerto *puertoPlayer1*. El *addon* de Blender lee de este puerto los datos y se completa la comunicación. A los dos cuadros del juego se vuelve a mandar otra petición de datos, y continúa el bucle de la comunicación hasta que emisor o receptor lo decidan.

El esquema de la Figuras 39 y 40 muestra el proceso de comunicación al completo.

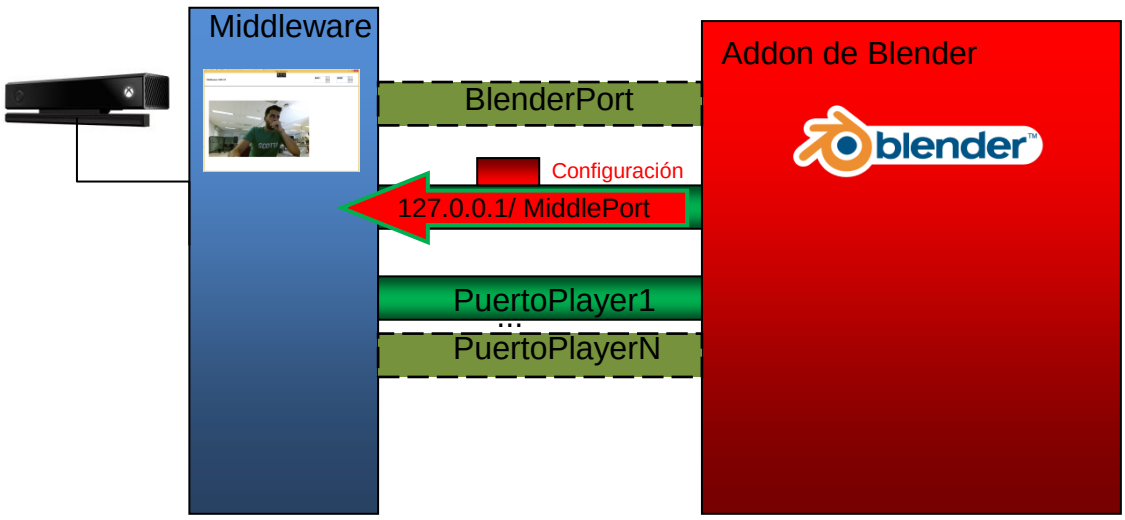


Figura 39. Intercambio de mensajes al iniciar el receptor en Blender

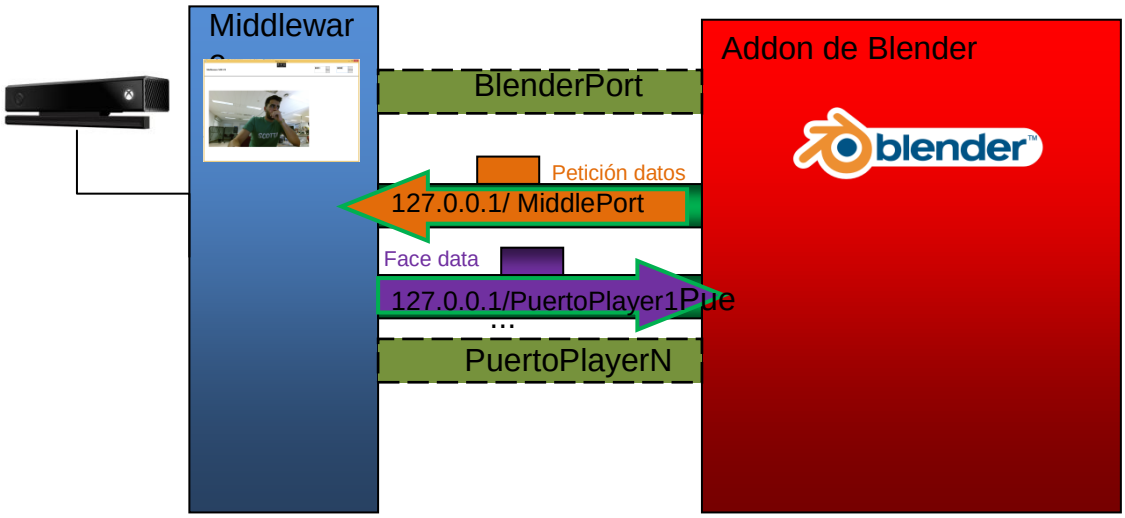


Figura 40. Intercambio de mensajes cada dos cuadros del juego

El puerto *BlenderPort*, que aparecía en la versión del *middleware* para Kinect 1, deja de utilizarse porque para la cara ya no es necesario enviar al *middleware* la información de la inclinación y la altura del sensor. Porque es interesante que la posición de la cara se sitúe en el centro del juego, no a la altura que tendría en la realidad. Se ha mantenido en la estructura para que en una futura versión en la que se transfieran los datos del esqueleto entero, pueda volver a utilizarse. Aunque se hayan puesto *N* puertos de jugadores, en este proyecto la comunicación solo es válida para un jugador.

4.2.1. Aplicación emisora.

Para trabajar en la obtención de los datos de Kinect se han de añadir las referencias Microsoft.Kinect y Microsoft.Kinect.Face, y a continuación los *namespace* correspondientes.

```
using Microsoft.Kinect;
using Microsoft.Kinect.Face;
```

Figura 41. Declaración del *namespace* que contiene los objetos relaciones con la detección

En esta primera versión del middleware se va a trabajar con la clase *faceFrameResults* que ofrece información en el espacio 2D. Se declaran todos los objetos necesarios para el reconocimiento del rostro:

- KinectSensor, es la instancia al sensor.
- BodyFrameReader, usado para identificar los cuerpos.
- Body array, que contiene la lista de personas detectadas.
- FaceFrameFeatures, para la detección de emociones.
- FaceFrameSource, la fuente marco de la cara.
- FaceFrameReader, el lector marco de la cara
- FaceFrameResult, lector de las características de la cara

Se necesitan los datos del cuerpo porque cada cara hace referencia a una instancia a un cuerpo, y refrescar en cada cuadro de imagen, por ello inicializar el lector de cara es algo más complicado. Primero se inicializa el lector del seguimiento del cuerpo.

El *FaceFrameSource* de la cara debe ser actualizada con el ID del cuerpo que le corresponda. Por lo que al actualizar el bodyFrame, se actualiza también el lector de la cara correspondiente, faceFrame.

Una vez que se ha conectado la cara al cuerpo, se puede especificar qué sucede cuando un nuevo frame está disponible. Los faceFrame funcionan exactamente igual que los frames de color, profundidad e infrarrojo, en primer lugar se obtiene una referencia al frame y posteriormente se adquiere la trama que contiene toda la información.

Por lo que simplemente basta con recuperar el frame resultado de la trama, y mostrar toda la información facial encapsulada en el faceFrameResult.

```
// Display the values

HappyResult = frameResult.FaceProperties[FaceProperty.Happy].ToString();
EngagedResult = frameResult.FaceProperties[FaceProperty.Engaged].ToString();
GlassesResult = frameResult.FaceProperties[FaceProperty.WearingGlasses].ToString();
LeftEyeClosed = frameResult.FaceProperties[FaceProperty.LeftEyeClosed].ToString();
RightEyeClosed = frameResult.FaceProperties[FaceProperty.RightEyeClosed].ToString();
MouthOpenResult = frameResult.FaceProperties[FaceProperty.MouthOpen].ToString();
MouthMovedResult = frameResult.FaceProperties[FaceProperty.MouthMoved].ToString();
LookingAwayResult = frameResult.FaceProperties[FaceProperty.LookingAway].ToString();
```

Figura 42. Obtención de las propiedades de la cara

También se obtienen los cinco puntos representativos de la cara que ofrece esta clase. Se decidió que la detección espacial la hiciera en espacio Infrarrojo y traducirla después a *CameraSpace*, así se conseguía una detección óptima incluso en malas condiciones lumínicas. Esta traducción se consigue por la utilización de la clase *CoordinateMapper* que se analizará más adelante.

A la hora de comunicar la cámara con la interfaz gráfica se vuelve a trabajar con cuadros de imagen de una manera similar, sólo que se tiene que hacer uso de un objeto *WritableBitmap* para conectar los cuadros que salen de la Kinect a la interfaz de la ventana XAML.

El diagrama de flujo del proceso de detección queda reflejado en la Figura 43.

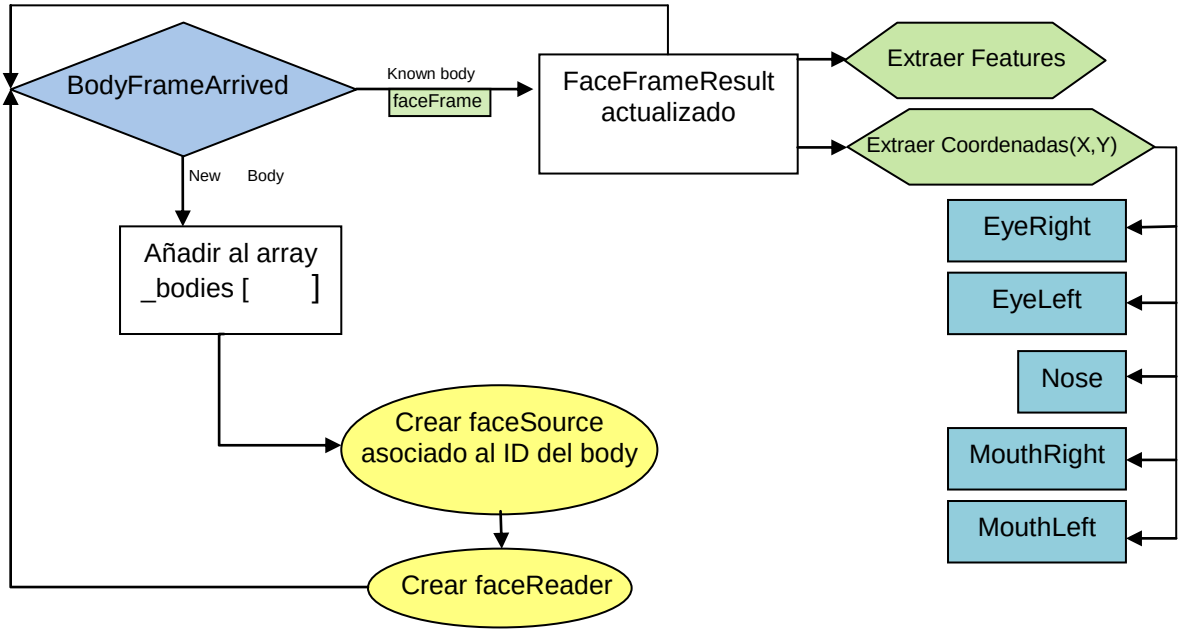


Figura 43. Diagrama de flujo de la detección

Para llevar a la comunicación aplicando el protocolo OSC es necesario añadir la referencia *VentuzOSC* y su *namespace*. Además C# por medio de la librería *System.Net* provee una clase *Socket* para implementar la conexión desde el lado del servidor.

Los *socket* son un mecanismo que nos permite establecer un enlace entre dos programas que se ejecutan independientemente el uno del otro (generalmente un programa cliente y otro servidor). Se deben conocer sus direcciones IP y el puerto por el cual se comunicarán, es bueno elegir puertos en el rango de 1024 a 65535. Para esta comunicación se han escogido el puerto 8000 para el envío de la información de configuración y el 8001 para el envío de los datos.

En la función IniciarLector( ), se crean los objetos de la clase UdpWriter de la librería *VentuzOSC*, que van a ser los encargados de escribir en los puertos para enviar los datos. También se abre el puerto de escucha (8000).

El puerto de escucha se mantiene a la espera hasta recibir un mensaje de 17 bytes, para lanzar un evento del tipo RecieveCallBack( ). Cuando esto sucede, se lee la cabecera del mensaje (primeros 4 bytes), que puede contener dos tipos de información:

- 1234567890, petición de comunicación con los datos de configuración.
- 876543210, petición de envío de datos.

En ambos casos el manejador debe reiniciar la escucha, para el siguiente mensaje que llegue.

Para el primer caso se llama a la función ReadSettings( ), que va a extraer del mensaje recibido los puertos que van a intervenir en la conexión y otros datos de configuración.

En el segundo caso se llama a la función SendAll( ), para enviar la información correspondiente a la detección de la cara. Antes de realizar el envío, en lo que refiere a las coordenadas de los puntos faciales, se traducen de *DepthSpace* a *CameraSpace*. Esta conversión es necesaria porque el espacio de profundidad está expresado en pixeles, mientras que el espacio de cámara se mide en metros, que es más conveniente para trabajar en la ventana 3D de Blender.

Aquí es donde entra en juego la clase *CoordinateMapper*, que con el método *MapDepthPointToCameraSpace* permite obtener las coordenadas en metros. [26]

CameraSpace hace referencia al sistema de coordenadas 3D usado por Kinect (Figura 44), este sistema toma como origen de coordenadas el centro del sensor IR, con las X positivas hacia la derecha del sensor, las Y positivas hacia arriba y las Z saliendo perpendiculares al plano del Kinect.

Figura 44. Sistema de coordenadas *CameraSpace*

DepthSpace es el sistema empleado para expresar coordenadas 2D en la escena de profundidad. La imagen está definida por la división de la misma en filas y columnas, donde las x son columnas y las y son filas, dando lugar cada posición fila/columna a un pixel. Por lo que X=0 e y=0, corresponde a la esquina superior izquierda, mientras que x=511 e y=423, hace referencia a la esquina inferior derecha, como se muestra en la Figura 45.

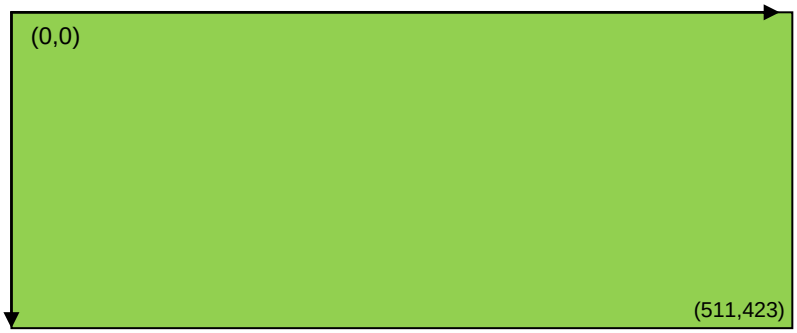


Figura 45. Sistema de coordenadas *DepthSpace*

Una vez se tienen las coordenadas en las unidades deseada se procede al empaquetado y envío del mensaje. Todos los mensajes de coordenadas van a tener la etiqueta "/face/"+"Nombre del punto facial" de manera que se pueda distinguir la información a enviar. Un mensaje de envío de coordenadas sería el mostrado en la Figura 46.

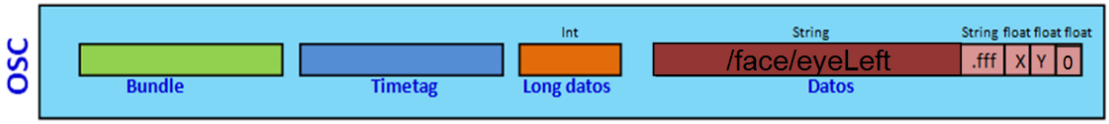


Figura 46. Etiquetado del mensaje de datos de coordenadas enviado por el *middleware*

La cadena *string* ".fff" indica que a continuación vienen tres datos del tipo *float*, el tercero representa la coordenada z, que como la función de detección 2D empleada no nos la proporciona, se pone el valor 0.

Como se vio en la sección 3.1.2.1, los resultados de las propiedades de la cara analizadas podían ser YES, NO, MAYBE y UNKNOWN. Teniendo esto en cuenta y para simplificar el envío de los datos, se optó por codificar el resultado YES como un 1, y los demás como 0. De modo que si se ha detectado que se tiene la boca abierta, se envía un 1 entero en lugar de una cadena *string*. Estos mensajes tendrán la etiqueta "/face/"+"Propiedad de la cara detectada", y serán de la forma de la Figura 47. Cuatro han sido las propiedades de la cara que se han

considerado interesantes implementar en este proyecto han sido: ojo derecho o izquierdo abierto/cerrado, boca abierta/cerrada y sonrisa.

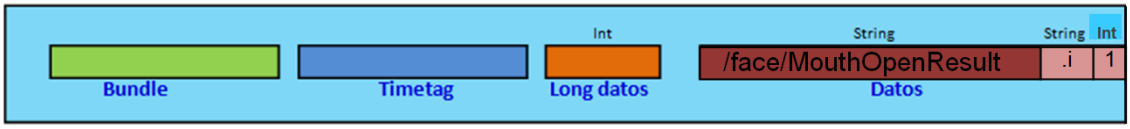


Figura 47. Etiquetado del mensaje de datos de propiedades enviado por el *middleware*

La interfaz de la aplicación tiene el aspecto de cualquier ventana de Windows y es bastante sencilla. Una cabecera con el nombre del proyecto contiene los botones para introducir el número de los puertos que van a intervenir en la comunicación, basta con hacer *click* con el botón izquierdo del ratón sobre el cuadro de texto, teclear el valor y pulsar *intro*. Si se desea limpiar el cuadro de texto, sólo hay que hacer doble clic con el botón izquierdo del ratón sobre el mismo.

Un cuadro de imagen situado en la parte central de la ventana reproduce la salida de la cámara (Figura 48).

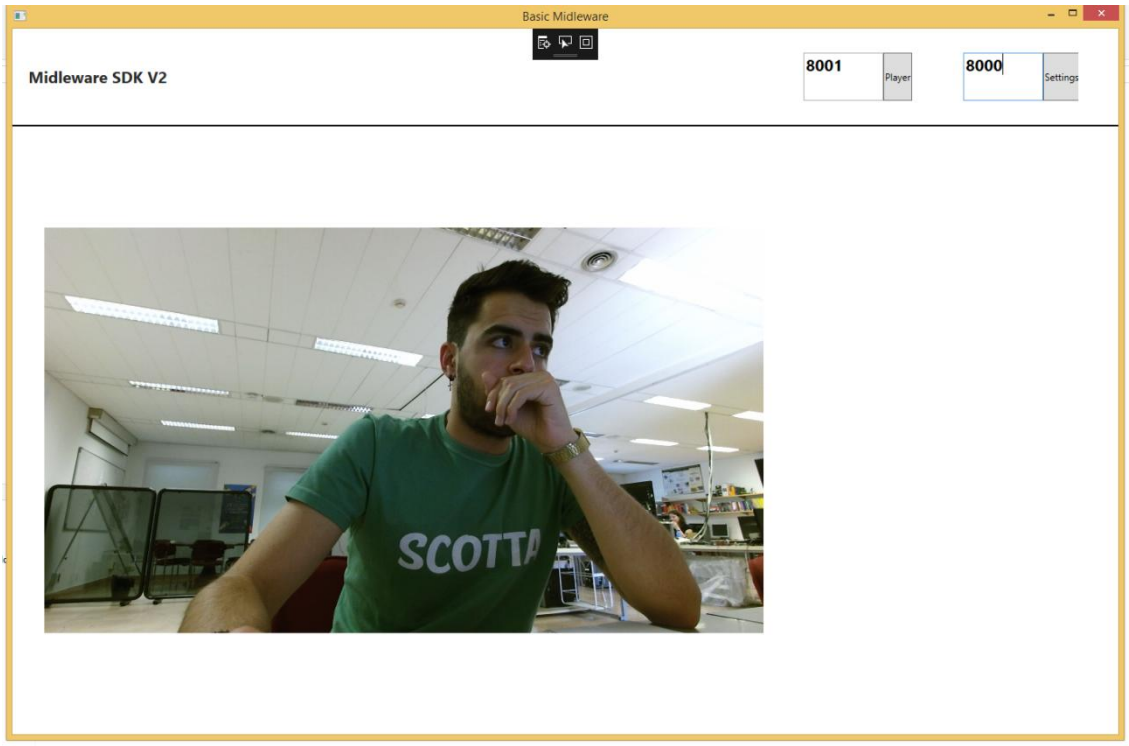


Figura 48. Interfaz de la aplicación *middleware*

4.2.2. Addon receptor de Blender.

El propósito de este apartado es la creación de un *addon* para Blender que actúe de receptor en la comunicación. Un *addon* es un *plugin* que se instala en el fichero .blend para conseguir una funcionalidad más allá de lo que proporciona el propio Blender.

Blender ofrece*addons* ya instalados listos para usar, pero el usuario también tiene la posibilidad de crearse los suyos propios en lenguaje Python.

Se pretende crear una herramienta más incorporada a la interfaz de Blender, un receptor llamado "KinectFace" que se introduce en el entorno de desarrollo y representa la cara del jugador. El *addon* tendrá una arquitectura modular para que se permitan introducir nuevas funcionalidades en versiones futuras.

La herramienta de cara a la interfaz tendrá aspecto de botones, con los cuales se selecciona añadir el receptor "KinectFace", también se puede modificar el valor de los puertos que van a intervenir en la comunicación y añadir más caras (Figura 49).

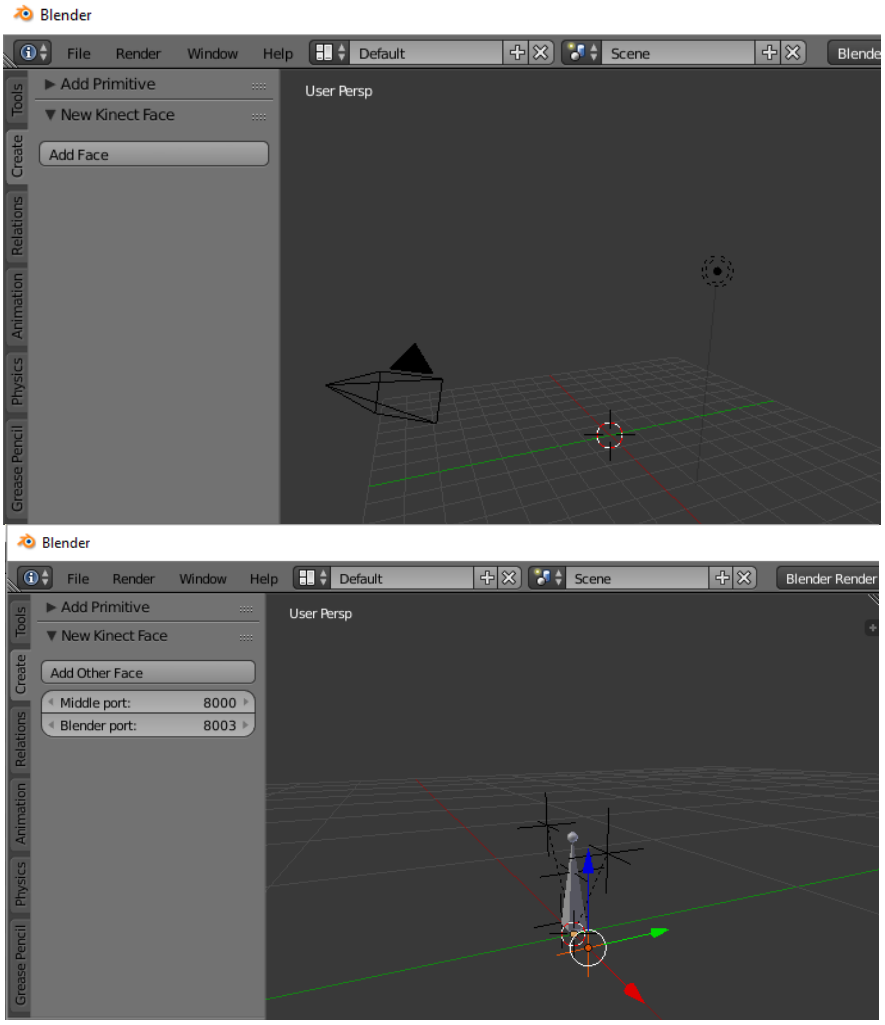


Figura 49. Apariencia de la herramienta en la interfaz de Blender

Cinco objetos "*empty*" son los puntos que reproducirán las coordenadas recibidas de cada uno de los puntos de la cara. Este tipo de objetos no aparecen en la imagen final producida por Blender aunque siguen ahí. Se ha optado por estos objetos porque para juegos en los que la lógica sea cualquier expresión facial (guiñar ojo derecho y gira a la derecha, etc..), no es necesario que aparezca el rostro en pantalla. Y por ejemplo para juegos en los que sí se requiera ver la reproducción del rostro, basta con emparentar cualquier malla, por ejemplo una esfera, al objeto *empty*.

Cada uno de estos *empty* se nombrará con el nombre del punto que se ha pasado en la etiqueta del mensaje OSC y el número del sensor KinectFace al que pertenecen (Eyeleft1, EyeRight1, mouthLeft1, mouthRight1, nose1).

El *addon* asigna automáticamente a cada una restricción de posición que les obliga a seguir en todo momento el movimiento en el rostro del jugador. Para facilitar el manejo del conjunto de elementos que constituyen la cara, se crea un objeto "armature" con un sólo hueso (HipCenterRot), al que se emparentan todos los *empty* creados.

A través de la función addGameLogic( ) se añaden todas las propiedades al receptor KinectFace:

- Open. Indica si el puerto de escucha del receptor está abierto.
- Player. Indica a qué jugador pertenece el receptor.
- Port. Indica el puerto en el que el receptor debe escuchar los datos.
- Receiving. Indica si se están recibiendo datos.
- MouthOpenResult, LeftEyeClosed, RightEyeClosed y HappyResult. Son enteros que indican si se cumple o no la expresión y que se van modificando según la información recibida.

Los sensores:

- Start. Se activa cuando se lanza el GameEngine.
- Pulse. Genera un pulso periódico mientras la propiedad Open vale True. Se configura para que se genere a 30 fps, la misma velocidad que Kinect.

Tres controladores del tipo "python" que van a ejecutar funciones del propio *addon* por defecto:

- StartSocket. Ejecuta la función GESocket, que abre el puerto de escucha del receptor.
- Receive. Ejecuta la función GEReceive, función de recepción de los datos.
- Settings. Ejecuta la función GESettings para el envío de los datos de configuración.

Y por último un actuador "Armature" conectado al sensor *start* para que se registren los movimientos al iniciar *GameEngine*.

Para enviar el mensaje con los datos de configuración al middleware se emplea la función SendSettings ( ), en la que se codifican en bytes los datos a enviar y se empaqueta el mensaje tal y como muestra la Figura 50.

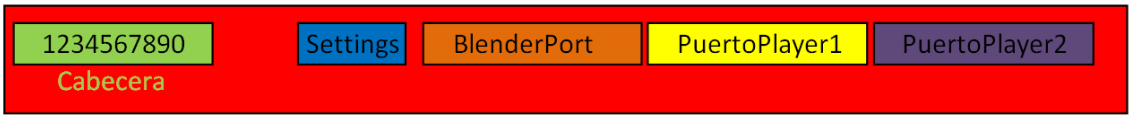


Figura 50. Mensaje de configuración

En total 17 bytes, 4 bytes de cabecera, 1 byte de Settings, y 4 bytes por cada puerto.

La función GEReceive( ) se encarga de enviar el mensaje de petición de datos que consta de una cabecera de 4 bytes y un cuerpo de 13 bytes de ceros, para que la longitud sea la que espera el middleware.

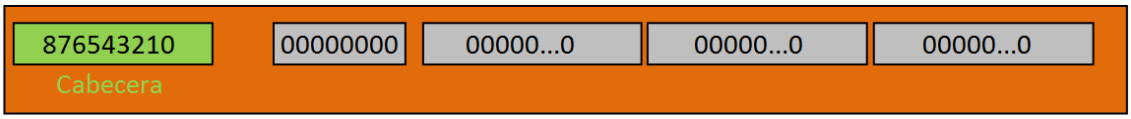


Figura 51. Mensaje de petición de datos

Al recibir la información del middleware, se van leyendo por orden de llegada los mensajes OSC. Según el dato recibido se procede de una forma:

- Tres *floats*. Se trata de las coordenadas de un punto, se toman y se asignan al *empty* correspondiente teniendo en cuenta la etiqueta del mensaje.

El orden del sistema de coordenada de Kinect y Blender es diferente por lo que se tuvo que modificar el posicionamiento de las coordenadas (lo que Blender llama z, Kinect lo llama y).

- Un entero. Se trata de la información de una de las propiedades de la cara, por lo que se cambia el valor de la propiedad del KinectFace que corresponda con la etiqueta del mensaje.

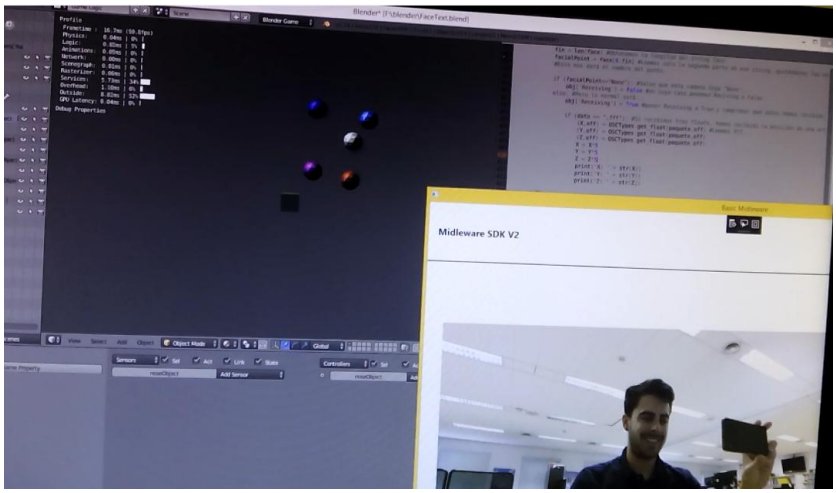


Figura 52. Detección de los cinco puntos faciales

En la Figura 52 se puede observar el posicionamiento de los *emptys* al arrancar la aplicación. Se emparentó una espera a cada *empty* para que se pudiera seguir el movimiento.

4.3. Implementación de videojuegos basados en el reconocimiento facial

En esta sección se desarrollan juegos sencillos para Blender basados en las distintas posibilidades de reconocimiento facial que ofrece este middleware.

Se comprobó que con el movimiento de los cinco puntos de la cara que se ofrecía, daba pocas posibilidades de jugabilidad, por lo que el trabajo se centró en las propiedades de reconocimiento facial (sonrisa, boca abierta, ojo derecho cerrado y ojo izquierdo cerrado). Según el valor que tomen estas propiedades del receptor KinectFace se construye una lógica que da lugar a movimientos en el juego, de modo que sólo con la cara se puedan realizar movimientos dentro del juego. Como consecuencia se han obtenido los tres juegos que se detallan a continuación.

4.3.1. SeaAdventure

La escena está ambientada en el mar con tablones de madera flotantes y monedas que el jugador tiene que ir cogiendo a lo largo del juego.

Una barca de color verde haces las veces de avatar, que siempre avanza y según guiñes el ojo derecho o el izquierdo, gira hacia esas direcciones respectivamente. Al recoger todas las monedas se pasa a la pantalla de fin de juego (Figura 53).

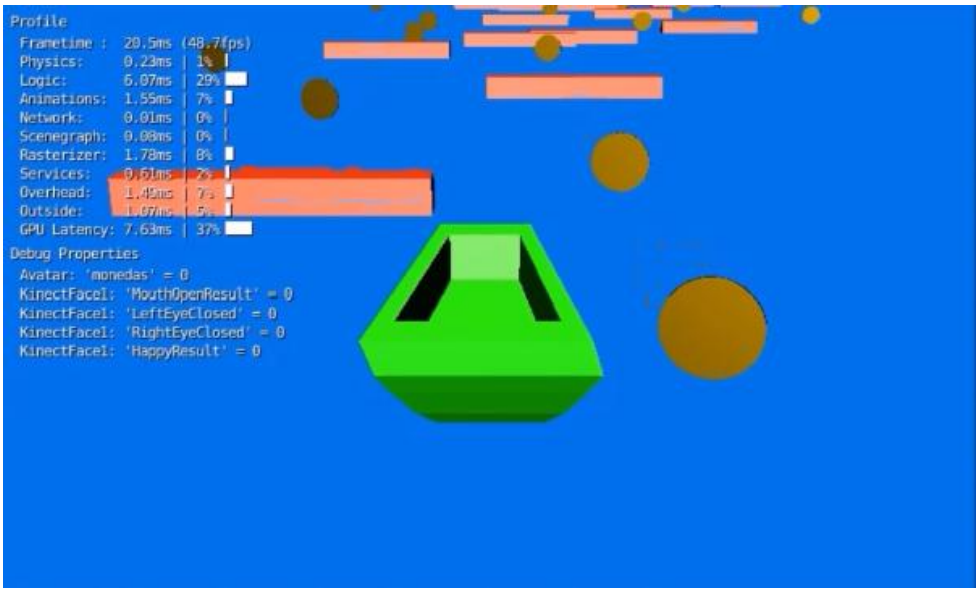


Figura 53. Escena del juego SeaAdventure

La lógica empleada en este juego es muy sencilla, está basada en las propiedades RightEyeClosed y LeftEyeClosed del receptor KinectFace. Cuando se activan (toman el valor 1), mediante un actuador "message" se manda un mensaje al avatar para que modifique la trayectoria de la barca. El avatar es el objeto que controla el movimiento de la barca y que se mueve con ella por toda la escena.

Cuando el sensor "message" del avatar lo recibe, se activa el actuador "motion" al que está conectado, realizando un giro de 2° a la izquierda o a la derecha según corresponda.

La lógica de recolectar las monedas, música de fondo y pantalla final no se han incluido en las capturas de pantalla de las Figuras 54 y 55 para hacer más visible el procesado que se hace de la información recibida por el *middleware*.

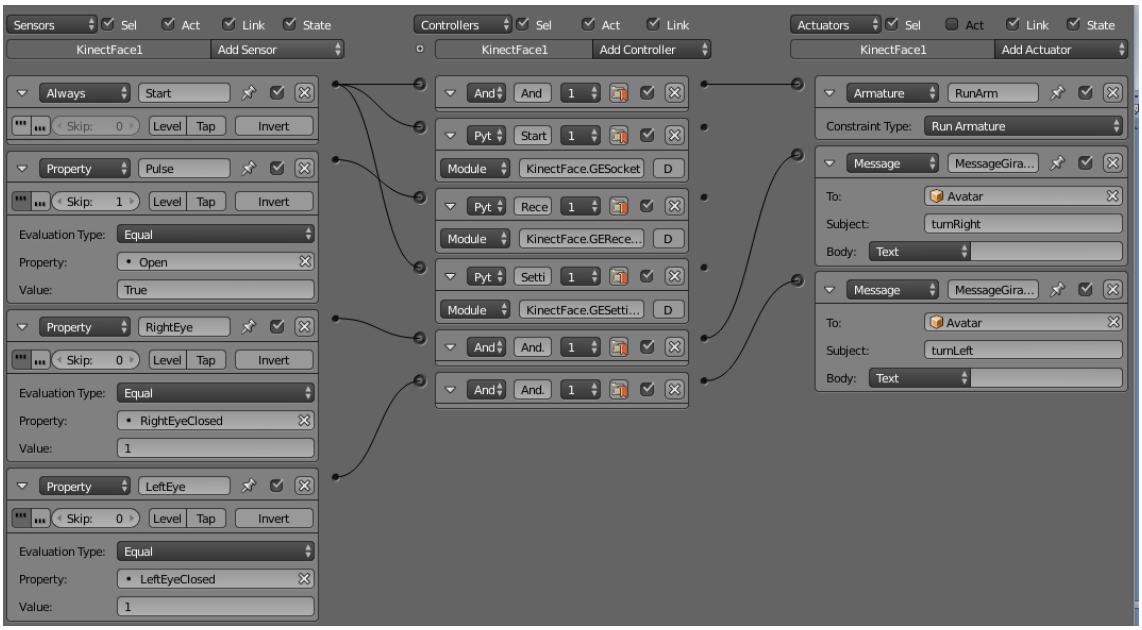


Figura 54. *Logic Editor* del receptor KinectFace para el juego SeaAdventure

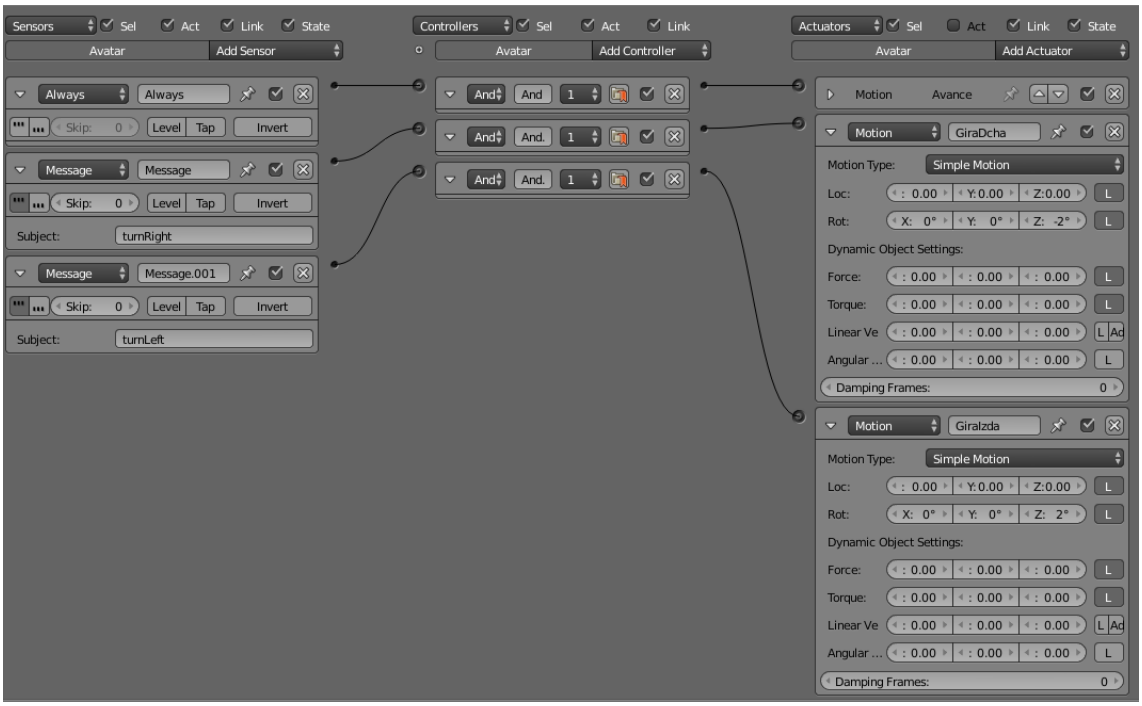


Figura 55. *Logic Editor* del avatar para el juego SeaAdventure

4.3.2. Troncos Locos

En este juego el avatar tiene que avanzar por una rampa en la que van cayendo troncos, el objetivo es ir saltándolos hasta llegar a la línea de meta. Los saltos se producirán sólo cuando el jugador abra la boca, para lo cual vamos a hacer uso de la propiedad MouthOpenResult del KinectFace. El avatar se materializa en un rectángulo verde y un contador en la parte superior muestra la distancia que queda hasta la meta (Figura 56).

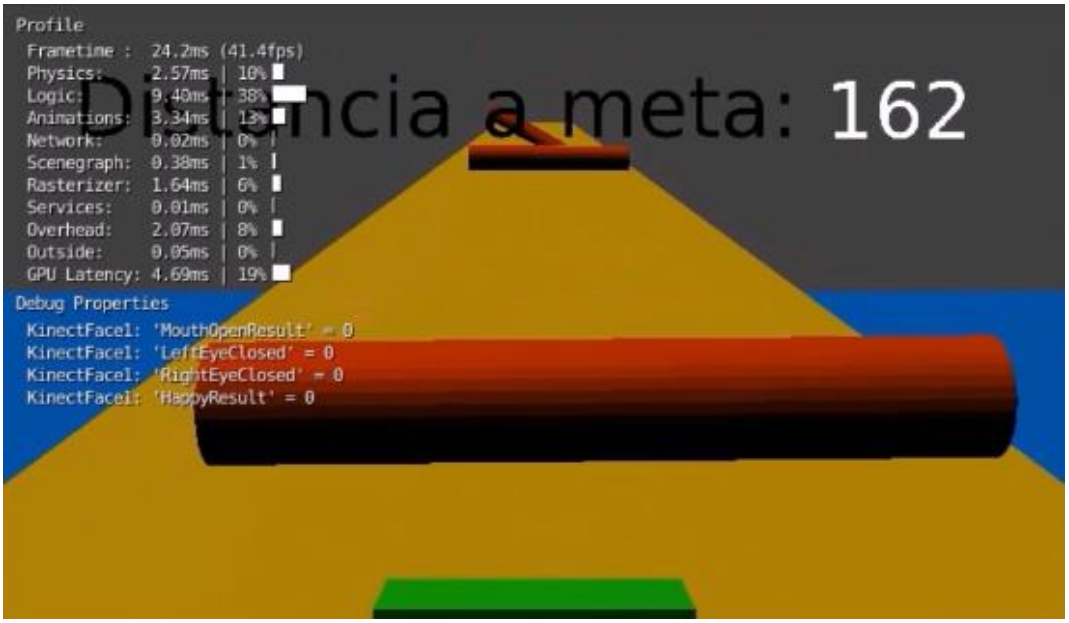


Figura 56. Escena del juegoTroncos Locos

Como se ha dicho, la lógica del salto está basada en la propiedad MouthOpenResult, cuando se activa (tiene valor 1), se mandan un mensaje al avatar de nuevo con un controlador

"message". Se usa este tipo de actuador porque simplifica la conexión lógica entre diferentes objetos de la escena.

Cuando el avatar tiene un sensor tipo "allways" que hace que siempre avance ascendiendo la cuesta, por lo que al recibir el mensaje, éste activa un actuador del tipo "motion" que hace al rectángulo elevarse en el eje Z y sortear los troncos.

Cuando el avatar ha llegado a una distancia predeterminada, la meta, se pasa a la pantalla de fin de juego. El contador que fija la distancia hasta la meta se ha realizado mediante una función en fichero python sencillo previamente desarrollado.

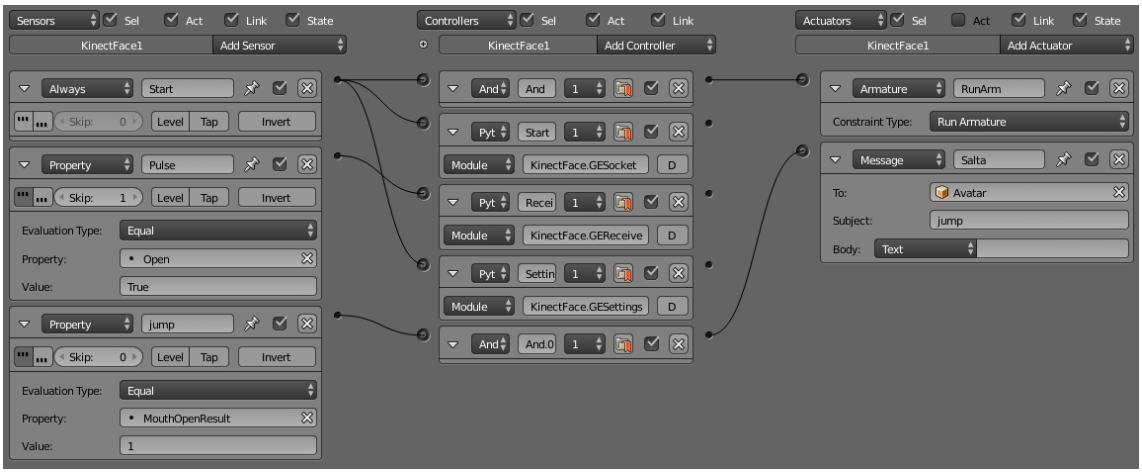


Figura 57. *Logic Editor* del receptor KinectFace del juego Troncos Locos

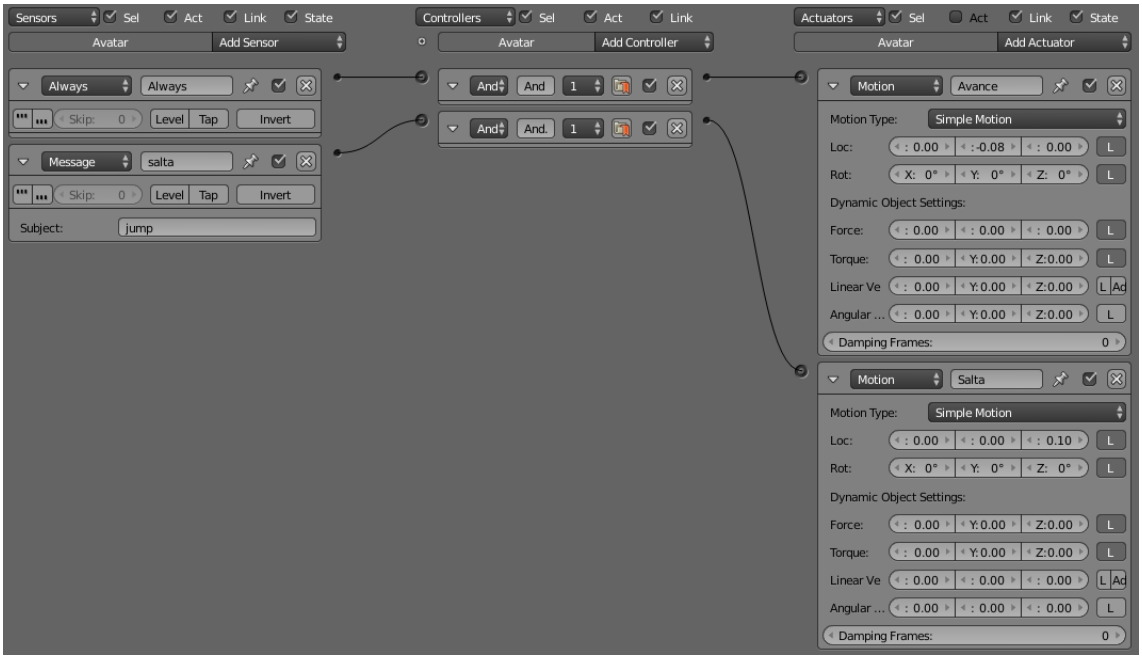


Figura 58. *Logic Editor* del avatar del juego Troncos Locos

4.3.3. Smile

El objetivo de este juego es que un *smile* (carita amarilla de los emoticonos típicos de aplicaciones de chat), imite los gestos del jugador a tiempo real. Es capaz de guiñar los ojos, poner boca de sorpresa o de sonrisa, para ello se van a utilizar todas las propiedades faciales que el middleware ofrece.

En la lógica de este juego intervienen diferentes escenas con mallas de *smiles* de todas las combinaciones posibles de las expresiones faciales comentadas, se muestran en la Figura 59.

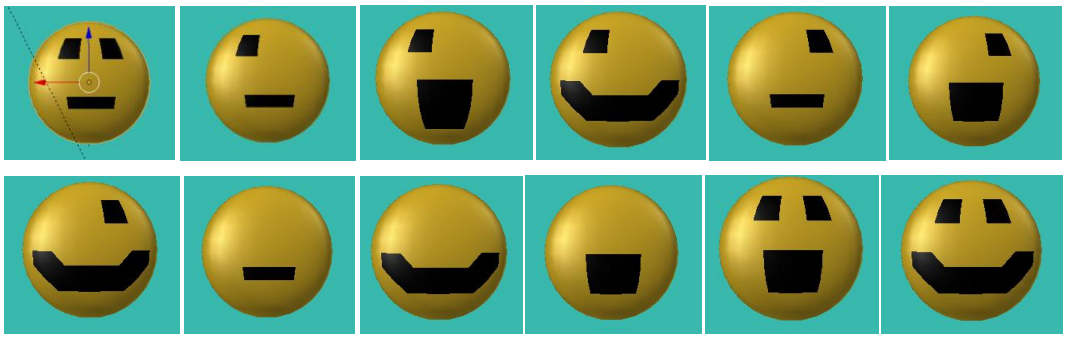


Figura 59. Diferentes escenas que configuran el juego*Smile*

Según la combinación de los valores de cada una de las propiedades faciales (RightEyeClosed, MouthOpenResult, LeftEyeClosed, HappyResult), se mandará un mensaje al *smile* de la escena principal, el que corresponde a la cara en reposo.

Dependiendo del mensaje recibido, se activará un actuador "Edit Object" que reemplazará la malla del *smile* en reposo por la que corresponda.

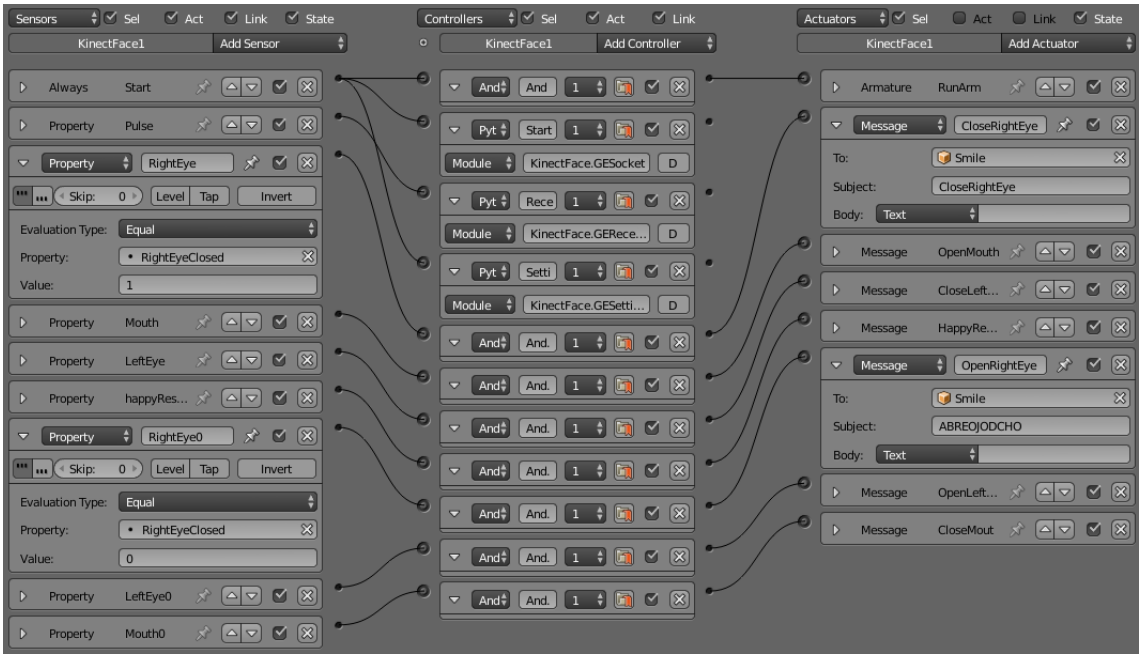


Figura 60. *Logic Editor* del receptor KinectFace del juego *Smile*

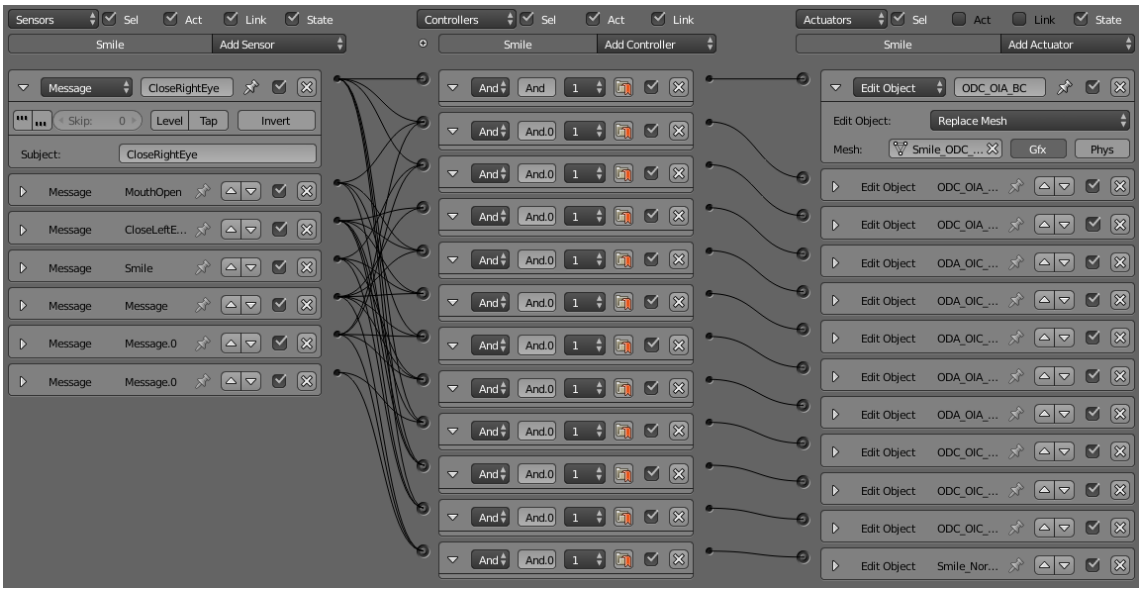


Figura 61. *Logic Editor* del smile en reposo del juego *Smile*

4.4. Implementación del middleware 3D

A la hora de llevar a cabo el desarrollo de los juegos se vio que el *middleware* implementado con la clase faceFrameResult en espacio 2D limitaba el modelado de la cara a cinco puntos. Se necesita una mayor resolución para poder detectar otro tipo de expresiones faciales más interesantes a la hora de ejercitar músculos del rostro.

Analizando el SDK 2.0 se descubre la API HD Face, en la cual con la clase faceModel, en C# y C++, se pueden obtener hasta 1347 puntos que configuran una máscara del rostro. Además el High Definition Face Tracking permite obtener 17 AUs que se pueden emplear para detectar otras emociones como si la ceja está levantada o bajada, labios o mejillas, pero su sintaxis es de C++ y en este proyecto no se ha llegado a analizar. Por lo que en este apartado se ha trabajado para desarrollar un middleware que implemente la clase faceModel.

4.4.1. Aplicación Emisora

El desarrollo de la aplicación emisora va a ser muy similar a la versión anterior, por lo que sólo se van a detallar las diferencias significativas. Nuevamente es imprescindible importar al proyecto las librerías Microsoft.Kinect y Microsoft.Kinect.Face, y declarar los objetos necesarios para el reconocimiento facial.

- KinectSensor, es la instancia al sensor.
- BodyFrameSource, adquiere los datos de un cuerpo.
- BodyFrameReader, actualiza los datos del cuerpo cada cuadro imagen.
- HighDefinitionFaceFrameSource, adquiere los datos HD de una cara.
- HighDefinitionFaceFrameReader, actualiza los datos HD de la cara cada cuadro.
- FaceAligment, se requiere para acceder a los vértices de la cara.
- FaceModel, permite obtener una máscara de la cara.

Como se puede comprobar y ya sucedía en la versión del middleware que empleaba la función 2D, se necesita una fuente/lector de cuerpo a la que corresponda cada cara. No se puede hacer el seguimiento de una cara sin hacer el seguimiento de su cuerpo en primer lugar, por lo que el HDFaceFrameSource de la cara debe ser actualizada con el ID del cuerpo que le corresponda.

A la hora de obtener la información facial, se toma la actualización del FaceModel en cada cuadro que llega del sensor. Hay que tener en cuenta que los puntos faciales van a venir dados en un *array* de vértices (cada uno de ellos con coordenadas X,Y y Z), que describen la esquina de un triángulo geométrico. En aras de la simplicidad, se van a dibujar solamente los vértices.

Mediante la función CalculateVerticesForAligment( ) se obtiene este *array* de vértices, del cual simplemente se extraen las coordenadas de cada uno y se empaquetan con la correspondiente etiqueta en el mensaje a enviar.

```
for (int index = 0; index < vertices.Count; index++)
{
    CameraSpacePoint vertice = vertices[index];

    var lista1 = new System.Collections.Generic.List<OscElement>();
    lista1.Add(new OscElement("/face/" + "Point" + index.ToString(),
                                                        (float)vertice.X,
                                                        (float)vertice.Y,
                                                        (float)vertice.Z));
    osc[1].Send(new OscBundle(DateTime.Now, lista1.ToArray()));
```

Figura 62. Envío de las coordenadas de los vértices faciales

En lo que refiere a la interfaz gráfica, se ha decido simplificarlo al máximo posible y el cuadro de imagen en lugar de estar en espacio de color, es un fondo blanco por el que se desplazarán los puntos faciales detectados (Figura 63).

El bloque superior tiene la misma apariencia, muestra los botones que funcionan de igual forma para introducir el número de los puertos que intervienen en la comunicación.

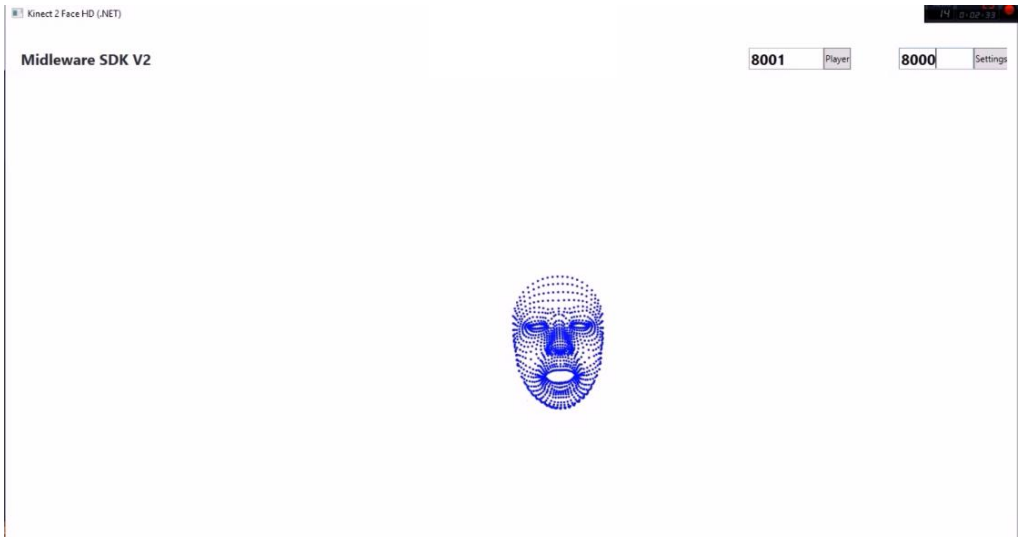


Figura 63. Interfaz middleware versión 3D

El proceso de comunicación entre el middleware y el *addon* de Blender es exactamente el mismo. Blender envía mensaje de configuración al middleware, acto seguido envía petición de datos y el *middleware* le responde con un mensaje que contiene las coordenadas.

4.4.2. Addon receptor de Blender

El código python del *addon* es exactamente igual al de la versión anterior salvo tres cosas.

- En las propiedades del receptor KinectFace no aparecen las expresiones detectadas de MouthOpenResult, LeftEyeClosed, RightEyeClosed ni HappyResult.
- Los 1347 *emptys* que en este caso referenciarán los vértices del rostro, no se les da una posición predeterminada antes de arrancar el *Game Engine*. Sino que aparecerán en el origen de coordenadas.

- El número de recepciones se incrementa notablemente, de 9 (5 puntos y 4 propiedades) a 1347.

También se emparenta una esfera a cada *empty* con un material determinado para que al arrancar el juego se visualicen todos los puntos.

Al ser tan elevado el número de objetos *emptys* que se importan cuando se añade el receptor KinectFace a la ventana *3D view*, tarda varios segundos hasta que se materializa.

Otro problema que ha surgido al incrementar de tal manera el número de recepciones ha hecho que se ralentice la tasa de transmisión de cuadros del juego, por lo que hay un pequeño *delay* entre la máscara de puntos mostrada en la interfaz del middleware y la del juego en Blender.

4.4.3. FaceTest3DDD

Es el juego que se ha desarrollado para testear el funcionamiento de esta nueva versión de middleware. La lógica es extremadamente básica, simplemente refleja el movimiento de los puntos del rostro por la pantalla como si de un espejo se tratase. El cubo verde que aparece, simula el origen de coordenadas de la escena, se decidió incluirlo para que el jugador se sitúe en la misma (Figura 64).

La escena quedó con apariencia tétrica, así que se pensó en añadir un hilo musical que la ambientara, para dar algo de vida al juego.

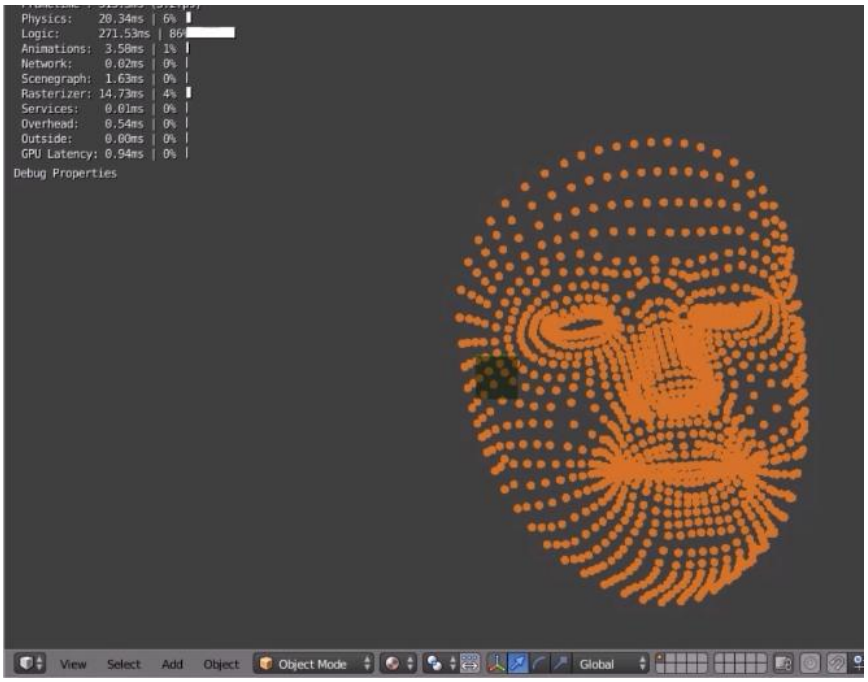


Figura 64. Escena del juego FaceTest3DDD

5. Aplicación real del proyecto

La posibilidad de crear una interfaz (*Natural User Interface, NUI*) donde el usuario se comunique sin un dispositivo más que con su propio cuerpo supone un gran potencial para la investigación, ingenieros, investigadores, profesores y músicos han aplicado a lo largo de estos años la tecnología que ofrece Kinect.

Uno de los objetivos futuros de este Proyecto Fin de Grado es que se puedan enfocar los videojuegos desarrollados a la rehabilitación de algún grado de discapacidad física, para que se puedan tratar patologías como daño cerebral adquirido y enfermedades neurodegenerativas. O simplemente que niños con movilidad reducida puedan jugar y divertirse con videojuegos NUI, sin las limitaciones que se encuentran en los convencionales que sacan las grandes empresas.

Por ese motivo se pensó en realizar unas pruebas con sujetos que padecieran alguno de estos problemas y la fundación AENILCE nos ofreció la posibilidad de realizarlas en sus instalaciones. La fundación AENILCE es la nueva titular del colegio concertado con la CAM de educación especial AENILCE donde se atiende y enseña desde hace más de 25 años a niños con lesiones cerebrales de entre 0 y 21 años. Desde el centro se promueve y patrocina el estudio y la investigación de las disciplinas neurocientíficas y las terapias específicas para el tratamiento de la lesión cerebral, por eso estuvieron encantados con la propuesta.

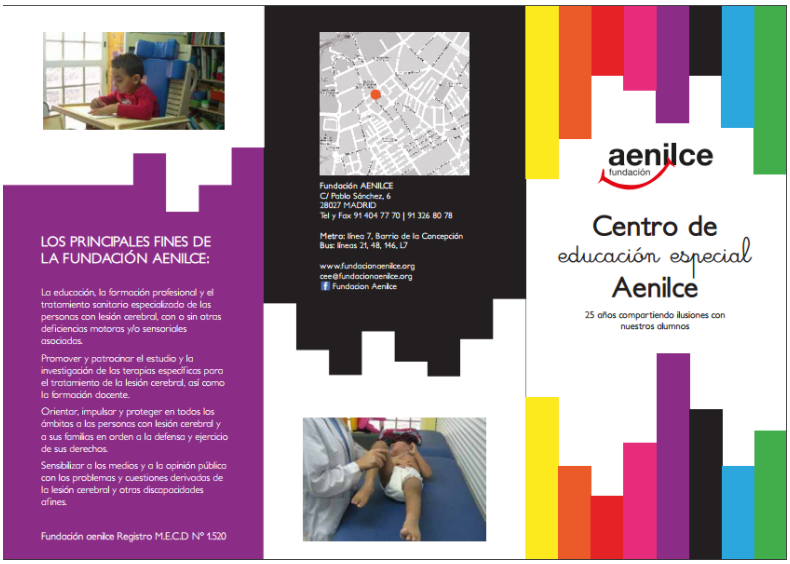


Figura 65. Centro de Educación Especial en el que se realizaron las pruebas

5.1. Descripción de las pruebas.

Las pruebas fueron realizadas el 21 de junio de 2016 en horario de mañana, con la colaboración de Santiago Iglesias, uno de los fisioterapias del centro y Esther De Higes, logopeda. Nos fue ofrecida una de las salas en la que realizar el montaje de todo equipo, e ir llamando a los niños uno a uno para detectar sus impresiones ante la actividad.

La mayoría de los niños que se encontraban en el centro ese día tenían un nivel cognitivo bastante bajo o eran bebés, por lo que sólo cinco de ellos pudieron ser seleccionados para las pruebas. Estos sujetos tenían edades comprendidas entre 4 y 11 años, dos varones y tres niñas. Los resultados obtenidos por cada uno de ellos se describen a continuación.

El primero en jugar fue un niño de 11 años, el mayor de todos, que padece trastorno general del desarrollo referido al espectro autista y lleva desde los 3 años en psicología. Le encantan los juegos de ordenador y mostró muchísimo interés por realizar la actividad, por eso empezamos con él. El juego que proyecta todos los puntos de su cara le llamó la atención porque seguía sus movimientos y era como un espejo, pero enseguida se cansó. El juego de la barca por el mar fue el que más le gustó, entendió muy rápido la lógica del mismo y no le costaba nada guiñar los ojos. A continuación, en el juego de saltar los troncos abriendo la boca tuvo más problema, porque mantenía la boca abierta todo el rato, y claro, el avatar saltaba pero había ocasiones en que al aterrizar en la rampa chocaba con uno de los troncos y caía. Por último con el juego del *Smile* que te imita, ocurrió lo mismo que con el de proyección de puntos faciales. Al principio le llama la atención, pero una vez ha entendido el mecanismo del juego se aburre y quiere dejarlo (Figuras 66 y 67).

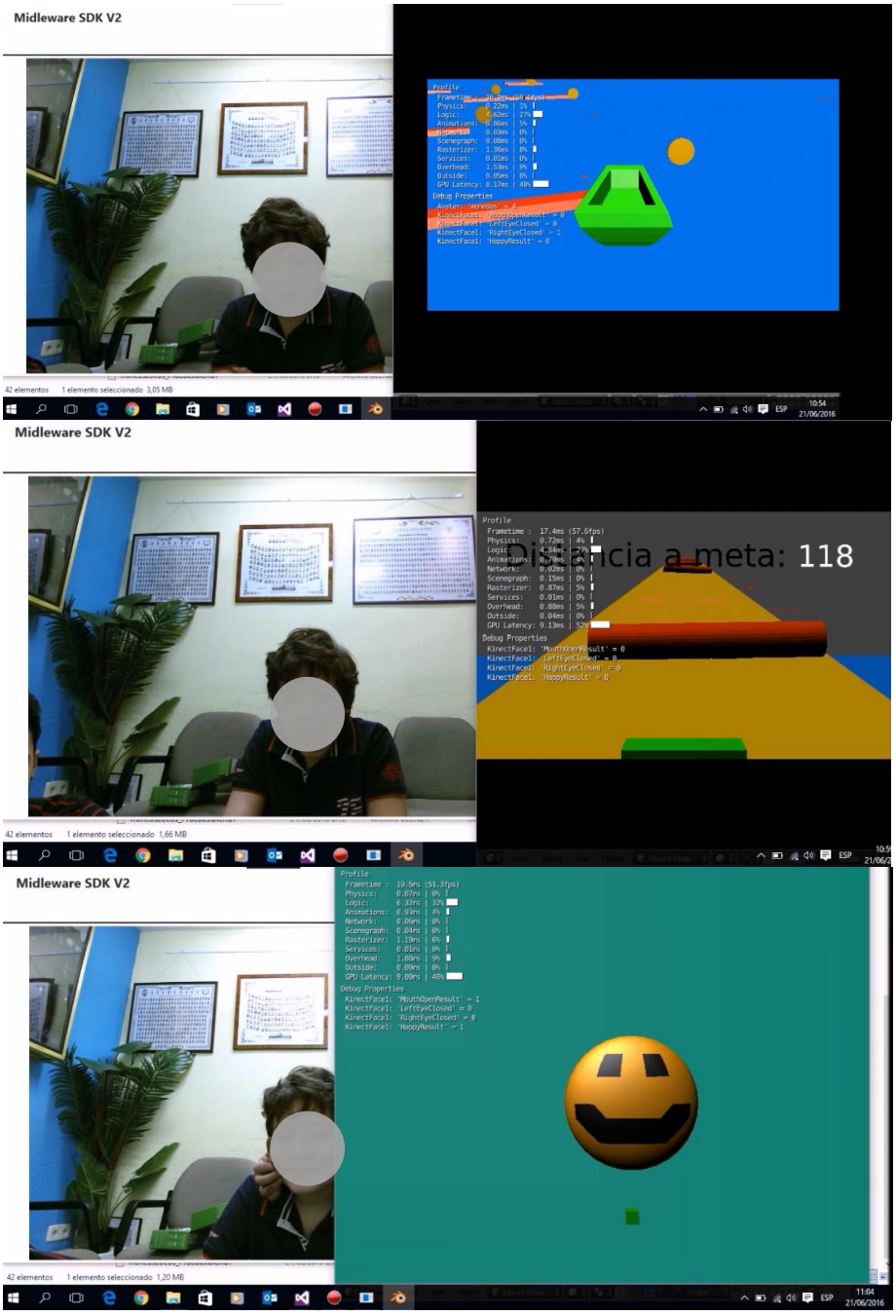


Figura 66. Pruebas realizadas por el primer sujeto (I)

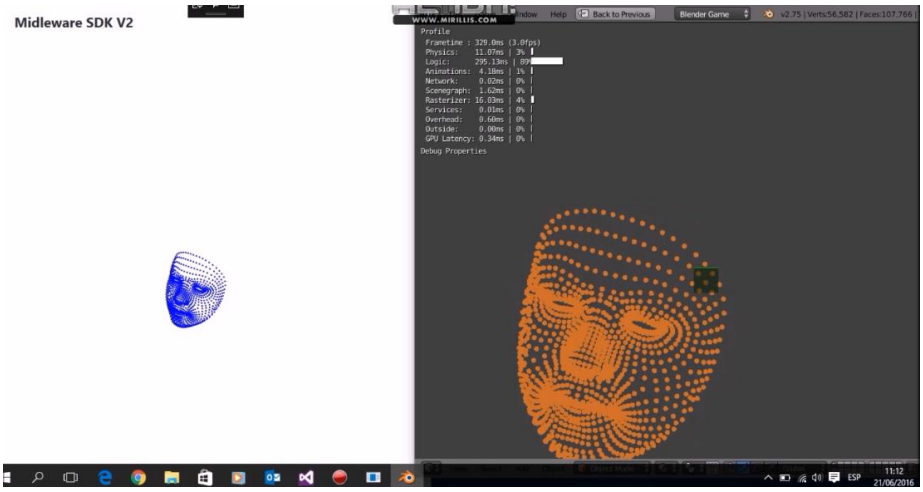


Figura 67.Pruebas realizadas por el primer sujeto (II)

La segunda fue la más pequeña, con solo 4 años que no tiene proyección vocal, no es capaz de reproducir ningún tipo de sonido, actualmente se encuentra bajo estudio para establecer si tiene algún síndrome. Gracias a ello nos dimos cuenta que la función *faceProperties* de 2D, no detecta rostros de tan pequeño tamaño, porque después de varios intentos no hubo manera de conseguirlo. Sin embargo la función de HDFace en 3D, sí que reconocía los puntos faciales de la niña, aunque para ello primero era necesario que detectara a un adulto y posteriormente se pusiera detrás de ella para pasarle la detección. A pesar de que no pudo jugar a los juegos del smile, la barca y los troncos, se entretenía bastante con verse reflejada en la máscara de puntos, suponemos que por su corta edad (Figura 68).

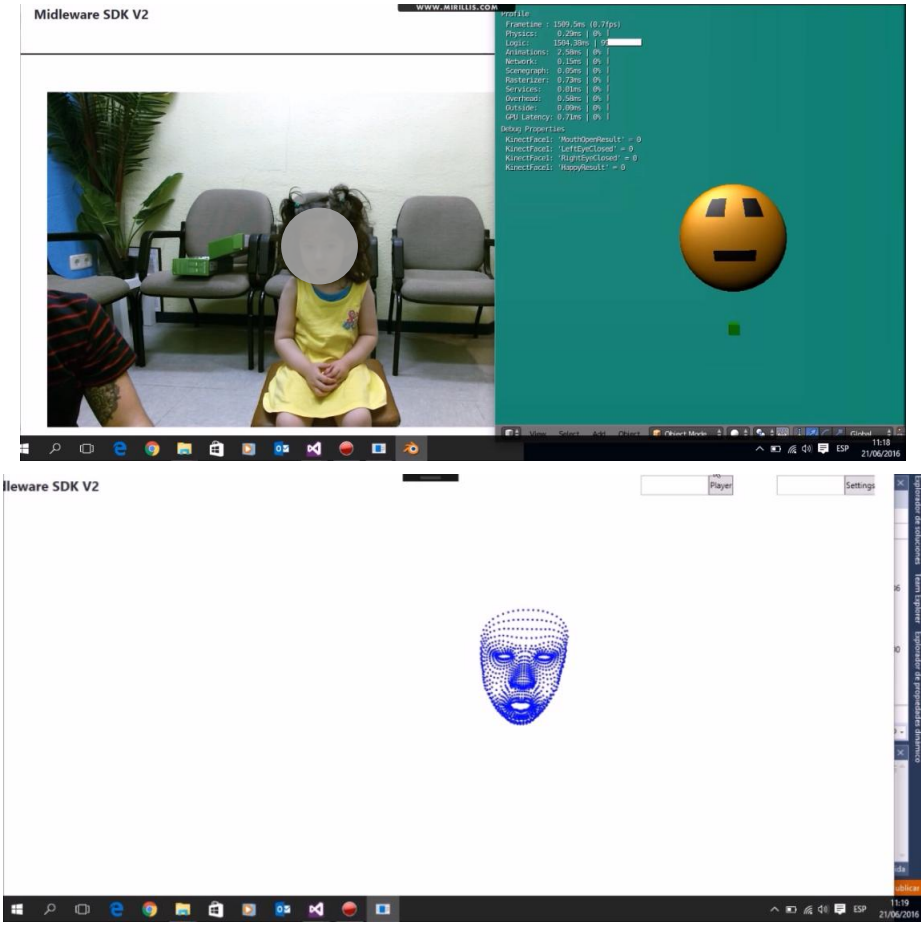


Figura 68. Pruebas realizadas por el segundo sujeto

El tercero de los participantes fue un niño de 5 años el cual padece retraso madurativo, trastorno general del desarrollo (TGD), dentro del espectro autista pero este niño está más afectado a nivel cognitivo, cerca del retraso mental. Al ser pequeño también tenía problemas en la detección, tardaba más pero finalmente se conseguía detectarlo, por ello nos dimos cuenta de que puede que costara detectar a la niña anterior por el corte de pelo a parte de por el tamaño de su cara. En el juego de la barca no conseguía guiñar los ojos, se puso muy nervioso y cambiamos de juego. En el de saltar troncos, el control con la boca era bastante bueno, pero la meta está demasiado lejos y se cansaba del juego. Los juegos de imitación del rostro fueron los que más le llamaron la atención y pasó tiempo observando cómo seguía sus expresiones (Figura 69).

Figura 69. Pruebas realizadas por el tercer sujeto

El cuarto sujeto fue una niña de 9 años, portadora de gafas que padece paraplejia espástica familiar asociado con retraso madurativo. Esta niña a nivel motoro tiene mucha espasticidad, lo que se traduce en un alto grado de rigidez y dificultad articulatoria, a nivel cognitivo lo que más tiene afectado es la producción, el habla, comprende bien pero la producción está bastante alterada, no es capaz de decir más de cinco palabras en total. Las gafas no supusieron ningún obstáculo para el seguimiento de los patrones faciales en los juegos. En el juego de la cara detectaba todas las expresiones salvo la de guiñar lo ojos por separado, esto le impidió por lo tanto jugar con normalidad al juego de la barca. Inteligentemente se tapaba los ojos con las manos porque se la decía que era necesario cerrar uno de los ojos para girar, pero esta acción no era detectada como guiño. En el juego de los troncos, se necesitaría una adaptación al grado de agilidad mental porque cuando ella veía que tenía que saltar era demasiado tarde, porque ya tenía el tronco encima (Figura 70).

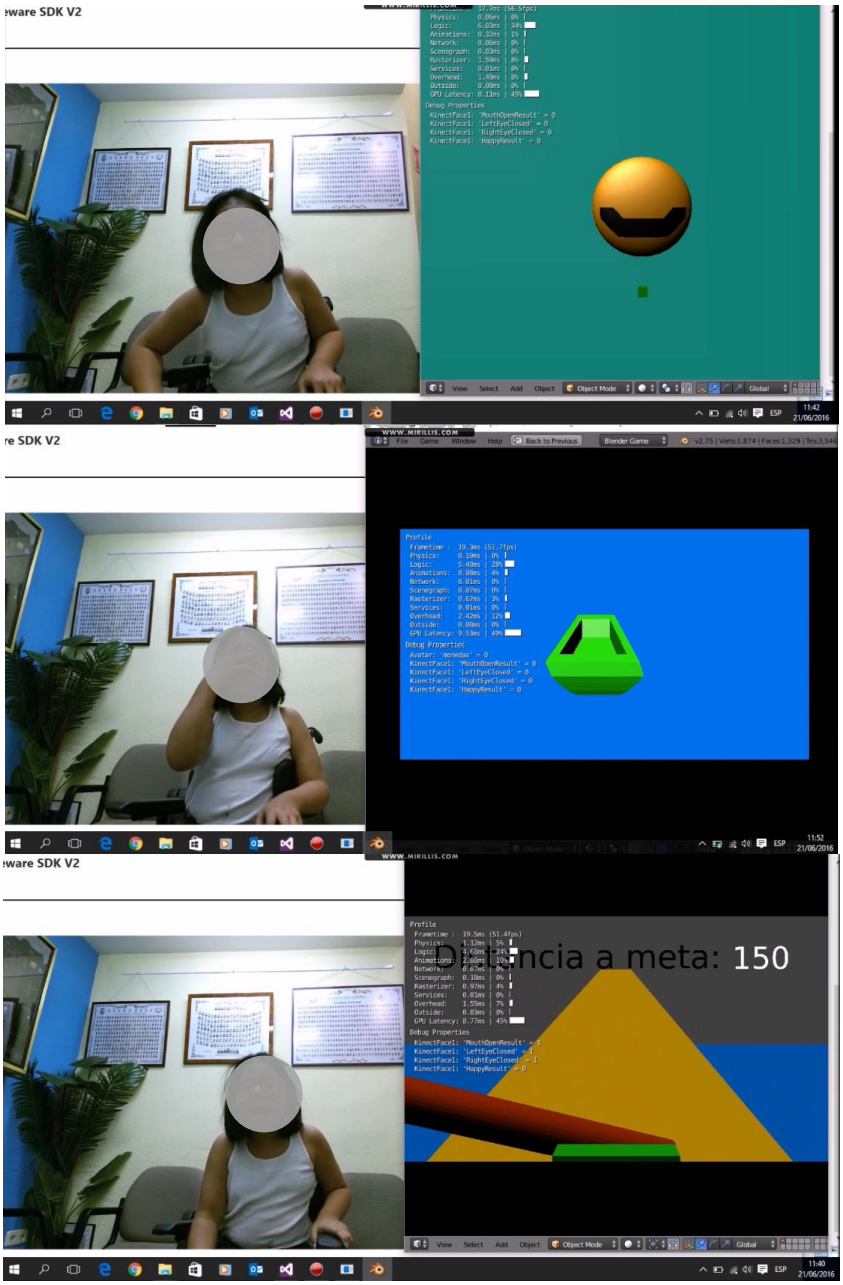


Figura 70. Pruebas realizadas por el cuarto sujeto

La actitud de los niños ante la actividad fue muy buena, todo lo novedoso les llama la atención insistían los trabajadores del centro. Con estas pruebas se deduce a título general, que la detección en rostros de pequeño tamaño es dificultosa y que los juegos con lógica más trabajada son aptos para los más mayores.

5.2. Impresiones de los logopedas y fisioterapeutas del centro.

Los trabajadores del centro mostraron mucho interés por la iniciativa y les pareció una buena forma de motivar a los niños para que realicen sus terapias. Con niños todas las expresiones faciales ligadas a emociones está bien trabajarlas, indicaron que sería un buen ejercicio un juego que salieran imágenes que representaran emociones y el niño tenga que poner la cara que lo expresa. Por ejemplo aparece un niño con una herida, refleja tristeza, luego el niño tiene que poner cara triste.

Además las emociones faciales son las que más ejercitan los músculos de la cara, junto con el habla y la deglución (morder, masticar...). Destacaron que guiñar el ojo, que de hecho muchos

adultos no son capaces de hacerlo, no es tan importante como la capacidad de abrir la boca, sacar la lengua o soplar. Habilidades básicas como lateralizar la lengua, pero serían ejercicios más avanzados.

Lo ideal sería un videojuego por niveles, se consigue controlar la apertura y cierre de la boca pues paso a otro nivel, ordenados por prioridad.

Comentaron que las actividades realizadas eran muy atractivas para los niños, estos niños tienen los tiempos de atención muy reducidos, y estaban bien porque ellos se concentraban e intentaba lograr los objetivos. También que es muy importante el campo de visión de cada niño, la parálisis cerebral está muy relacionada con problemas de visión, por eso mucho de os niños se agachaban y acercaban a la pantalla del ordenador.

6. Conclusiones

En este capítulo se hace un análisis del trabajo realizado en el desarrollo de este Proyecto Fin de Grado, destacando hasta qué punto se han cumplido los objetivos marcados desde el comienzo del mismo.

El propósito principal de este proyecto era la integración del reconocimiento de movimientos faciales ofrecido por Kinect V2 en un videojuego serio de rehabilitación. Para conseguir ese propósito global era imprescindible crear la base de comunicación entre Kinect y el motor de creación de videojuegos Blender. Y se puede decir que el resultado ha sido bastante satisfactorio, ya que se ha desarrollado un sistema que permite intercambiar cualquier tipo de información entre ambos dispositivos.

La fase teórica de este proyecto ha sido la más tediosa, porque se empezaba con lenguajes de programación y un entorno de desarrollo nuevos para el alumno. Afortunadamente las APIs de desarrollo elegidas contienen bastante información con ejemplos detallados y ejecutables, que ayudaron a la interpretación de las posibilidades ofrecidas. La versión del middleware para Kinect V1 en la que se basa el presente proyecto ha sido sin duda imprescindible para saber las líneas a seguir en el desarrollo.

El protocolo OSC de comunicación empleado en el proyecto permite que en el empaquetado del mensaje se pueda incluir cualquier tipo de datos (cadenas de *strings*, números enteros, *floats*, *doubles*...), lo que ha ofrecido gran versatilidad a la hora de compatibilizar los sistemas emisor y receptor.

De las dos librerías ofrecidas por el SDK 2.0 de Microsoft para la detección y el seguimiento del rostro, que se han aplicado en el desarrollo de la aplicación emisora, se destaca lo siguiente:

Microsoft.Kinect.FaceTracking es útil para detección de expresiones básicas y para hacer una primera aproximación del posicionamiento del rostro en la escena. Los movimientos de abrir y cerrar los ojos, poner boca de sorpresa o felicidad, pueden dar mucho juego porque se puede interpretar esta información de mil formas a la hora de realizar un videojuego. Pero tras la realización de las pruebas en el Centro AENILCE, y hablar con los fisioterapeutas y logopedas expertos en el tema, la realidad es que las personas que necesitan realizar ejercicios más serios con los músculos de la cara, requieren más posibilidades de detección.

A pesar de que los puntos en espacio 2D que ofrece este componente cubren los principales puntos de la cara, la falta de una coordenada Z supone un problema, porque cualquier movimiento respecto de la posición de grabación, genera un cambio de distancia entre los puntos, pero no se puede corresponder con la modificación real de la expresión facial. Con solo cinco puntos de referencia está muy limitada la posibilidad de detectar emociones más complejas.

Microsoft.Kinect.HighDefinitionFaceTracking es la segunda opción que se tomó, porque permitía crear una conexión emocional más fuerte entre el jugador y el juego. Esta librería sí que procesa las coordenadas en espacio 3D y permite el seguimiento de bastante más puntos del rostro.

Se procedió de manera similar para el envío de las coordenadas de estos puntos, esta vez de tres componentes, haciendo pequeños cambios en el *addon* de Blender. Pero al incrementarse de tal manera el número de recepciones, la tasa de transmisión de cuadros del juego se ve ralentizada.

Esta librería ofrece la posibilidad de trabajar con 17 AUs (*Action Units*) y más de 70 SUs (*Shape Units*), que permitirían aumentar el grado de detección a un nivel mayor, pero su sintaxis es de C++ y no ha dado tiempo a profundizar en esta posibilidad.

Las pruebas de la aplicación se iban realizando semana tras semana por el propio desarrollador en entorno casero, realizando modificaciones cuando era preciso. Pero el estudio definitivo fue cuando se trasladó el equipo al Centro Educativo y se experimentó con los niños. En general todo salió bien, salvo el problema que había en la aplicación 2D que no era capaz de detectar rostros de pequeño tamaño.

Es positivo señalar que la línea atractiva que siguen los juegos captaba la atención de los niños que realizaron las pruebas, es algo que destacó el personal del Centro, ya que los tiempos de atención de estos niños son muy reducidos.

Se piensa que este sistema puede ser realmente una muy buena posibilidad para cualquier persona que necesite hacer de manera regular sus ejercicios en casa, sin tener que trasladarse a un sitio especializado para realizarlos.

7. Trabajo futuro

El presente proyecto solo es una pequeña parte del plan de desarrollo de un sistema de rehabilitación serio que está en proceso. Ya se dieron los primeros pasos cuando se consiguió captar los movimientos corporales a través de la Kinect V1 y pasándolos a Blender, pero la aparición de una nueva versión del sensor, que ofrece una amplia variedad de mejoras, propició la iniciativa de adaptar todo el trabajo conseguido a esta nueva Kinect. Aún queda mucho trabajo por delante, y ésta sólo ha sido una primera aproximación; la tecnología avanza día a día y siempre habrá detalles que mejorar.

La primera línea de trabajo futuro consistiría en el desarrollo de una aplicación que hiciera uso del método de reconocimiento facial FACS, empleando los AUs y SUs que proporciona la librería *HighDefinitionFaceTracking*, de manera que se obtengan patrones de reconocimiento emocional de mayor complejidad.

Existen ciertos ejercicios que son altamente recomendados en las terapias de rehabilitación facial. La propuesta comprendería movimientos sencillos como subir cejas, fruncir el ceño, poner morro y desplazarlo de un lado a otro, enseñar los dientes y ocultarlos, morderse el labio inferior o superior, llenar la boca de aire y retenerlo. Desde el punto de vista logopédico, también interesaría la detección de la lengua y sus movimientos.

La parálisis cerebral está muy relacionada con los problemas de visión, y en muchos casos afecta al campo de visión. Por ello, otra propuesta de trabajo futuro sería un sistema que detectara la reducción del campo de visión que tiene el paciente y adaptara la pantalla de los juegos al mismo.

Hay que tener en cuenta que no todos los pacientes van a tener la misma velocidad a nivel cognitivo, por lo que sería necesario adaptar la velocidad de los juegos a cada uno. Un mismo juego puede parecerle lento y aburrido a un usuario, mientras que a otro le puede resultar demasiado rápido y complicado.

Habiendo conseguido trasladar la detección facial a Blender, sería interesante trabajar con el reconocimiento corporal también y combinar los movimientos.

Además, debería ser considerada la opción de incluir reconocimiento de voz en el sistema, ya que habría partes de la aplicación que se manejarían muy bien y no sería necesario que el usuario estuviera acompañado en todo momento de alguien que le introduzca los comandos en el propio PC.

Por último, se podría añadir un sistema de representación de resultados y un servidor base de datos, que fuera mostrando al finalizar cada sesión el avance del paciente en la terapia y los resultados obtenidos, de manera que el especialista que le está tratando tenga constancia de sus avances.

8. Referencias

[1] S. K. Modi,*Biometrics in identify Management: Concepts to Applications*, 2011.

[2] A. J. Goldstein, L. D. Harmon y A. B. Lesk,"Identification of human faces". Proc. IEEE 59. págs. 748-760, 1971.

[3] H. Moon y P.J. Phillips,"Computational and Performance aspects of PCA-based Face Recognition Algorithms", *Perception*, vol. 30, págs. 303-32, 2001.

[4] K. Etemad y R. Chellappa, "Discriminant Analysis for Recognition of Human Face Images", *Journal of the Optical Society of America A*, vol. 14, no. 8, págs. 1724-1733, Agosto 1997.

[5] P.N. Belhumeur, J.P. Hespanha y D.J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using Class Specific Linear Projection", *Proc. of the 4th European Conference on Computer Vision*, ECCV'96, Cambridge, UK, págs. 45-58, April 1996.

[6] T.F. Cootes y C.J. Taylor, "Statistical Models of Appearance for Computer Vision", Technical Report, University of Manchester, pág. 125.

[7] L. Wiskott, J.-M. Fellous, N. Krueuger y C. von der Malsburg, "Face Recognition by Elastic Bunch Graph Matching", *Chapter 11 in Intelligent Biometric Techniques in Fingerprint and Face Recognition*, eds. L.C. Jain et al., CRC Press, págs. 355-396, 1999.

[8] E.G. Fernández-Abascal, B. García, M.P.Jiménez, M.D.Martín y Domínguez, "Psicología de la Emoción". Madrid: Editorial Universitaria Ramón Areces. F.J. 2010.

[9] M. Singh, A. Majumder, y L. Behera, "Facial expressions recognition system using Bayesian inference," *Int. Jt.* Conf. *Neural Networks*, págs. 1502–1509, Julio 2014.

[10] M. Taner Eskil y K. S. Benli, "Facial expression recognition based on anatomy," *Comput. Vis. Image Underst.*, vol. 119, págs. 1–14, Febrero 2014.

[11] P. Lucey, J. Cohn, y T. Kanade, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," *Comput. Vision and Pattern Recognition Workshops (CVPRW)*, Julio 2010.

[12] N. Khan, "A comparative analysis of facial expression recognition techniques," IEEE 3rd InternationalAdv. Comput. Conf. (IACC),págs. 1262–1268, 2013.

[13] S. Zhang, X. Zhao, y B. Lei, "Robust facial expression recognition via compressive sensing.," *Sensors (Basel).,* vol. 12, no. 3, págs. 3747–61, Enero 2012.

[14] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vis*., vol. 60, no. 2, págs. 91–110, Noviembre 2004.

[15] "centrodeartigo.com", Disponible en:<http://centrodeartigo.com/articulos-enciclopedicos/article_96478.html> [Consulta: 15 de marzo de 2016].

[16] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput*. *Vis*., vol. 60, no. 2, págs. 91–110, Noviembre 2004.

[17] M. Abrego, "¿Sabías que Kinect para Windows puede escuchar y puede verte?" [en linea]. malenyabrego.wordpress.com. Disponible en: <https://malenyabrego.wordpress.com/2012/10/22/sabias-que-kinect-para-windows-puede-escuchar-y-puede-verte/> [Consulta: 20 de marzo de 2016].

[18] A. Palma, "Kinect para Xbox One" [en línea]. soloxboxone.com. Disponible en: <http://soloxboxone.com/kinect-para-xbox-one/> [Consulta: 20 de mayo de 2016].

[19] A. Kouba,"Robot Operating System (ROS)", *Springer*, Vol 1, págs. 112-156, Enero 2016.

[20] F. Biermann, N. Steenbergen y B. Walther-Franks , "BLOOP Rapid Motion Capturing using Blender and Kinect", Universität Bremen, [en línea]. Blender.org. Disponible en: <http://download.blender.org/documentation/bc2011/Bloop_Blender259.pdf> [Consulta: 10 de junio de 2016].

[21] A. Martinez, "Introducción al OSC (Open Sound Control)- Primera parte", www.inventable.eu , 3 de agosto de 2010. Disponible en: <http://www.inventable.eu/2010/08/03/introduccion-al-osc-open-sound-control-primer-parte/> [Consulta: 11 de junio de 2016].

[22] I. Gomez, "Estructura del middleware Chiro" Informe final de prácticas, CITSEM, Madrid, 2015.

[23] Microsoft, "Face Frame Result Class" [en línea]. Disponible en:<https://msdn.microsoft.com/en-us/library/microsoft.kinect.face.faceframeresult.aspx> [Consulta: 10 de marzo de 2016].

[24]Microsoft, "Kinect for Windows SDK 2.0" [en línea]. <https://msdn.microsoft.com/en-us/library/Dn785525.aspx> [consulta: 12 de abril de 2016].

[25]Autor desconocido. "Blender 3D. Noob to Pro/ Advanced tutorials/ Python Scripting/ Introduction" [en línea]. Wikibooks.org. Disponible en:<https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Advanced_Tutorials/Python_Scripting/Introduction> [Consulta: 18 de abril de 2016].

[26]B. Elons. "Kinect coordinate mapping summary: summary and pitfalls" [en línea]. Disponible en: <http://blog.elonliu.com/kinect-coordinate-mapping-summary-and-pitfalls/> [Consulta: 22 de abril de 2016].

9. Anexos

Anexo I. Instalación Kinect SDK 2.0

Si se quiere manejar el kinect de la nueva Xbox One, es necesario descargar el SDK adecuado, en este caso la versión 2.0, que se descarga gratuitamente desde Microsoft.

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

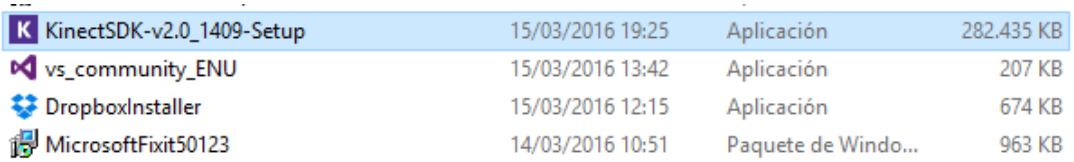


Figura 71. Ejecutable del SDK 2.0.

Una vez la descarga se ha completado se abre el ejecutable mostrado en la Figura 71, se acepta la licencia y empieza la instalación. Es importante no tener conectada la Kinect durante el proceso.

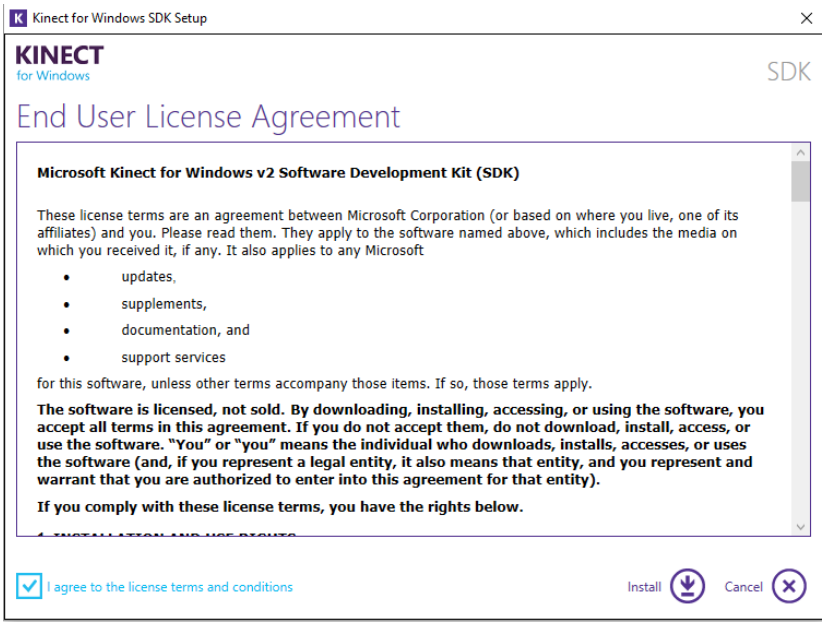


Figura 72. Asistente de instalación

Ahora es el momento de conectar la Kinect al PC, es recomendable seguir los pasos de la Figura 73.

Figura 73. Pasos para realizar la conexión de Kinect.

Para comprobar si el SDK está trabajando correctamente, se accede al *path* de la Figura 74 y se corre KinectV2ConfigurationVerifier.

C:\Program Files\Microsoft

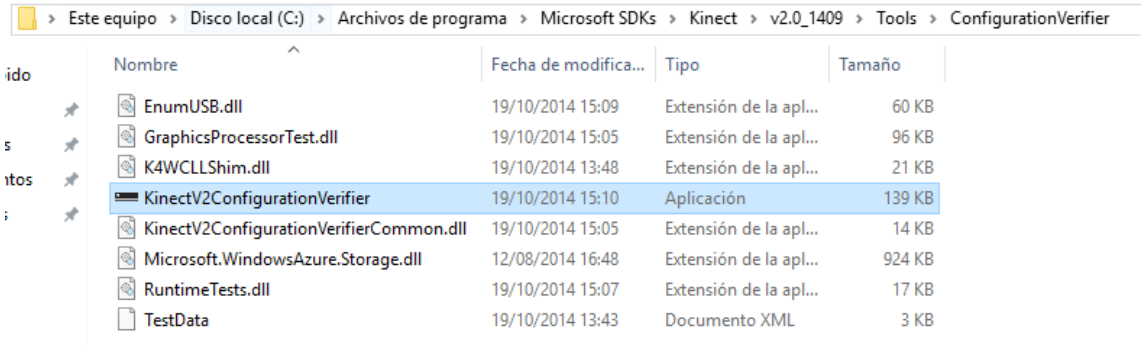


Figura 74. Ejecutable de KinectV2ConfigurationVerifier.

Si todo está correcto aparece una ventana como la mostrada en la Figura 75.

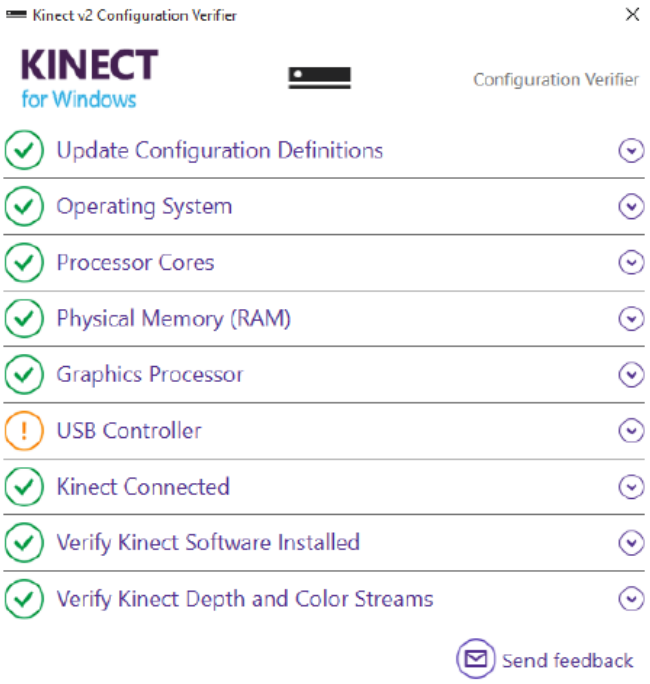


Figura 75. Asistente de KinectV2ConfigurationVerifier.

En nuestro caso marca una advertencia, es debido a que se está trabajando con un puerto USB 3.0 estandarizado por Microsoft, sin embargo, funciona perfectamente.

Una vez queda todo comprobado, ya se puede empezar a trabajar con la Kinect. En el siguiente bloque se analizará la forma de preparar el entorno de desarrollo.