Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

APLICACIÓN DE REALIDAD VIRTUAL CON COMUNICACIÓN ENTRE DISPOSITIVOS EN TIEMPO REAL

Alumno: Pedro Aguilar Aguilar

Tutor: José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Septiembre, 2019

ÍNDICE DE GENERAL

AGRADECIMIENTOS

Quiero agradecer a todas las personas que en mayor o menor medida me ha ayudado en la realización de este TFG. En especial a mi tutor José Enrique Muñoz Expósito y a todos los compañeros del Departamento de Ingeniería de Telecomunicación, que día tras día me han empujado para la realización de este TFG.

A mi familia, Marian, María y Carla, por darme el tiempo necesario para poder continuar con mi formación.

GLOSARIO

CPU	*Central Processing Unit (Unidad de Procesamiento Central)*
CV	*Computer Vision (visión artificial)*
EEPROM	*Electrically Erasable Programmable Read-Only Memory* *(*ROM programable y borrable eléctricamente *)*
FPS	*First Person Shooter (tirador en primera persona)*
HDMI	High-Definition Multimedia Interface (salida de video digital)
IDE	*Integrated Development Environment* *(entorno de desarrollo integrado)*
IMU	*Inertial Measurement Unit (unidad de medición inercial)*
ISO	Imagen completa del sistema operativo.
IP	*Internet Protocol (Protocolo de Internet)*
JSON	*JavaScript Object Notation (notación de objeto de JavaScript)*
LAN	*Local Area Network (red de área local)*
MCU	*Microcontroller (microcontrolador)*
PLA	*Polylactic Acid* (Filamento para la impresión 3D)
SPI	*Serial Peripheral Interface Bus (bus serial de interfaz de periféricos)*
SSID	*Service Set Identifier (nombre de red)*
TFG	*Trabajo Fin de Grado*
TTL	*Transistor-Transistor Logic (Lógica transistor a transistor)*
UDP	*User Datagram Protocol (Protocolo de Datagrama de Usuario)*
URL	*Uniform Resource Locator (Localizador Uniforme de Recursos)*
USB	*Universal Serial Bus (Bus Universal en Serie)*
VR	*Virtual Reality (realidad virtual)*
WIFI	*Red inalámbrica*

ÍNDICE DE FIGURAS

1. RESUMEN

1.1 Resumen

Este trabajo fin de grado (TFG) tiene como objetivo la creación de una aplicación de realidad virtual, dicha aplicación realizará una comunicación con dispositivos en tiempo real. La aplicación propuesta es un videojuego de estilo *FPS*.

Para la realización de la aplicación de realidad virtual, se utilizará el entorno de desarrollo *Unity*, incorporando las librerías necesarias para la comunicación con los dispositivos. Se han desarrollado 2 dispositivos, por un lado tenemos la pistola que es la encargada de recibir los datos de elevación y orientación, por otro lado se ha configurado un dispositivo que a través de un procesado de imagen, sirve para determinar los movimientos del jugador dentro de la estancia.

La información recopilada por los 2 dispositivos es enviada a la aplicación principal para su procesamiento, interactuando dentro del entorno virtual. Durante la realización de este TFG se realizaron diferentes pruebas para que el sistema fuese atractivo. Se han realizado varias demostraciones de su funcionamiento para grupos de estudiantes de ESO y bachillerato como parte de las actividades de las distintas jornadas de divulgación de las titulaciones de la EPS de Linares. En todas ellas se ha obtenido muy buena acogida, despertando gran interés entre el alumnado.

Figura 1.1 Visita el IES Santa Engracia.

1.2 Abstract

This final degree project (TFG) aims to create a virtual reality application, this application will communicate with some devices in real time. The proposed application is an FPS style videogame.

For the realization of the virtual reality application, the Unity development environment will be used, incorporating the libraries necessary for communication with the devices. Two devices have been developed, one of them are the gun that receive the elevation and orientation data, the other device has been configured that through an image processing, serves to determine the movements of the player.

The information collected by the 2 devices is sent to the main application for processing, interacting within the virtual environment. During the realization of this project, different tests were carried out so that the system was attractive. There have been several demonstrations of its operation for groups of students of ESO and high school as part of the activities of the different days of dissemination of the degrees of the EPS of Linares. In all of them, very good reception has been obtained, arousing great interest among the students.

2. INTRODUCCIÓN

En este capítulo se explicará los antecedentes y las circunstancias que han motivado el desarrollo de este TFG. Se describe también, la estructura del presente documento, detallando el contenido de cada uno de los capítulos que lo componen.

2.1 Antecedentes

Desde la creación del primer videojuego “Tennis For Two", la tecnología ha ido avanzando a pasos agigantados. Conforme se mejoraban los dispositivos, los videojuegos han evolucionado en gráficos, sonido, jugabilidad, etc. En el aspecto de la jugabilidad ha sido varios los dispositivos que se han utilizado. El primer mando de videojuegos fue el utilizado en el Pong.

Figura 2.1.1 Mando Pong.

La evolución de los mandos para videojuegos ha sufrido varias evoluciones, desde el mando para la *Atari 2600*, pasando por el mando de la *Nintendo NES*. En esta evolución cabe destacar el cambio de concepto que supuso los mandos para la *Nintendo Wii*, ya que el jugador debía de realizar movimientos más parecidos a la realidad.

Figura 2.1.2 Mando Nintendo Wii.

En la actualidad hay un aumento del uso de las nuevas tecnologías. Los *Smartphone* se han consolidado como los dispositivos más utilizados por la población y el consumo de videojuegos se incrementa año a año. Existen diferentes dispositivos para el control de los videojuegos, los sistemas de realidad virtual, se perfilan como la opción principal para el futuro.

Actualmente los sistemas de realidad virtual en algunos casos, son dispositivos caros y requieren de equipos de gran potencia, como pueden ser *HTC Vive*, *Oculus Rift, Xiaomi VR,* etc.

Figura 2.1.3 HTC Vive.

El elevado coste de estos dispositivos, hace que la incorporación a los hogares no esté siendo tan rápida como se esperaba. Los fabricantes de dispositivos están sacando al mercado equipos que pueden ser utilizados de forma autónoma e inalámbrica, de tal forma que aumenta la inmersión en la realidad virtual. Estos dispositivos disponen de unos sistemas operativos básicos para la ejecución de diferentes aplicaciones.

Figura 2.1.4 Oculus Go.

2.2 Justificación

En este TFG se va a tratar de implementar un sistema de realidad virtual, con dispositivos de bajo coste. Se quiere conseguir una experiencia lo más inmersiva posible. De tal forma que el usuario tenga una experiencia lo más parecida a la realidad. La aplicación que se ha desarrollado, es un videojuego *shooter* en primera persona. Para la realización de la aplicación se ha utilizado *Unity*. Dentro del juego se dispone de un espacio virtual, por el que el jugador puede desplazarse. Para conseguir esto, se ha utilizado una *Raspeberry Pi*, con una aplicación de reconocimiento de imagen. Dentro del juego se tiene una pistola virtual, para comandar sus movimientos, se ha creado una pistola, realizada con impresión 3D. Este dispositivo, tiene contenido un *Arduino* y un *IMU*, para enviar al juego la posición de la pistola.

Una de las principales motivaciones para la realización de este TFG es realizar demostraciones a colegios e institutos, explicando las diferentes tecnologías para la realización del sistema. Se ha realizado el juego para que sea atractivo para los posibles usuarios. Cabe destacar, que ya se ha estado utilizando en las divulgaciones de la Escuela Politécnica Superior de Linares, con una gran acogida por parte de los visitantes.

Figura 2.2 Escuela Politécnica Superior de Linares.

2.3 Estructura del documento

A continuación se expone los diferentes capítulos que contiene este documento:

- **Capítulo 1. Resumen:** Se expone brevemente un resumen del trabajo realizado en el TFG.
- **Capítulo 2. Introducción:** En este capítulo se expone la justificación y el contexto de este documento.
- **Capítulo 3. Objetivos:** Se describirán los objetivos planteados en la realización de este TFG.
- **Capítulo 4. Materiales y métodos:** Se describen las principales tecnologías utilizadas para la realización de la aplicación, técnicas utilizadas y arquitectura de comunicaciones entre los diferentes dispositivos.
- **Capítulo 5. Resultados y discusión:** Se explicarán los resultados obtenidos, comparándolos con dispositivos comerciales.
- **Capítulo 6. Conclusiones:** Se detallan las conclusiones obtenidas en la realización del sistema.
- **Capítulo 7. Líneas futuras:** Se comentarán las posibles mejoras, para el sistema de realidad virtual desarrollado.
- **Capítulo 8. Estudio económico:** Se realizará un presupuesto para la realización del proyecto.
- **Capítulo 9. Referencias:** Reseñas a los libros, documentos online, revistas y sitios web que se han consultado durante la realización del TFG.
- **Capítulo 10. Anexos:** Se incorporan los anexos necesarios para una correcta interpretación de este TFG.

3. OBJETIVOS

Se realizará una aplicación de realidad virtual para dispositivos móviles. La aplicación recibirá datos en tiempo real de dispositivos de bajo coste, los cuales recogen los datos de sensores (giroscopio, brújula , webcam, etc.).

Para el envío de datos entre los dispositivos se utilizará una red WIFI, utilizando los protocolos necesarios para garantizar una experiencia en tiempo real. El sistema de realidad virtual debe ser de bajo coste y fácilmente configurable, para su utilización en talleres y demostraciones.

Para cubrir dichos objetivos se deben realizar una serie de tareas relacionadas:

- Desarrollar una aplicación para dispositivos móviles, utilizando el entorno de desarrollo *Unity*.
- Determinar el dispositivo microcontrolador, capaz de enviar los datos recibidos de un sensor de movimiento.
- Determinar el mejor sensor de movimiento para poder enviar al dispositivo móvil, la posición de la pistola.
- Estudiar los diferentes métodos posibles para incorporar la posibilidad de movimientos dentro del entorno virtual.
- Realizar un envío de datos en tiempo real, para conseguir una mejora de la experiencia.
- Realizar un modelo 3D de la pistola, utilizando programas de edición 3D y software de impresión 3D.
- Realizar el software necesario para el procesado de video y su posterior envío de datos al dispositivo móvil.
- Recibir, procesar y filtrar los datos de los sensores, para poder determinar de forma exacta la posición de la pistola.
- Realizar un control de estabilidad de la visión en primera persona.
- Implementar un sistema de juego, que sea lo más atractivo posible.

4. MATERIALES Y MÉTODOS

En este capítulo se describirán las tecnologías usadas en el desarrollo del presente TFG y los motivos de su utilización. Se describirá el esquema de conexión entre los distintos dispositivos, así como una breve descripción de los lenguajes de programación utilizados. Finalmente, se detallan las pruebas realizadas con los diferentes sistemas de posicionamiento.

4.1 Tecnologías usadas

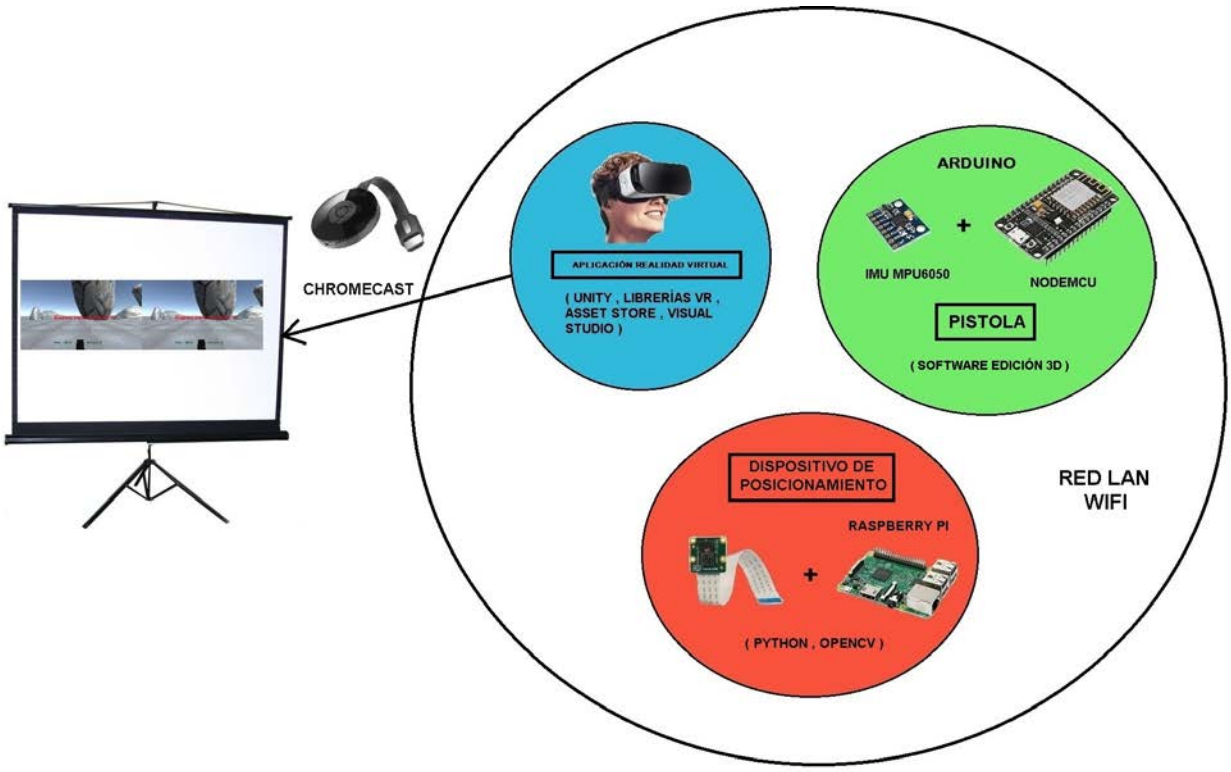

Figura 4.1 Tecnologías Usadas.

Dentro de las tecnologías utilizadas cabe distinguir las tecnologías, que han sido necesarias para la creación de los 3 dispositivos que componen el sistema. Por una parte está la aplicación para el dispositivo móvil, por otro lado, la pistola y por otro lado el sistema de control de posicionamiento del jugador dentro del entorno virtual.

4.1.1 Unity

Unity es un entorno de programación para el desarrollo de videojuegos creado por la empresa *Unity Technologies. Unity* ha sido desarrollado para funcionar en multiplataforma, teniendo compatibilidad con los principales sistemas operativos del mercado. Con él se puede desarrollar aplicaciones para gran cantidad de dispositivos, esta es una de las principales ventajas de *Unity*, ya una vez creada la aplicación es fácilmente exportable a otros sistemas operativos, para su funcionamiento.

Puede trabajar con diferentes tipos de motores gráficos como pueden ser *OpenGL, Directx y OpenGL ES* (especifico en dispositivos móviles). Incorpora además un sistema de gran versatilidad para incorporar diferentes librerías, de esta forma se pueden controlar interfaces de diferentes fabricantes. Existe además una gran comunidad de desarrolladores de librerías para *Unity*.

Figura 4.1.1 Logotipo Unity.

En capítulos posteriores explicaremos en detenimiento el interfaz de desarrollo.

4.1.2 Librerías VR (Virtual Reality)

Para adaptar la aplicación a un sistema de Realidad Virtual, ha sido necesaria la instalación de librerías específicas. Esta librería implementa un sistema para dividir la pantalla de 2, de esta forma colocando nuestro dispositivo móvil en una Cardboard (tecnología que permite convertir dispositivos móviles, en dispositivos de VR) podemos visualizar nuestra aplicación en 3D.

Esta librería puede llevar un control de los movimientos de la cabeza del jugador, para incorporarlos en el juego. Además de un filtrado de los sensores internos de los dispositivos móviles.

4.1.3 Asset Store (Repositorios de modelos 3D)

Unity dispone de un sistema de descarga e instalación de modelos y librerías llamado *Asset Store*. Para el desarrollo del juego se han utilizado diferentes modelos 3D. Por ejemplo, dentro del juego, unos de los principales objetos son los zombies, este modelo ha sido descargado y utilizado. La gran ventaja de utilizar modelos descargables es que incorporan una serie de movimientos predefinidos, lo que facilita en gran medida la creación del videojuego.

4.1.4 Visual Studio

Para poder dotar a los objetos de *Unity* de diferentes acciones necesarias para la realización del videojuego, es necesario, incorporar funciones a dichos objetos. Para esto se ha utilizado Visual Studio, que se trata de un entorno de programación desarrollado por *Microsoft*. Dentro de Visual Studio, el lenguaje de programación utilizado ha sido *C Sharp*, el cual está orientado a objetos. Su sintaxis básica está derivada de *C / C++*. En capítulos posteriores explicaremos en detenimiento algunas de las funciones más importantes para la realización de la aplicación.

4.1.5 Arduino

Para recibir los datos del sensor de movimientos, dentro de la pistola, se ha utilizado el microcontrolador *NodeMCU*. Este dispositivo está programado con el entorno de desarrollo *Ardunio IDE*. El lenguaje de programación del dispositivo es C. Arduino es un proyecto basado en hardware y software libre.

Figura 4.1.5 Logotipo Arduino.

NodeMCU no es un dispositivo nativo de Arduino, pero gracias, a una librería específica, se puede añadir este dispositivo al *IDE*, lo que facilita en gran medida su programación. Se incluye en el Anexo 10.1, el manual de instalación, de las placas basadas en el dispositivo ESP8266, dentro del entorno Arduino.

4.1.6 NodeMCU

NodeMCU es una plataforma de código abierto. Incluye el microcontrolador que se ejecuta en el SoC Wi-Fi ESP8266 de la empresa Espressif Systems. El hardware específico del módulo utilizado es ESP-12E. El dispositivo NodeMCU incluye además un conversor *USB-TTL* para facilitar la programación del módulo ESP12E. Como principal característica de este dispositivo es su bajo coste, además de sus grandes prestaciones, ya que incluye una *CPU* de 32bits. Además este módulo dispone de una interfaz wifi, que permite trabajar como punto de acceso wifi o como cliente wifi.

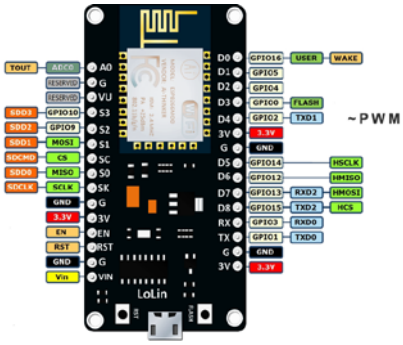

Figura 4.1.6 ***NodeMCU Pinout.***

4.1.7 Sensores de unidad de medición inercial (IMU)

Los sensores *IMU* son dispositivos electrónicos de medida, que son capaces de detectar la posición en la que se encuentran, a través de acelerómetros, giroscopios y una brújula interna. Para este TFG se ha optado en utilizar el *IMU MPU-6050,* por su bajo coste y su fácil utilización. Este se comunica con nuestro *NodeMCU* por un bus *SPI* (*Serial Peripheral Interface*).

Figura 4.1.7 Sensor MPU-6050.

4.1.8 Raspberry PI

Para el posicionamiento del jugador dentro del entorno virtual, se estudiaron diferentes soluciones, que se explicaran en capítulos posteriores. Finalmente para la realización de la solución planteada, fue necesario utilizar la *Raspberry PI*.

Raspberry PI es un mini ordenador completo, de bajo coste. Dependiendo del modelo incluye unas características u otras. Para la realización de este TFG se ha utilizado la *Raspberry Pi 3*, este modelo incluye una gran cantidad de puertos de entrada y salida digitales, así como interfaces wifi y bluetooth. La placa *Raspberry Pi 3* dispone de 4 puertos USB, una salida de video HDMI, un interfaz *Gigaethernet*, un puerto para conectar la cámara, así como una ranura para tarjetas *microSD*. La interfaz wifi, será utilizada para el envío de datos al dispositivo móvil que ejecuta el juego.

Figura 4.1.8 Raspberry Pi 3.

4.1.9 Python

Dentro de la Raspberry PI se ha desarrollado una aplicación que recoge las imágenes de la webcam. Para realizar esta aplicación se ha utilizado *Python*. Este lenguaje de programación es un lenguaje interpretado, orientado a facilitar la lectura del código fuente. Se trata de un lenguaje de programación orientado a objetos, programación imperativa y programación funcional. Es un lenguaje multiplataforma, ya que es posible ejecutarse en diferentes sistemas operativos. Actualmente Python está siendo utilizado para diferentes soluciones, tanto en cliente, como en servidor.

Figura 4.1.9 Python logo.

4.1.10 Open CV

Para poder recoger y analizar las imágenes de la webcam, conectada a la *Raspberry PI*, se ha utilizado las librerías de *OpenCV*, que facilitan en gran medida la tarea. *OpenCV* es una librería libre de visión artificial originalmente desarrollada por Intel.

Desde su creación se ha utilizado para gran cantidad de aplicaciones, como por ejemplo, sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. *OpenCV* es multiplataforma, pudiéndose utilizar en diferentes sistemas operativos (Linux, Windows , Android, OSX, etc.).

4.1.11 LAN

Para el envío de datos, tanto de la pistola, como desde la *Raspberry PI*, ha sido necesaria la creación de una *LAN*. Para la realización del sistema se colocado el dispositivo móvil como punto de acceso, al cual se conectan vía *WIFI*, el *NodeMCU* y la *Rasbperry PI*. Dado que nuestra aplicación de funcionar en tiempo real, después de varias pruebas se optó por utilizar el protocolo *UDP* para el transporte de datos, ya que simplifica y agiliza el sistema.

4.1.12 Chromecast

Este dispositivo juega un papel importante en hacer atractivo el videojuego para los usuarios. En un principio, únicamente la persona jugaba podía experimentar la sensación de realidad virtual. Durante las visitas y los talleres se observó que era difícil mostrar cómo funcionaba el sistema a los visitantes. Se optó por la solución de utilizar C*hromecast* para poder enviar lo que estaba viendo el jugador a un proyector y de esta forma hacer más participes del juego a los visitantes. Esto incrementó de una manera notable la experiencia global del grupo.

Chromecast es un dispositivo de reproducción multimedia, desarrollado por la empresa Google. Como principal característica, cabe destacar, la facilidad para realizar la tarea de Pantalla Remota, reproducción de videos y ejecución de aplicaciones multimedia.

Figura 4.1.12 Chromecast.

4.1.13 Software edición 3D

Para la realización de la pistola ha sido necesario utilizar un software de edición 3D. El software utilizado ha sido ***SketchUp***. Se trata de un software creado por la empresa Last Software. Actualmente Google es la empresa propietaria del software. Se ha utilizado la versión de evaluación para realizar los modelos de la pistola.

SketchUp permite realizar los modelos 3D complejos, a partir de figuras geométricas simples, como círculos, cuadrados y hexágonos. Dispone de la posibilidad de incorporar librerías externas para realizar suavizado de curvas. *SketchUp* es un software con mucha documentación en internet, lo que facilita su utilización.

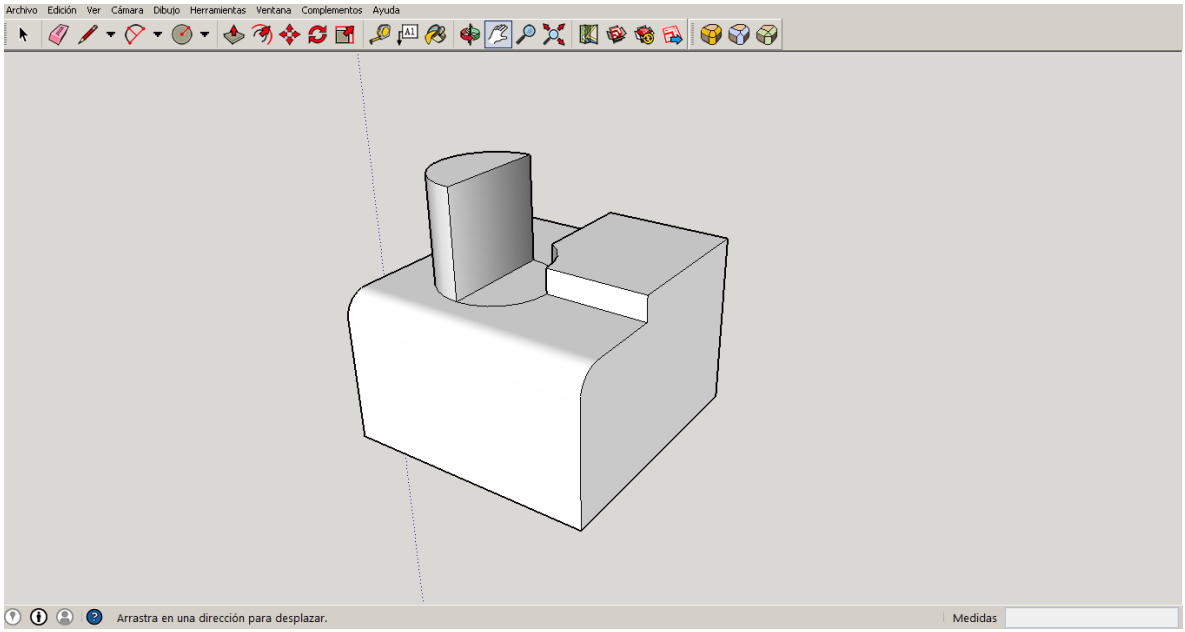

Figura 4.1.13 Software Edición 3D

4.2 Análisis y diseño

En este apartado se explicará el diseño del sistema de realidad virtual, así como el proceso que ha llevado a la solución implementada.

4.2.1 Elementos del sistema de realidad virtual

El sistema está dividido en 3 elementos bien diferenciados, por un lado se tiene el dispositivo móvil, donde se ejecuta el videojuego. Como ya se ha comentado anteriormente se ha utilizado, para la creación de la aplicación en el dispositivo móvil, Unity, Librerías VR, *Asset Store* (descarga de modelos) y Visual Studio para la creación de los scripts. Por otro lado está la pistola, para la que ha sido necesario utilizar Arduino, un módulo NodeMCU y un sensor IMU. El último elemento desarrollado, ha sido el dispositivo de posicionamiento, para el cual ha sido necesario utilizar una *Raspberry Pi*, *Python* y *OpenCV*. Se comentará en las siguientes secciones con más detenimiento cada uno de los elementos. Estos 3 dispositivos se encuentran conectados siguiendo el siguiente esquema:

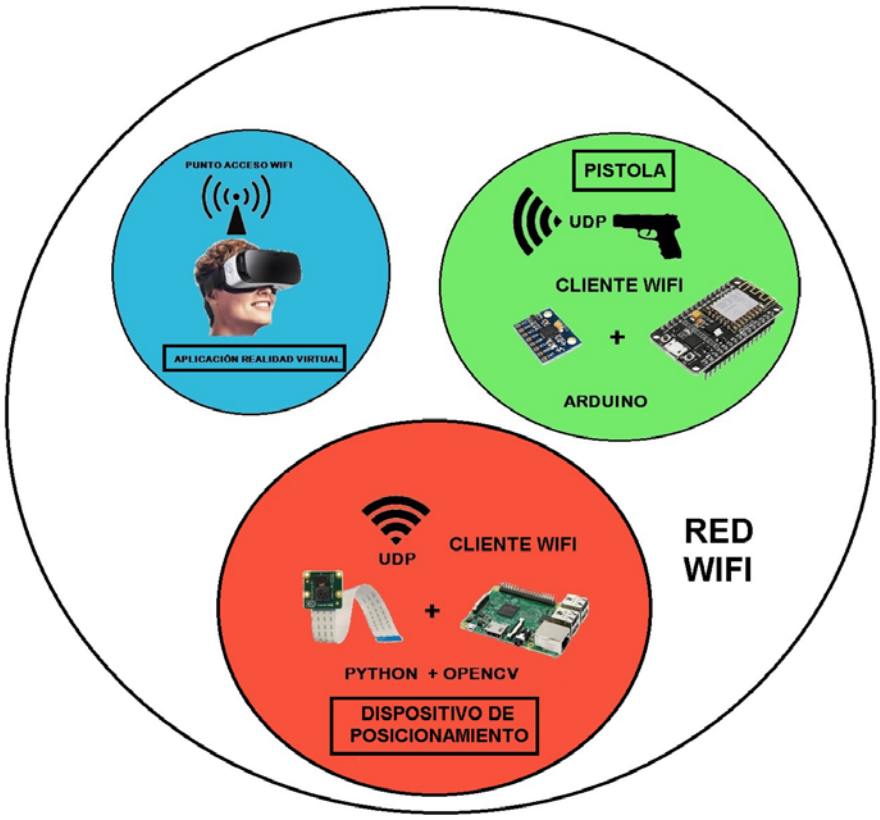

Figura 4.2.1 Elementos del sistema de realidad virtual.

Como se puede observar viendo la Figura 4.2.1 el dispositivo móvil se configura en modo Punto de Acceso de tal forma que los otros 2 dispositivos se conectan a él, vía wifi. Se ha utilizado en la capa de Transporte el protocolo UDP, ya que permite enviar los datos de forma más rápida.

4.2.2 La Pistola

Una de las partes principales de este proyecto, era conseguir trasladar los movimientos de la pistola física al mundo virtual. Para conseguirlo se han realizado diferentes pruebas, hasta conseguir un resultado óptimo. Para realizar las primeras pruebas, fue necesario crear una aplicación que recibía los datos de la *IMU* y se mostraban por pantalla. Se observó que los datos proporcionados por el MPU6050, eran muy variables, por lo que fue imprescindible realizar un calibrado previo para los valores obtenidos de la brújula interna del sensor, que corresponden con los ejes magnéticos (X,Y,Z), ya que dependiendo del lugar donde se encontraba el dispositivo, se obtenían diferentes valores (X,Y,Z). Para realizar el calibrado, fue necesario orientar el sensor MPU6050, hacía el Polo Norte, anotar los valores obtenidos y restarlos a los valores de los ejes magnéticos (X,Y,Z). De esta forma los valores obtenidos si el sensor MPU6050 apunta hacia el Polo Norte serán (0,0,0). En la referencia [1] de la bibliografía, puede observarse con más detenimiento el proceso para realizar una correcta calibración.

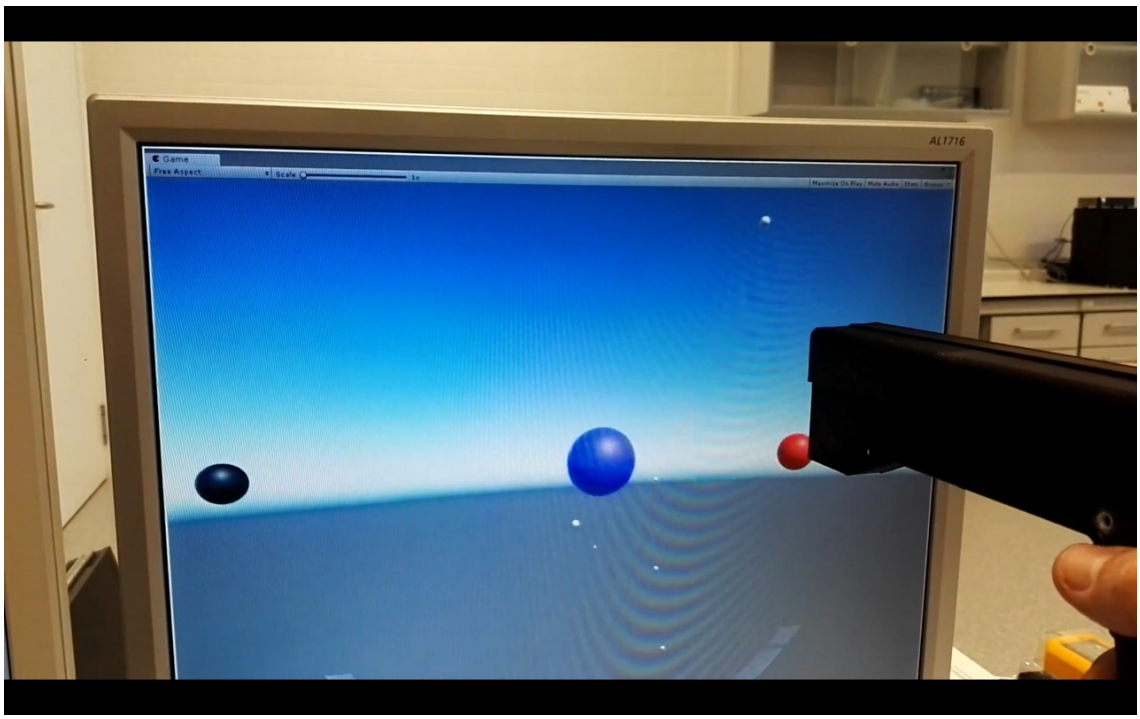

Figura 4.2.2.1 Aplicación para recepción datos MPU-6050.

Se estudiaron diferentes tipos de filtrado, hasta llegar a la solución adoptada, que fue implementar un filtro de Kalman [2] en su implementación más simple para poder adaptarla a Arduino. El filtro de Kalman [2], es un filtro adaptativo. Se parte de un estado inicial, donde el error es muy grande. Conforme recibimos valores del sensor IMU MPU6050, los valores se van estabilizando, hasta obtener valores que se acercan en gran medida a los valores reales. Se estudiaron diferentes proyectos([3],[4],[5]) para su implementación. Uno de los principales problemas era el modo en el que se recogían los datos y se enviaban al ordenador, ya que en la mayoría de los casos esta implementación requería de una velocidad de transmisión elevada. En nuestro caso la comunicación con el dispositivo móvil, tenía una serie de limitaciones debido a la velocidad de conexión del *NodeMCU*.

Se optó por una implementación compartida del filtro de Kalman [2], donde parte del procesamiento lo realizaba el *NodeMCU* y la parte más compleja la implementaba el dispositivo móvil, de esta forma se requería un menor ancho de banda para la transmisión de los datos entre la pistola y la aplicación.

La comunicación entre el *NodeMCU* y el *IMU MPU 6050* , se realiza a través del bus *SPI*. La dirección utilizada por este *IMU* es la 0x68. Es necesario configurar correctamente el *IMU* para su buen funcionamiento, se debe configurar el nivel de resolución tanto del acelerómetro, como del giroscopio [6]. Una vez configurado los parámetros, es posible la lectura de los datos del *IMU*, se procesan y se envían al servidor configurado en el dispositivo móvil; el cual en uno de los hilos ejecución está esperando los datagramas *UDP* en el puerto 9000. Los datos se envían en formato *JSON*, el primer parámetro corresponde a si se está disparando o no la pistola, enviando un 1 o un 0. El siguiente parámetro corresponde con el valor de la aceleración en el eje Y, este dato es un dato numérico, por lo que no corresponde con un ángulo determinado. Los siguientes datos enviados, son los valores obtenidos de la brújula, que corresponde con valores en los ejes magnéticos (X,Y,Z), como ya se ha comentado anteriormente es imprescindible realizar un filtrado de los datos obtenidos del sensor. Estos datos son recogidos por la aplicación y transformados en ángulos, se explicará más detenidamente el proceso en siguientes apartados.

Como ya se ha comentado anteriormente la pistola se conecta al punto de acceso que se encuentra configurado en el dispositivo móvil. Se ha incluido la posibilidad de realizar modificación en la red a la que se conecta por defecto la pistola. Para poder configurar los parámetros *SSID* y contraseña, de la red a la que se quiere conectar, se debe pulsar el botón del gatillo antes del encendido de la pistola. Se mantiene apretado el gatillo durante 5 segundos y de esta forma la pistola, entra en el modo de configuración.

Una vez la pistola entra en modo configuración, crea un punto de acceso propio. Esto se realiza con la función:

```
boolean result = WiFi.softAP("TURMANDREAMS", "");
```

La red wifi creada es TURMANDREAMS, para que el usuario se conecte a ella. Además se crea un *web server* en el *NodeMCU*, al que el usuario puede acceder. Esto se realiza con las funciones:

```
WiFiServer server(80);

server.begin();
```

De esta forma el usuario podría introducir los datos *SSID* y contraseña. La petición que debe enviar el cliente web debe ser del tipo:

http://192.168.0.1/config@turmandreams;turmandreams;192.168.0.100;

Se debe modificar la *URL*, con los parámetros adecuados. Introduciendo en primer lugar la dirección IP del punto de acceso creado por la pistola, seguido del directorio /config@. A continuación se introduce la *SSID* de la red que vamos a crear y la contraseña, separados por ";". Por último introducimos la dirección IP de dispositivo móvil, para que se envíen correctamente los paquetes *UDP*.

Una vez el cliente web envía la nueva la configuración, el *NodeMCU* almacena los datos en su *EEPROM.* Esto lo realiza con el siguiente código:

```
EEPROM.begin(512);
String dato="";
addr=0;

EEPROM.put(addr,cadena.length());addr++;

for(i=0;i<cadena.length();i++){
  EEPROM.put(addr,cad[i]);addr++;
}

EEPROM.end();
```

Figura 4.2.2.2 Código guardar datos EEPROM.

Después de guardar los datos en la *EEPROM*, se procede al reinicio del *NodeMCU*. La pistola arranca de nuevo y recoge los nuevos datos almacenados en la *EEPROM*, quedando configurada la nueva conexión. Podemos observar los modos de operación en el siguiente flujograma:

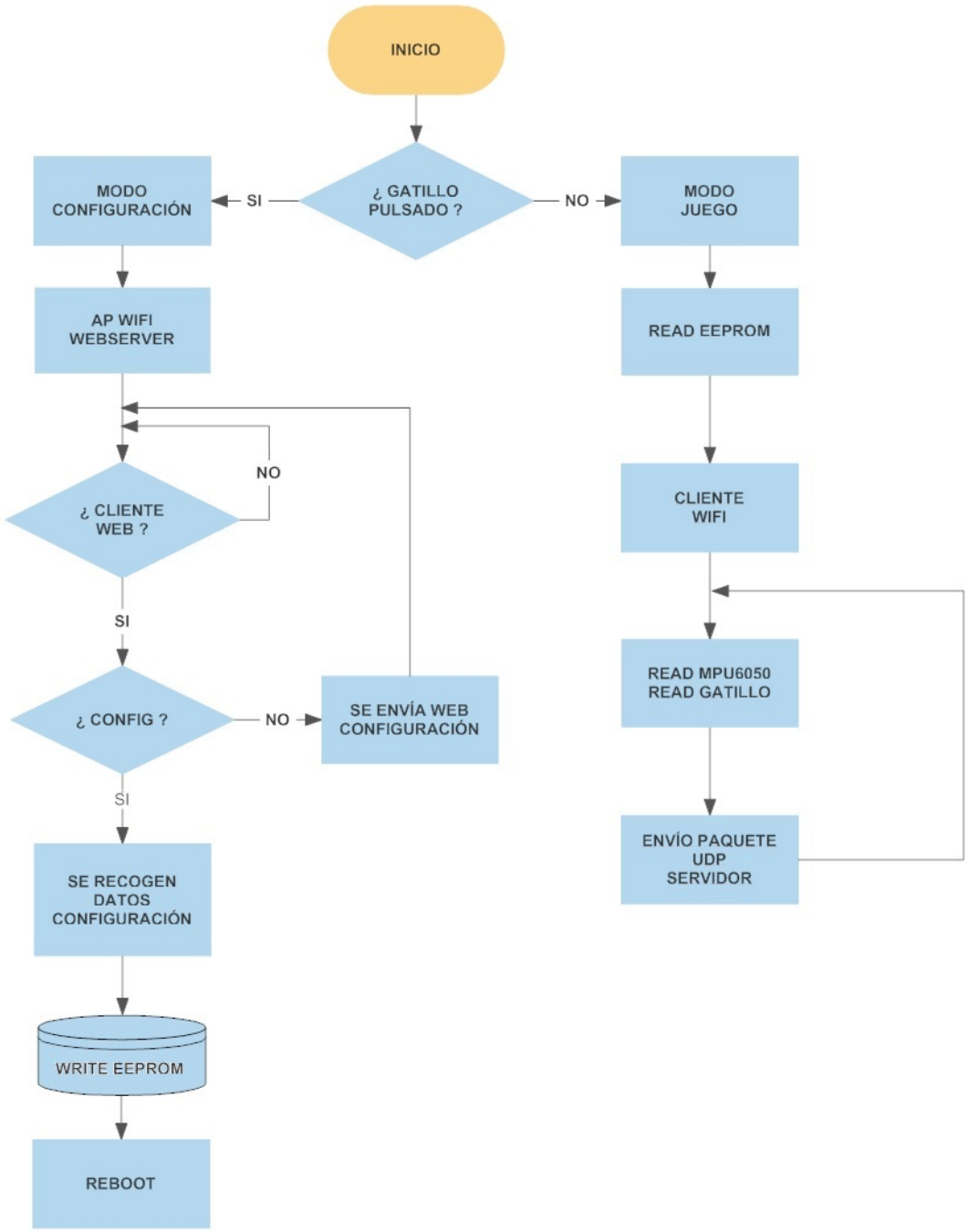

Figura 4.2.2.3 Diagrama flujo firmware pistola.

Como puede observase en el flujograma, al iniciarse la pistola se comprueba si se encuentra pulsado el gatillo. De esta forma la pistola entraría en modo configuración, como ya se ha comentado anteriormente. Si el gatillo no se encuentra activo, la pistola entraría en modo juego. Este modo de funcionamiento es el modo por defecto.

En modo juego, en primer lugar se leen los datos almacenados previamente en la *EEPROM,* estos datos son: *SSID*, contraseña de la red *wifi* y dirección IP del dispositivo móvil, que ejecuta el videojuego de realidad virtual. Seguidamente el dispositivo *NodeMCU* se conecta la red indicada y comienza la lectura de los datos proporcionados por el sensor *MPU6050*. Además, se realiza una comprobación del estado del gatillo, para indicar a la aplicación si se está disparando o no. Se envían los datos al dispositivo móvil como se ha comentado anteriormente. Como puede observarse en el flujograma este proceso de lectura y envío de datos se realiza indefinidamente, hasta que se apague la pistola.

Se pasa a describir el conexionado interno de la pistola. En este esquema podemos observar cómo se realiza la alimentación del *NodeMCU* y del *IMU*. Como alimentación se ha utilizado una batería *Litio-ION* de 3.7v, la cual es recargada gracias a un circuito de carga TP4056, específico para este tipo de baterías. Se ha incluido un led indicador de encendido. Para el gatillo se ha utilizado la entrada digital D8. El sensor MPU6050, se ha conectado al puerto *SPI* por defecto del *NodeMCU*.

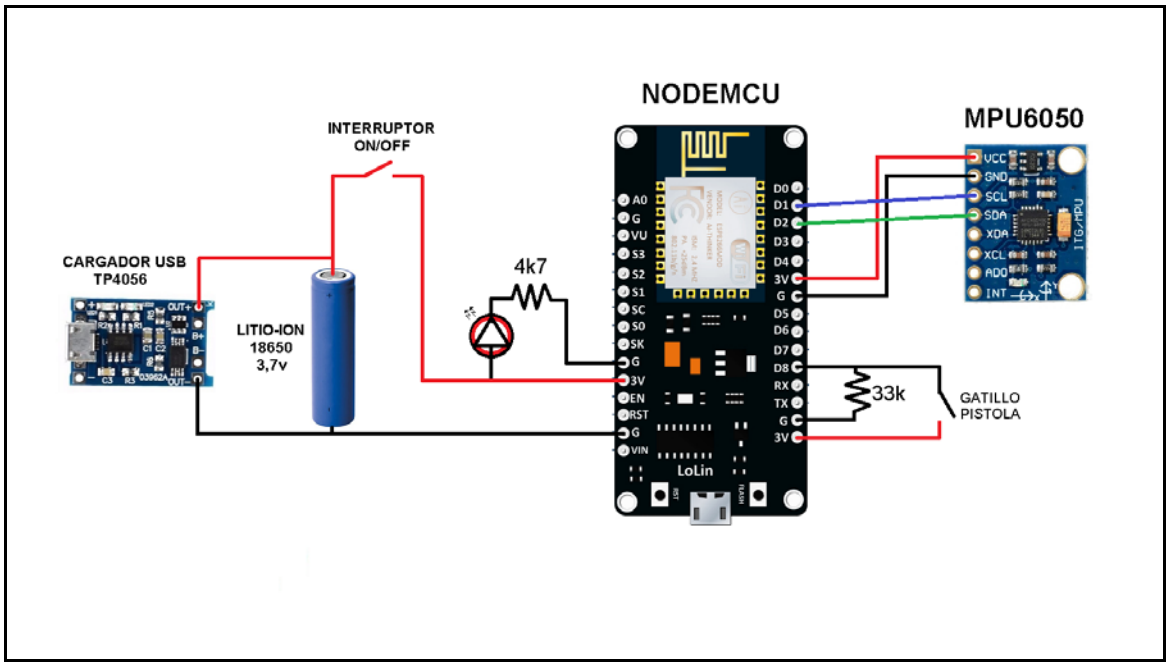

Figura 4.2.2.4 Esquema conexionado circuito pistola.

También cabe destacar que la pistola ha sido fabricada con una impresora 3D. Para realizar el diseño de la pistola se ha utilizado *SketchUp*. Han sido necesarios realizar 2 modelos, un modelo de la parte superior y otro modelo de la parte inferior. El tiempo de impresión de cada parte, de la pistola, ha sido de 6 horas. Se ha utilizado la impresora Anet A8 y se ha impreso en PLA, material que es biodegradable y puede ser reciclado. En la siguiente figura se puede observar el programa de edición 3D.

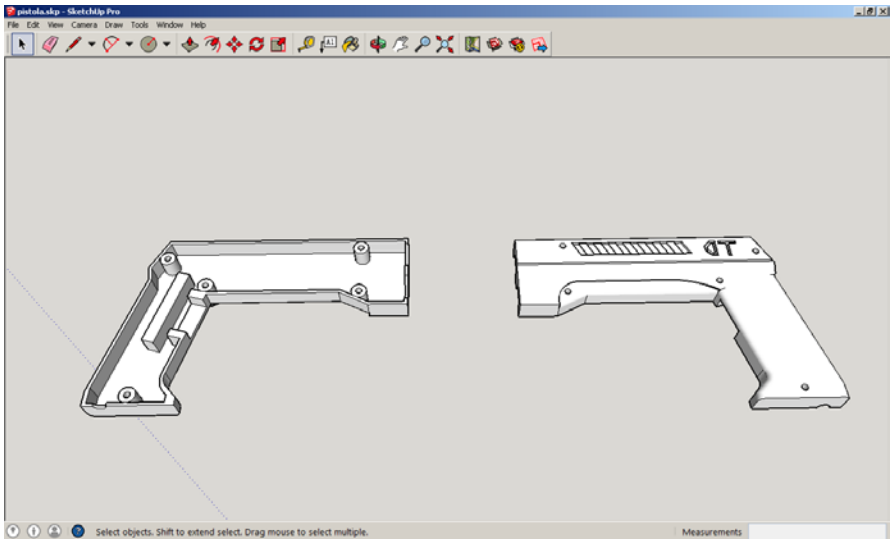

Figura 4.2.2.5 Diseño modelo 3D pistola.

Este modelo se ha ensamblado virtualmente y se ha incorporado a la aplicación de realidad virtual, de tal forma que la pistola que se utiliza en el juego, corresponde exactamente con la pistola física.

Figura 4.2.2.6 Fotografía pistola impresa en 3D.

4.2.3 Dispositivo de posicionamiento

Una parte importante de este TFG es conseguir detectar la posición del jugador dentro de la habitación para permitir que el usuario se desplazase por el mundo virtual, aumentando así la inmersión. Para realizar esto, se utilizaron diferentes métodos.

Una de las primeras pruebas fue realizando triangulación de la señal wifi de la pistola respecto a unas balizas colocadas en las 4 esquinas de la habitación. Cada baliza contenía un *NodeMCU*, el cual se configuraba en modo promiscuo, de tal forma que iba rastreando las diferentes señales wifi. Se localizaba la señal que emitía el dispositivo móvil, configurado como punto de acceso y se observaba la señal que se recibía. Se recogían los datos de las 4 balizas y se calculaba la posición del dispositivo móvil, dentro de la habitación.

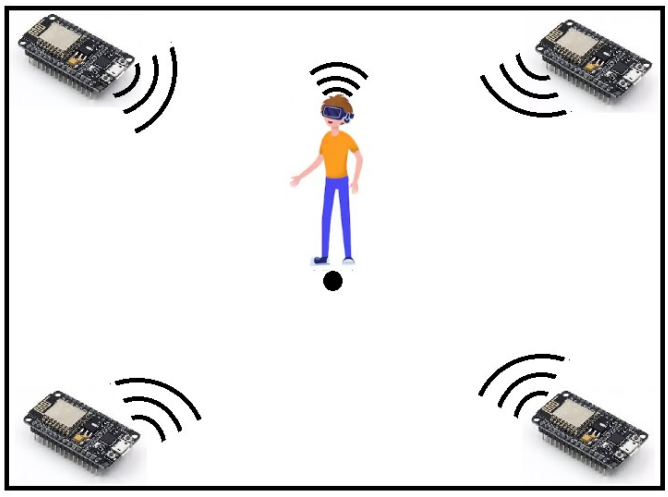

Figura 4.2.3.1 Posicionamiento con balizas.

El principal problema de esta implementación, es que el sistema es estable cuando el dispositivo móvil, mantiene la posición de horizontalidad y verticalidad, pero en nuestro sistema de realidad virtual, el móvil se encuentra colocado en la cabeza del jugador y al tratarse de un videojuego de acción, el jugador tiende a mover mucho la cabeza. Esto provocaba que los datos no fuesen fiables, ya que una inclinación de la cabeza de 10 grados de diferencia provocaba desplazamientos de varios metros dentro del entorno virtual, por lo que era difícil, estabilizar al jugador dentro del entorno.

El segundo sistema de posicionamiento que se probó fue utilizando los sistemas de reconcomiendo de imagen. La idea era utilizar 2 *Raspberry Pi* de tal forma que cada una de las *Raspberry* tuviese conectada una webcam, se procesaba la imagen y se calculaba el punto medio donde se encontraba el jugador. Es decir, una *Raspberry Pi* calculaba la posición en el eje X y otra *Raspberry Pi* calculaba la posición en el eje Y. De esta forma se obtenía X e Y, estos datos se enviaban al dispositivo móvil, y modifican la posición dentro del mundo virtual.

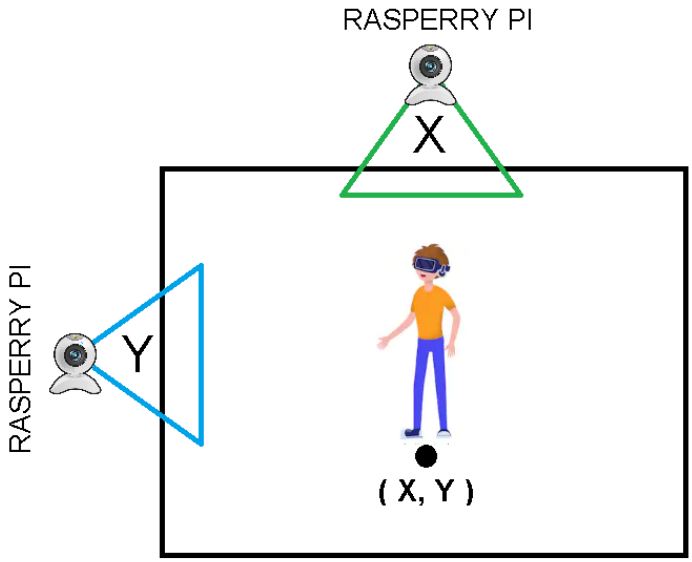

Figura 4.2.3.2 Posicionamiento Jugador.

Este sistema tenía varios problemas, uno de ellos era que el desplazamiento del jugador estaba limitado por el espacio de la habitación en la que se encontraba. Además para este videojuego es necesario poder tener un libre desplazamiento por un amplio espacio virtual. El sistema se comportaba correctamente cuando el jugador se movía hacia delante, pero tenía problemas cuando el jugador se desplazaba hacia atrás, por lo que provocaba una mala experiencia para el jugador.

Finalmente se optó por utilizar únicamente una *Rapsberry Pi*, que detectase únicamente si existía movimiento en la imagen capturada o no. Si existe movimiento por encima de un umbral determinado el sistema interpreta que el jugador se está moviendo y provoca su movimiento dentro del entorno virtual. En este caso el jugador siempre se desplaza hacia delante, la dirección del desplazamiento está determinada por la dirección a la que está mirando el jugador. Este sistema permite su utilización en espacios reducidos, además genera una experiencia inmersiva para el jugador.

Para la realización de la solución final fue necesaria la instalación, en el sistema operativo *Raspbian*, de los paquetes del intérprete de *Python* y las librerías de *OpenCV*. En el Anexo 10.3, se incluyen un manual de instalación.

Se pasa a explicar con más detenimiento el funcionamiento de la solución adoptada. En primer lugar se cargan todas las librerías necesarias para utilizar *OpenCV*. Se inicializa la lectura de la webcam, conectada a la *Raspberry*. Dentro del bucle principal se realiza la captura de la imagen. Se reescala la imagen, a un tamaño de 400 pixeles de ancho y 300 pixeles de alto, para poder analizarla de forma más rápida. Posteriormente se transforma la imagen a escala de grises. Se realiza una comparación entre la imagen actual y la imagen previa, con la función:

```
(cnts, _) = cv2.findContours(thresh.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
```

De esta forma se obtiene el número de puntos diferentes entre las imágenes. Si el número de puntos es significativo, se envía un paquete UDP al dispositivo móvil, con número de puerto 9000, indicando que se ha producido una variación en la imagen. Se puede observar con más detenimiento el código fuente de la aplicación en el Anexo 10.5.

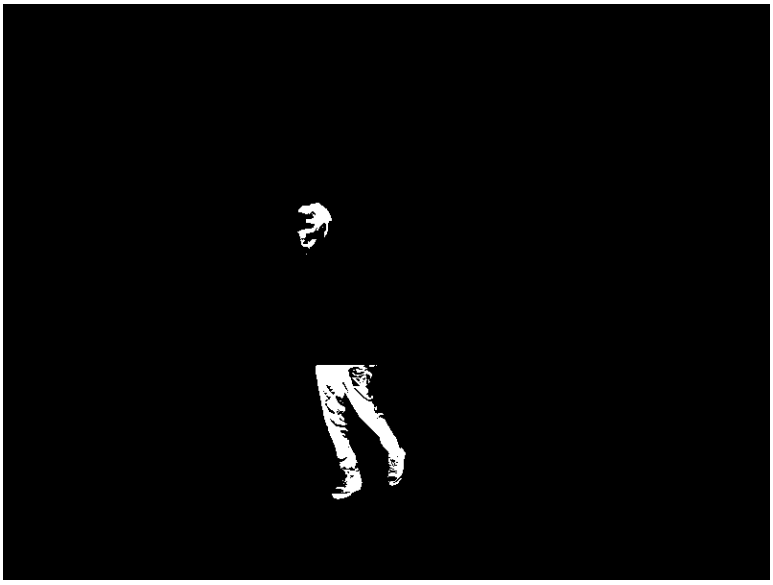

Figura 4.2.3.3 Detección movimientos OpenCV.

4.2.4 Aplicación del dispositivo móvil

El móvil se configura como punto de acceso wifi, para crear la red a la que se conectarán pistola y Raspberry Pi.

Figura 4.2.4.1 Configuración zona wifi.

Como podemos observar en la figura anterior, el *SSID* por defecto utilizado es @esplinares_uja, no se ha configurado seguridad, por lo que no es necesario introducir contraseña. La red local que se crea en el caso de la figura es del tipo 192.168.43.X. El dispositivo móvil, toma como dirección IP: 192.168.43.1 .

La aplicación del dispositivo móvil, ha sido realizada, como se ha comentado anteriormente, con el entorno de desarrollo Unity. Se ha desarrollado el videojuego para poder funcionar en sistemas Android, aunque desde Unity es fácilmente exportable a otros sistemas operativos.

Dentro del entorno de desarrollo Unity cabe destacar los siguientes apartados:

- *Scene*: Un proyecto de Unity se descompone en escenas, dentro de una escena se incorporan los diferentes objetos que la componen. En el apartado gráfico Scene, se pueden ubicar los objetos dentro del entorno virtual, colocándolos en la posición seleccionada.

- *Game*: En este apartado se muestra el resultado final, para poder visualizarlo como se vería en el dispositivo.
- *Hierarchy*: Dentro del apartado jerarquía se incluyen los objetos que van a estar en la escena. Cada objeto puede tener asociado varios objetos, de tal forma que se define una jerarquía asociada a cada objeto.
- *Project*: Dentro de la carpeta se incluyen todos los archivos que pueden ser utilizados en el proyecto, como pueden ser: audios, prefabs, imágenes, texturas, materiales, terrenos, script, etc.
- *Console*: Este apartado es utilizado para realizar una monitorización de la aplicación y poder detectar diferentes fallos.
- *Animator*: Este apartado es utilizado para poder dotar a los objetos 3D de diferentes animaciones, en posteriores secciones explicaremos con más detenimiento este apartado.
- *Inspector*: En este apartado se puede modificar, añadir o eliminar propiedades asociadas a objetos, audios, etc.

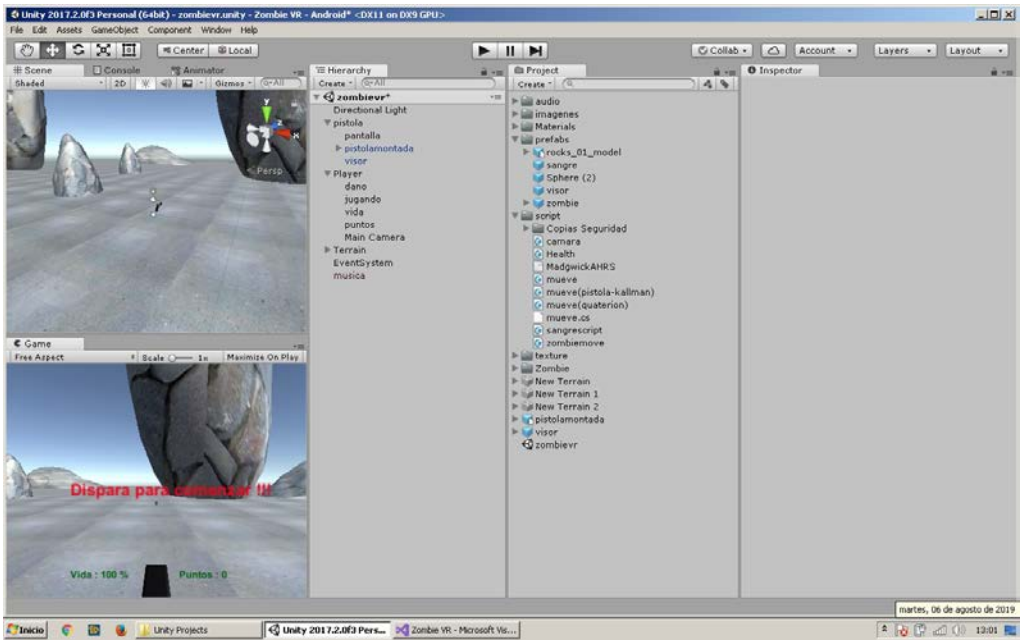

Figura 4.2.4.2 Entorno de desarrollo Unity.

Pasamos a explicar los objetos más destacados de nuestra aplicación. El videojuego es un juego de disparos en primera persona, en el que el jugador se encuentra en un mundo virtual, para ello debemos definir cómo será el entorno. Para crear nuestro mundo virtual necesitamos un terreno, el cual tiene asociada una textura determinada, se puede observar su propiedades en la Figura 4.2.4.3.

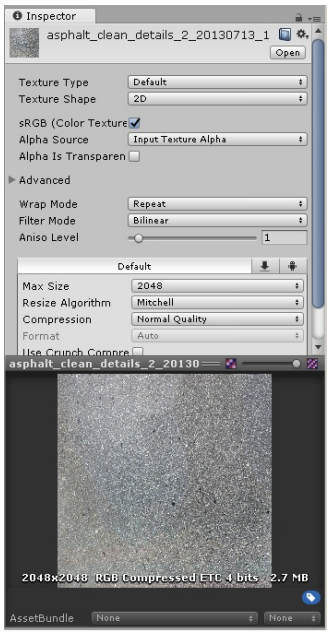

Figura 4.2.4.3 Propiedades del Terreno (Unity).

Asociado al terreno, se han incorporado varios objetos para dar más detalle al entorno virtual, se han incluido varios modelos de roca, estos modelos se han incorporado de la librería *Standard Assets,* que se puede descargar gratuitamente desde *Asset Store* de Unity.

Dentro de la escena cabe destacar el objeto pistola, este objeto corresponde con la pistola dentro del videojuego. Este objeto tiene asociados 2 objetos, visor y pistolamontada. El visor corresponde con el punto de mira de nuestra pistola, indica al jugador la dirección de nuestro disparo. Pistolamontada es el modelo 3d de nuestra pistola, diseñada previamente con *Sketchup*.

El objeto pistola, tiene asociado el script mueve. Este script es el encargado de recoger la posición de la pistola, realizar los cálculos necesarios para cambiar la posición de la pistola dentro del entorno virtual. Además realiza el posicionamiento del visor dentro del entorno. Pasamos a explicar en detenimiento el script mueve.

El script mueve carga la función *Start()*, cuando es llamado por primera vez. Dentro de la función *Start()*, cargamos el servidor UDP en el puerto 9000, este servidor será el encargado de recibir los datos de la pistola. Para poder recibir los datos lo más rápidamente posible, se carga un hilo de ejecución, donde se recibirán los datos. Esto se hace para poder independizar la recepción de información, de la actualización del script mueve.

Los datos que se reciben de la pistola son almacenados en las variables:

- “disparo”: Su valor es 0 si no se está disparando. Si se está disparando su valor es 1.
- “iay”: Dato necesario para el cálculo de la elevación de la pistola. Está asociada a la aceleración de la pistola en el eje Y.
- “imx”: Dato necesario para el cálculo de la dirección a la que apunta la pistola. Este dato es obtenido de la brújula de la pistola. Está asociada al magnetómetro en el eje X.
- “imy”: Dato necesario para el cálculo de la dirección a la que apunta la pistola. Este dato es obtenido de la brújula de la pistola. Está asociada al magnetómetro en el eje Y.
- “imz”: Dato necesario para el cálculo de la dirección a la que apunta la pistola. Este dato es obtenido de la brújula de la pistola. Está asociada al magnetómetro en el eje Z.

Estos datos se filtran y se obtiene un valor bastante fiable de la posición en la que se encuentra la pistola, para ellos es necesario pasar los valores obtenidos a grados, esto se realiza con el código de la siguiente figura:

```
double rad = Math.Atan2(antimx,antimy);

grados = rad * 57296;
grados = grados / 1000;

grados+= 180;
if (grados < 0.0) { grados += 360.0; }
if (grados > 360) { grados -= 360.0; }
```

Figura 4.2.4.4 Código para pasar a grados.

Dentro del script mueve, cabe destacar la función *Update()*, la cual es llamada en cada actualización de la pantalla. Esta función es la encargada de actualizar la posición de nuestra pistola virtual dentro del entorno, esto se realiza con:

```
transform.rotation = Quaternion.Euler((float)-pitch, (float)grados, 0);
```

Como se puede observar, se utilizan los valores de las variables pitch y grados, calculados en el hilo de ejecución. Esta función realiza la rotación dentro del entorno virtual.

Además en la función *Update()*, se realiza el posicionamiento del visor de la pistola. En la mayoría de los juegos *FPS,* habitualmente el visor se encuentra posicionado en el centro de la pantalla, en este caso, no es así, ya que el jugador puede disparar hacia un lado y estar mirando hacia el otro lado, de tal forma que el visor es independiente de la cámara principal. Esto obliga a realizar la función *Raycast* [7]. Unity genera un vector que parte desde la pistola, hasta un punto lejano en la dirección en la que apunta la pistola. De esta forma se posiciona el visor dentro del vector Raycast [7], a una distancia de la pistola determinada. Si dentro de este vector se encuentra algún objeto, se posiciona el visor en la intersección.

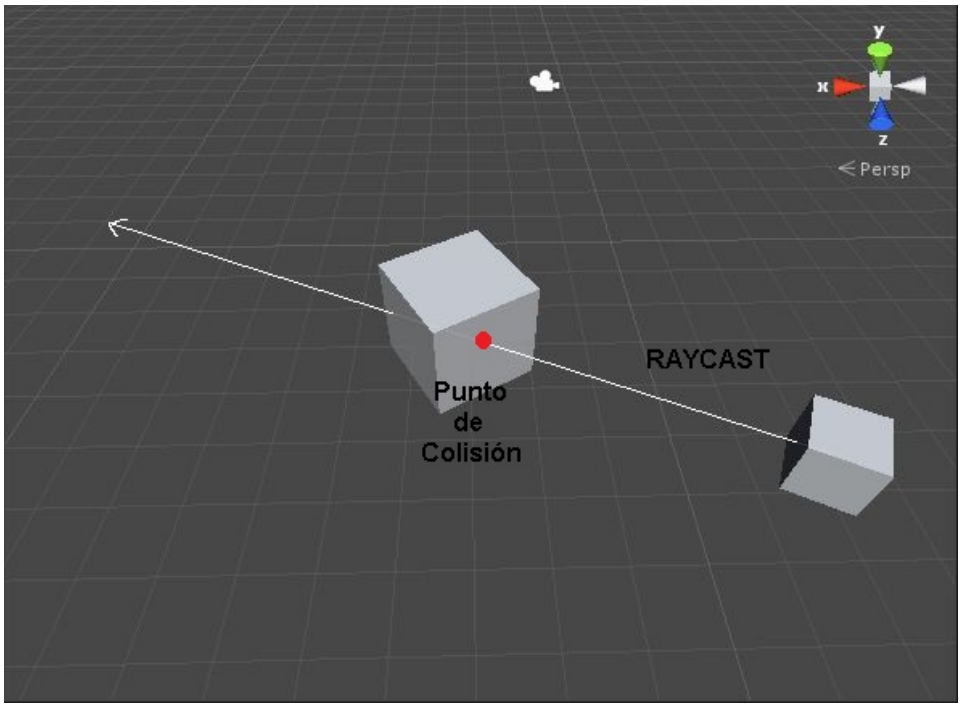

Figura 4.2.4.5 Raycast.

Dentro de *Update()*, se lleva el control de los disparos a los zombies, actualizando sus puntos de daño. Cada vez que se pulsa para disparar se decrementa el contador de disparos, si se utilizan 10 disparos seguidos, es necesario recargar la pistola. Para realizar la recarga es necesario apuntar el cielo del entorno virtual y disparar.

Figura 4.2.4.6 Jugador realizando recarga.

Dentro de la escena principal cabe destacar el objeto Player, este objeto corresponde con el jugador. Tiene asociado la *Main Camera* (cámara principal), así como los objetos daño, vida, jugando y puntos. Estos objetos corresponden con los indicadores que el jugador visualiza en pantalla, para poder llevar un control de la partida.

Figura 4.2.4.7 Detalle objetos *Text Mesh* dentro de la escena.

Estos objetos son del tipo Text Mesh (malla de texto). Se puede observar las características en la siguiente figura:

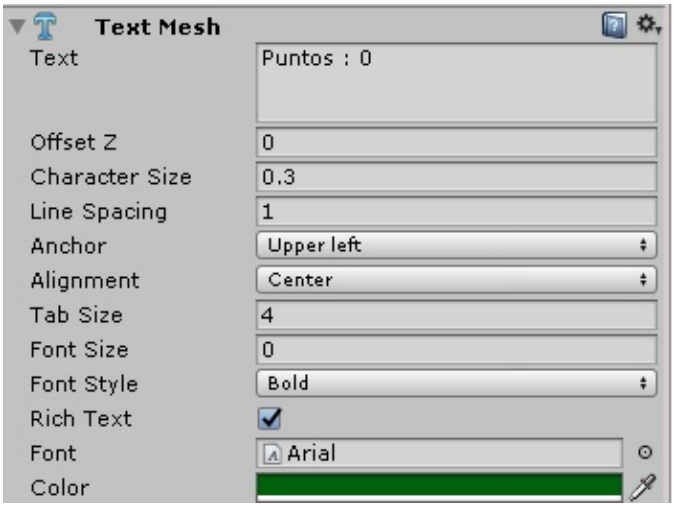

Figura 4.2.4.8 Propiedades objetos *Text Mesh*.

El objeto Player tiene asociado el script cámara, que realiza modificaciones sobre los objetos *Text Mesh*. Además controla la posición y orientación de la cámara principal. Se explicará con detalle las funciones que se realizan dentro del script.

Cuando en Unity se realiza la compilación de la aplicación, permite que automáticamente se realice el control sobre los movimientos de la cabeza del jugador. De tal forma que cuando el jugador, cambia la posición de la cabeza, este modifica el punto de vista dentro del entorno virtual. En este caso se observó que el control se realizaba de forma muy brusca, provocando excesivas vibraciones en la visualización, por lo que se hizo necesario realiza un control de los movimientos de la cabeza con un sistema de filtrado propio. Esta estabilización del control de los movimientos de la cabeza se realiza en el script cámara.

Dentro del script cámara, cabe destacar la función *Start()*, donde se realiza una inicialización de la lectura del acelerómetro del dispositivo móvil, así como la inicialización de la lectura de la brújula.

Dentro del script cámara, la función *Update()*, se ejecuta cada ciclo de visualización. Se realiza en primer lugar una actualización de los Text Mesh Vida y puntos. Si la variable vida es menor de 0, se inicializa el juego, mostrando en pantalla la frase “Dispara para comenzar !!!”. Además se eliminan todos los objetos zombies, creados en la escena.

Si la partida está activa se van incorporando a la escena los objetos zombies, hasta un máximo de 10 zombies. La posición de los zombies sobre el entorno virtual se hace de forma aleatoria. Esto se realiza con las funciones:

```
var x = 0; while (x == 0) { x = Random.Range(-5, 5) * 20; }

var y = 0; while (y == 0) { y = Random.Range(-5, 5) * 20; }

var spawnPosition = new Vector3(x, -5.0f, y);

var spawnRotation = Quaternion.Euler(0.0f, Random.Range(0, 180), 0.0f);

var enemy = (GameObject)Instantiate(zombie, spawnPosition, spawnRotation);
```

Como se puede observar para la rotación del objeto zombie, es necesario utilizar la función Quaternion.Euler , igual que pasaba con el objeto pistola.

Para realizar el control de la dirección y elevación de la cámara principal, se utilizan los valores ofrecidos por el dispositivo móvil, como son:

```
float elevacion = Input.acceleration.z * -90.0f;
float roll = Input.acceleration.x * -80.0f;
float ang = Input.compass.magneticHeading;
```

Estos datos tienen una gran fluctuación, por lo que es necesario realizar un filtrado, para poder estabilizar los valores, de tal forma que se produzcan las menos vibraciones posibles en la visualización del jugador. Para realizar la estabilización de los valores se realiza siguiente la siguiente ecuación:

$$datoanterior = (datoanterior * (1 - k)) + (dato * k);$$

Donde datoanterior, es el valor calculado, dato corresponde con el valor obtenido en el instante actual y k corresponde con un valor que puede cambiar entre 0.03 y 0.3. Si datoanterior es muy diferente de dato, k adquiere el valor de 0.3, si la diferencia entre los datos es poca, k adquiere el valor 0.03. Estos valores han sido optimizados para conseguir una experiencia óptima. Este esquema se utiliza para calcular elevación, roll y ang.

Dentro de la función *Update()*, se realiza el control del posicionamiento del Player dentro del entorno virtual, para que el jugador pueda moverse por el terreno. Como ya se comentó anteriormente, esto lo realizamos con la Raspberry Pi, que envía al dispositivo móvil si el jugador se está moviendo o no. Para ello se utiliza la variable anda, si su valor es 1 indica que el jugador está en movimiento y se modifica la posición del Player. Se desplaza al objeto Player gradualmente en la dirección en la apunta la *Main Camera*. Cuando el jugador está en movimiento se inicializa la posición de la pistola en el centro de la visualización, para mejorar el realismo en la escena.

Como se ha comentado anteriormente en la función *Update()* se incorporaban a la escena los objetos zombies. Estos objetos son modelos descargados del Asset Store y se le ha añadido los scripts zombiemove y Health.

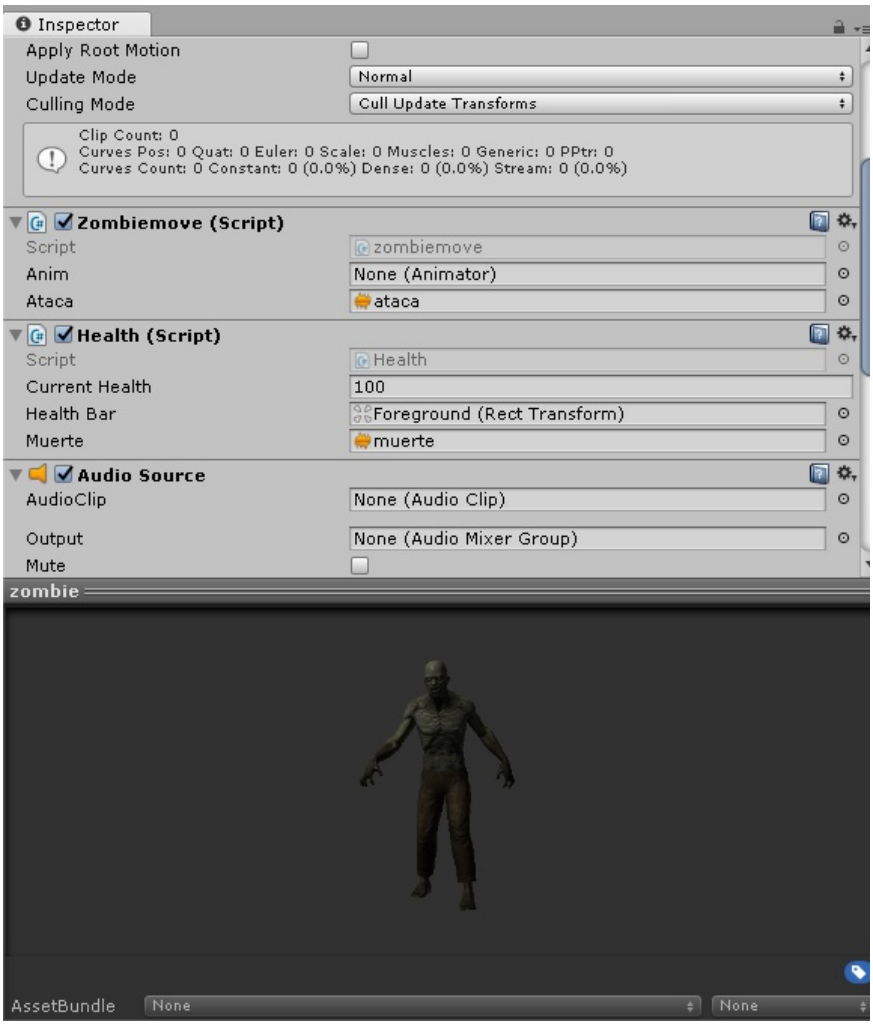

Figura 4.2.4.9 Propiedades objeto zombie.

El script Health se encarga de modificar la barra de vida del zombie, para dar esa información al jugador. La barra de vida consiste en una barra de color verde y una barra de color rojo. Cuando la vida del zombie es del 100% la barra de vida es de color verde, cuando va disminuyendo la vida del zombie se va reduciendo el tamaño de la barra de color verde y se va aumentado el tamaño de la barra de color rojo.

Figura 4.2.4.10 Barra de vida zombie.

El script zombiemove realiza un control de los movimientos del zombie dentro del entorno, así como un control sobre las animaciones del objeto zombie. Para desplazar al zombie por el mapa, se orienta al objeto zombie hacia el punto donde se encuentra el objeto Player y se va disminuyendo la distancia entre ellos.

Por defecto, la animación predeterminada, es la animación de andar. Si la función Update(), detecta que el jugador se encuentra a una distancia menor de 9, se activa la animación de atacar, además esto activa el TextMesh del jugador que indica que está disminuyendo la vida y provoca vibraciones en la visualización del jugador.

Se explicará a continuación el sistema de Unity, para el control de las animaciones. En la sección Animator, se pueden incorporar las diferentes animaciones asociadas a un objeto. En esta sección se define la secuencia en la que las animaciones se ejecutan. La animación comienza en el estado *Entry* (entrada), a partir de este estado se van definiendo los estados siguientes, esto se realiza a través de unas flechas que indican los diferentes saltos entre estados. Para el caso de las animaciones del zombie se sigue el siguiente esquema:

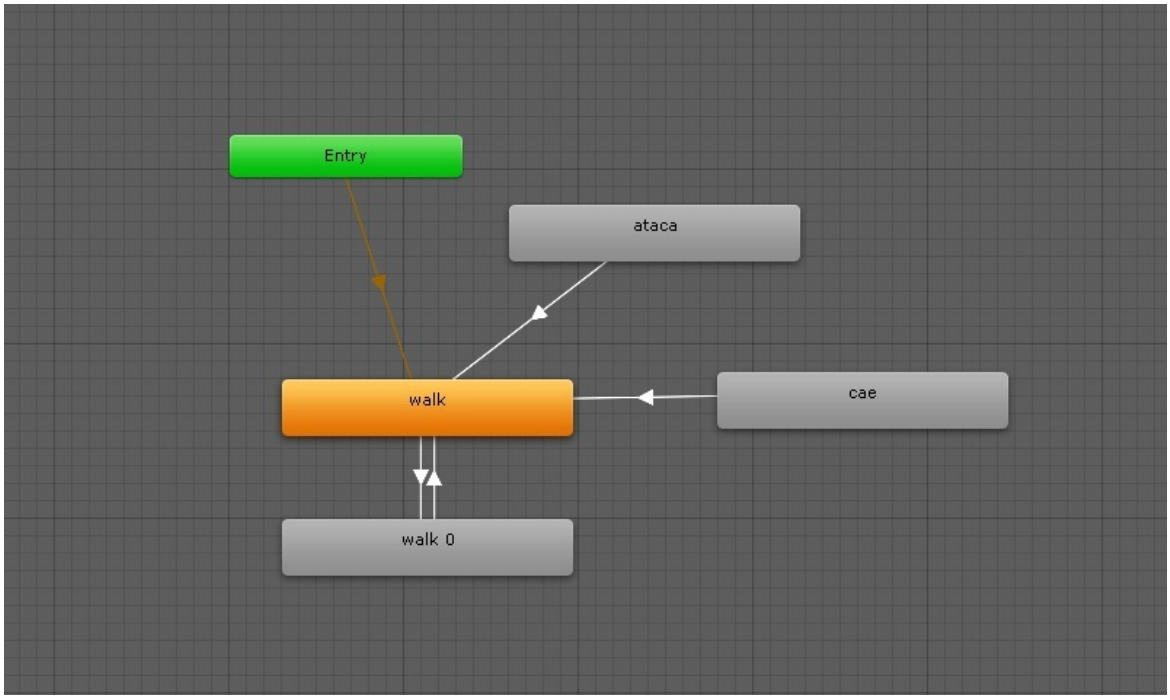

Figura 4.2.4.11 Sección *Animator* Unity.

El salto entre estados está predefinido, pero puede probarse el cambio de estado con eventos que se ejecutan en los scripts. De tal forma que se puede ejecutar la animación de ataque, si el jugador se encuentra muy cerca. Esto se realiza con la función:

```
anim.Play("ataca");
```

Una vez se ejecuta la animación ataca, vuelve a la animación walk, como puede observarse en la Figura 4.2.4.11. De esta forma se puede ejecutar la animación deseada. Las animaciones asociadas al objeto zombie han sido predefinidas en la librería *Standard Assets*, aunque pueden ser modificadas desde Unity en las propiedades de las animaciones.

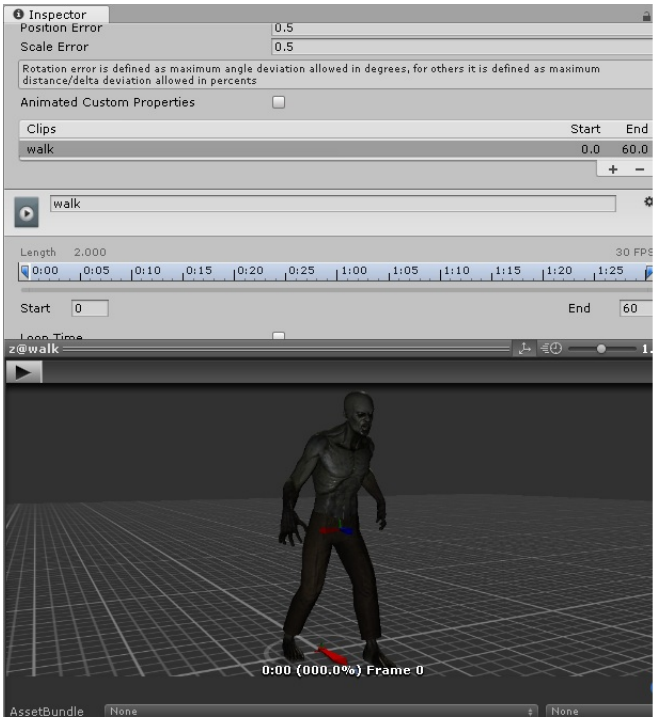

Figura 4.2.4.12 Propiedades animación *walk*.

5. RESULTADOS Y DISCUSION

Durante la realización de este TFG se han ido superando diferentes dificultades, para conseguir un resultado óptimo. Se partía desde una idea inicial, replicar un sistema de realidad virtual a bajo coste. Para ello ha sido necesaria la utilización de diferentes tecnologías.

En primer lugar se experimentó con el sensor MPU6050, que era capaz de enviar datos del acelerómetro, giroscopio y del magnetómetro. Estos datos eran muy variables por lo que fue necesario su procesamiento. Se realizó un estudio de los diferentes filtrados posibles. Se realizaron pruebas de varios filtros, hasta llegar a la solución obtenida. De esta forma se obtuvo unos datos bastante estables de la posición del sensor.

Una vez se obtenían los datos desde el sensor, se realizaron pruebas para trasladar la posición del sensor a un entorno virtual simple. Para ello se hizo necesaria la creación de la pistola física, para posicionar correctamente el sensor y obtener una correcta interpretación en el mundo virtual. Una vez construido el modelo 3D, se introdujo en el entorno. Se diseñó e integró el circuito electrónico dentro de la pistola.

Una vez los movimientos de la pistola eran enviados correctamente a la aplicación, se observó que el método con el que se procesaban los datos, no era lo bastante rápido. Por lo que se probaron diferentes métodos, para su procesamiento, modificando parámetros en el envío y recepción. Después de varias modificaciones se obtuvo un funcionamiento óptimo.

Se pasó seguidamente a mejorar el entorno virtual, incluyendo objetos y texturas, para conseguir que la aplicación fuese lo más atractiva posible. Para ello fue necesario, un estudio de las diferentes modos de funcionamiento de Unity. Una parte que cabe destacar es el procesamiento de las animaciones de Unity. Realizándose varios proyectos previos para su correcto entendimiento.

Se comenzó a mejorar el funcionamiento del juego, incluyéndole el sistema de control de vida del jugador y de los zombies. Fue necesario, entender como Unity realizaba el control de las colisiones. Fue necesario implementar los movimientos de nuestro puntero dentro del entorno virtual. En este punto, el jugador, siempre se encontraba en el centro del entorno virtual. Era necesario estudiar la forma de trasladar los movimientos del jugador al entorno virtual. Se realizaron diferentes pruebas para conseguir realizar el posicionamiento, como ya se ha comentado anteriormente, hasta llegar a la solución adoptada.

Por último, se realizó una optimización de todo el sistema, para evitar posible fallos. Durante este periodo se realizaron varias experiencias con visitantes a los laboratorios. De estas demostraciones se observaron algunos fallos, que fueron subsanados. Las experiencias con los visitantes han sido muy positivas.

Figura 5.1 Visita IES Presentación

Como resultado de este TFG se ha diseñado y construido un sistema de realidad virtual con comunicación entre dispositivos en tiempo real. Ha sido necesario, utilizar diferentes tecnologías: Unity, librerías VR, Asset Store, Visual Studio, Arduino, NodeMCU, sensor IMU MPU6050, software de edición 3D, Raspberry Pi, Python, OpenCV y Chromecast.

Se ha conseguido implementar un sistema totalmente funcional para las demostraciones, realizando una aplicación de gran interés para los usuarios. El sistema se ha utilizado en numerosas visitas a los laboratorios docentes del departamento de Ingeniería de Telecomunicación.

Figura 5.2 Visita SAFA – Linares.

Una de las grandes ventajas del sistema implementado es su portabilidad, pudiendo utilizarse en cualquier ubicación, sin necesidad de una infraestructura previa. Se ha implementado, de forma que sea fácil su utilización, por parte de cualquier usuario, sin necesidad de unos conocimientos previos sobre sistemas de realidad Virtual.

Actualmente existen una gran cantidad de dispositivos de realidad virtual, que ofrecen una magnífica experiencia. Estos sistemas suelen ser sistemas de un coste elevado y necesitan una infraestructura previa para su utilización. El sistema planteado ofrece la ventaja de ser un sistema de bajo coste y sin necesidad de un equipamiento previo.

6. CONCLUSIONES

El principal objetivo de este TFG era conseguir realizar un sistema de realidad virtual para dispositivos móviles. Este sistema debía utilizar diferentes tipos de sensores y ser capaz de interactuar con una aplicación en tiempo real. Para la realización de las comunicaciones se debía utilizar una red *wifi*. Era necesario realizar el sistema con dispositivos de bajo coste.

Para cubrir dichos objetivos se han realizado una serie de tareas relacionadas:

- Se ha desarrollado una aplicación para dispositivos móviles, utilizando el entorno de desarrollo *Unity*.
- Se realizó un estudio de los dispositivos disponibles en el mercado capaces de realizar comunicaciones wifi. Finalmente se utilizó el módulo NodeMCU, que disponía de un módulo wifi. El NodeMCU tiene un coste bajo y además dispone de los puertos necesarios para recibir los datos del sensor IMU.
- Se estudiaron diferentes sensores para recibir la posición de la pistola, pero finalmente se utilizó el IMU MPU6050, por su bajo coste.
- Se realizaron diferentes pruebas para conseguir recoger los movimientos del jugador dentro del entorno, finalmente se optó por la solución de una única Raspberry Pi, como se ha comentado anteriormente.
- Uno de los principales objetivos era conseguir una experiencia en tiempo real. Este objetivo se ha conseguido en gran medida, ya que existe un retardo de menos de 5 ms, desde que se produce el movimiento hasta que se traslada al videojuego.
- Se utilizado un software de edición 3D, para la creación de una pistola, como se ha podido observar en capítulos anteriores.
- Para el sistema de posicionamiento se ha realizado una aplicación que recoge los movimientos del jugador utilizando una webcam conectada a una Raspberry Pi.

Se han adquirido gran cantidad de conocimientos en diferentes áreas, como son:

- Programación de dispositivos electrónicos, con el sistema de desarrollo Arduino.
- Conocimientos en sensores de tipo IMU.
- Creaciones de aplicaciones con comunicaciones basadas en redes wifi.

- Creación de modelos 3D, con el software de edición *SketchUp*.
- Programación de aplicaciones gráficas, para dispositivos móviles.
- Desarrollo de aplicaciones de realidad virtual.
- Comunicaciones entre dispositivos en tiempo real.
- Programación de aplicaciones con reconocimiento de imágenes, utilizando la librería OpenCV.
- Conocimientos en programación en Python.
- Programación de script, con el entorno de programación Visual Studio.

Cabe destacar que el sistema desarrollado cumple perfectamente con los objetivos planteados. Se ha conseguido desarrollar un sistema que despierta gran interés en el alumnado, como se ha podido observar durante las visitas a los laboratorios.

Como inconveniente al sistema planteado, sería interesante mejorar el sistema de posicionamiento del jugador dentro del entorno. Se han probado varios sistemas comerciales de realidad virtual, que mejoran la experiencia en este sentido. Aunque, cabe destacar, que el sistema se comporta de una forma adecuada y consigue una experiencia inmersiva por parte del jugador.

Figura 6.1 Fotografía alumno SAFA-Linares

7. LINEAS FUTURAS

Dentro de posibles líneas de futuro, cabe destacar:

- **Mejora Sistema Posicionamiento:** Se ha observado que el sistema de posicionamiento ofrece una experiencia positiva, pero tiene bastantes deficiencias, con respecto a los actuales sistemas de posicionamiento. Sería interesante implementar un nuevo sistema de posicionamiento, basado en balizas.
- **Implementación para sistemas comerciales:** Actualmente existen sistemas de realidad virtual, que realizan un posicionamiento basado en varias cámaras. Sería interesante realizar una adaptación de la aplicación a dichos sistemas.
- **Juego en red:** En la actual implementación del videojuego, solo existe un jugador. En futuras implementación se podría adaptar para que pudiesen jugar simultáneamente varios jugadores.
- **Adaptación a otros dispositivos:** El sistema se ha implementado para dispositivos Android. Se podrían realizar compilaciones para OSX, LINUX, WINDOWS, etc.
- **Mejora sensores electrónicos:** Sería interesante realizar pruebas con otras IMUs, para comparar los resultados obtenidos.
- **Mejora envío de datos:** Todas las comunicaciones entre los dispositivos se han realizado a través de wifi, lo que conlleva un cierto retardo. Sería interesante, probar un envío de datos utilizando *bluetooth*.
- **Creación de nuevos dispositivos:** Se podrían crear nuevos dispositivos, como por ejemplo: sables laser, arcos, brochas, etc.
- **Creación de nuevos niveles:** Esta aplicación es un sistema cerrado, en el cual solo hay un entorno virtual. Se podrían crear diferentes niveles, incluyendo nuevos enemigos y dificultades.
- **Estudiar nuevas aplicaciones:** Se podría realizar un estudio de nuevas aplicaciones de realidad virtual.
- **Incluir nuevos sensores:** Para la realización de este TFG se ha utilizado un sensor posición. Se podrían realizar pruebas con diferentes sensores, como por ejemplo de ultrasonidos, de proximidad, gases, etc.

8. ESTUDIO ECONÓMICO

En este apartado se realizará un presupuesto total para la realización de este proyecto, indicando el tiempo empleado y los costes de material, para cada uno de los componentes del sistema.

8.1 Pistola

- Desglose de horas empleadas:

ACTIVIDAD	HORAS
Estudio previo y documentación	30
Realización de pruebas sensor MPU6050	30
Desarrollo aplicación	20
TOTAL	**80**

- Costes de materiales:

PRODUCTO	PRECIO
Impresión 3D pistola	20 €
Sensor MPU6050	2 €
NodeMCU	5 €
Componentes electrónicos (Resistencias, cables, led, interruptor, pulsador)	2 €
Software Arduino	0 €
TOTAL	**29 €**

- Recursos humanos:

OCUPACIÓN	HORAS	PRECIO / HORA	IMPORTE
Ingeniero	80	48 €	3840 €
TOTAL	**3840 €**		

- Costes totales:

Concepto	PRECIO
Costes de materiales	29 €
Recursos humanos	3840 €
Costes indirectos (20%)	773,8 €
Subtotal	4642,8 €
I.V.A. (21%)	974,98 €
TOTAL	**5617,78 €**

El total del coste de creación de la **pistola** es de **cinco mil seiscientos diecisiete euros con setenta y ocho céntimos de euro**.

8.2 Dispositivo de posicionamiento

- Desglose de horas empleadas:

ACTIVIDAD	HORAS
Estudio previo y documentación	30
Realización de pruebas posicionamiento	70
Desarrollo aplicación	10
TOTAL	**110**

- Costes de materiales:

PRODUCTO	PRECIO
Raspberry Pi 3 (Kit completo)	55 €
Cámara Raspberry	10 €
Software (Raspbian, OpenCV y Python)	0 €
TOTAL	**65 €**

- Recursos humanos:

OCUPACIÓN	HORAS	PRECIO / HORA	IMPORTE
Ingeniero	110	48 €	5280 €
TOTAL	**5280 €**		

- Costes totales:

Concepto	PRECIO
Costes de materiales	65 €
Recursos humanos	5280 €
Costes indirectos (20%)	1069 €
Subtotal	6414 €
I.V.A. (21%)	1346,94 €
TOTAL	**7760,94 €**

El total del coste de creación del **dispositivo de posicionamiento** es de **siete mil setecientos sesenta euros con noventa y cuatro céntimos de euro**.

8.3 Aplicación de realidad virtual

- Desglose de horas empleadas:

ACTIVIDAD	HORAS
Estudio previo y documentación	50
Desarrollo aplicación realidad virtual	70
TOTAL	**120**

- Costes de materiales:

PRODUCTO	PRECIO
Ordenador Completo	600 €
Dispositivo Android	100 €
Gafas de realidad virtual	20 €
Software (Unity)	0 €
TOTAL	**750 €**

- Recursos humanos:

OCUPACIÓN	HORAS	PRECIO/ HORA	IMPORTE
Ingeniero	120	48 €	5760 €
TOTAL	**5760 €**		

- Costes totales:

Concepto	PRECIO
Costes de materiales	750 €
Recursos humanos	5760 €
Costes indirectos (20%)	1302 €
Subtotal	7812 €
I.V.A. (21%)	1640,52 €
TOTAL	**9452,52 €**

El total del coste de creación de la **aplicación de realidad virtual** es de **nueve mil cuatrocientos cincuenta y dos euros con cincuenta y dos céntimos de euro**.

8.3 Coste total del proyecto

Concepto	PRECIO
Pistola	5617,78 €
Dispositivo posicionamiento	7760,94 €
Aplicación de realidad virtual	9452,52 €
TOTAL	**22831,24 €**

El total del coste de creación del **sistema de realidad virtual** es de **veintidós mil ochocientos treinta y uno euros con veinticuatro céntimos de euro**.

9. BIBLIOGRAFÍA

[1] **Vicente García.** Electrónica Práctica Aplicada, 2-3-2018. [en línea]

Dirección del recurso:

https://www.diarioelectronicohoy.com/blog/configurar-el-mpu6050

[2] **mplanaslasa.** QuantDare, 28-3-2014. [en línea]

Dirección del recurso:

https://quantdare.com/filtro-kalman/

[3] **LUIS LLAMAS.** LUIS LLAMAS, 7-9-2016. [en línea]

Dirección del recurso:

https://www.luisllamas.es/medir-la-inclinacion-imu-arduino-filtro-complementario/

[4] **Alex Vargas Benamburg.** GitHub, 11-2-2017. [en línea]

Dirección del recurso:

https://github.com/alexvargasbenamburg/arduino_FiltroKalman

[5] **Tr4nsduc7or.** ROBOLOGS, 16-8-2015. [en línea]

Dirección del recurso:

https://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

[6] **JUAN CARLOS.** BOOLEANBITE, 10-2-2015. [en línea]

Dirección del recurso:

https://booleanbite.com/web/adquisicion-de-datos-con-arduino-i-tiempo-de-muestreo-y-resolucion/

[7] **ismomo1.** UNITY3DTUTORIAL, 16-8-2015. [en línea]

Dirección del recurso:

https://unity3dtutorial.wordpress.com/2015/08/16/raycast/

[8] **Iván Martín.** Topes de gama, 16-9-2018. [en línea]

Dirección del recurso:

https://topesdegama.com/noticias/gadgets/cambiar-red-wifi-chromecast-sin-reiniciarlo

10. ANEXOS

10.1 ANEXO INSTALACIÓN PLACAS ESP8266 EN ARDUINO IDE

Para poder programar dispositivos basados en el módulo ESP8266, es necesario seguir los siguientes pasos:

- Descarga la última versión disponible del Arduino IDE. (https://www.arduino.cc/)
- Iniciar Arduino y abrir la pantalla de preferencias.
- Dentro del campo “Gestor de URLs adicionales de Tarjetas”, se debe introducir la url: https://arduino.esp8266.com/stable/package_esp8266com_index.json y aceptamos.
- En la pestaña Herramientas, se debe ir a la sección Placa - > Gestión de Tarjetas
- En el buscador introducimos “esp8266” e instalamos el módulo.
- Ya se puede trabajar con los módulos basados en esp8266.

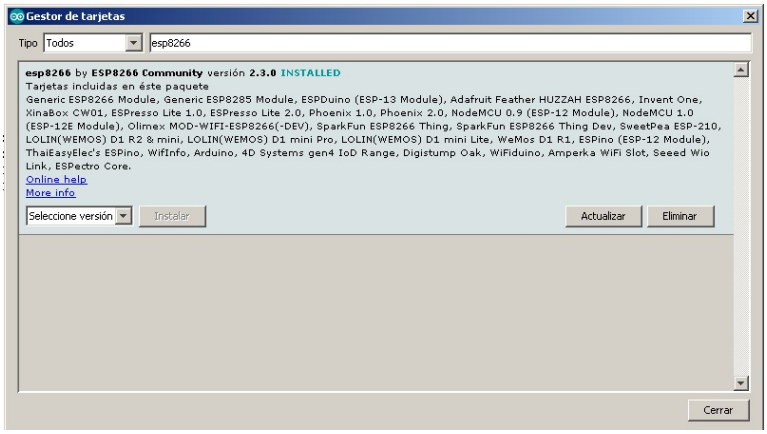

Figura 9.1 Gestor Tarjetas Arduino IDE.

10.2 ANEXO MANUAL INSTALACIÓN

10.2.1 Instalación aplicación en dispositivo móvil.

Para realizar la instalación de la aplicación en el dispositivo móvil, es necesario descargar el archivo “zombievr.apk”, desde la url :

http://turmandreams.es/archivos/zombievr.apk

Una vez descargado, se debe habilitar la opción de instalación desde orígenes desconocidos. Si no se encuentra habilita esta opción, se puede habilitar desde el menú Ajustes -> Seguridad .

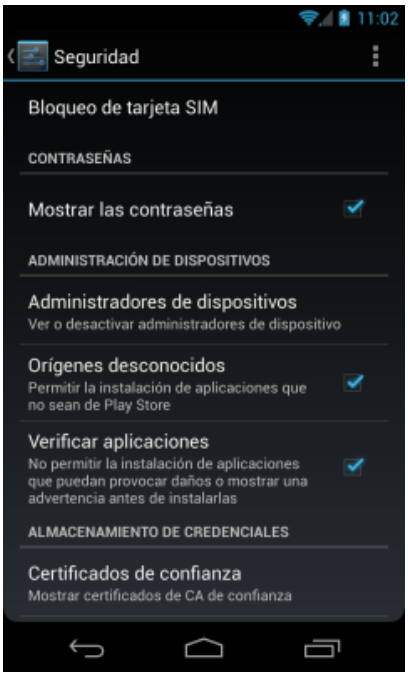

Figura 10.2.1.1 Habilitar orígenes desconocidos.

Una vez habilita la opción de Orígenes desconocidos podemos proceder a la instalación de la aplicación en el dispositivo móvil, desde el navegador de archivos de Android.

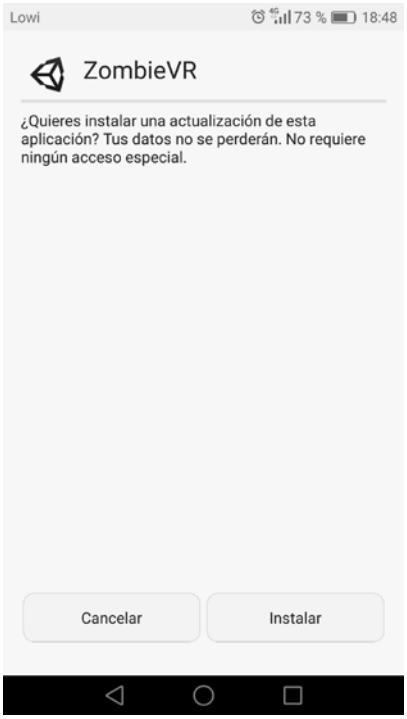

Figura 10.2.1.2 Instalación aplicación en dispositivo móvil.

Se pulsa la tecla “Instalar”, y una vez finalice el proceso, la aplicación puede ser ejecutada.

10.2.2 Instalación Raspberry Pi.

Para el dispositivo de posicionamiento, es necesario disponer de una Raspberry Pi y de una cámara.

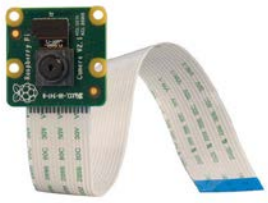

Figura 10.2.2.1 Cámara para Raspberry Pi.

La cámara debe conectarse correctamente en el puerto específico, como se representa en la siguiente figura:

Figura 10.2.2.2 Conexión Cámara en Raspberry Pi.

Una vez ha sido conectada la cámara podemos cargar en una tarjeta microSD, la imagen de nuestro sistema operativo. Previamente se debe descargar el archivo “tfg-rasp-iso.rar”, desde la url:

http://turmandreams.es/archivos/tfg-rasp-iso.rar

Una vez descargado el archivo, se descomprime y se obtiene un archivo llamado “tfg-rasp-iso”, que ocupa sobre 8 GB. Para realizar la instalación del sistema operativo en una tarjeta *microSD*, esta debe ser de al menos 8 GB de capacidad.

Figura 10.2.2.3 MicroSD.

Para cargar el sistema operativo en la tarjeta *microSD*, se utiliza el software "win32diskimager", que podemos descargarlo de la url :

https://sourceforge.net/projects/win32diskimager/

Una vez instalado, ejecutamos el "win32diskimager", pulsando sobre el icono "carpeta", como se representa en la figura:

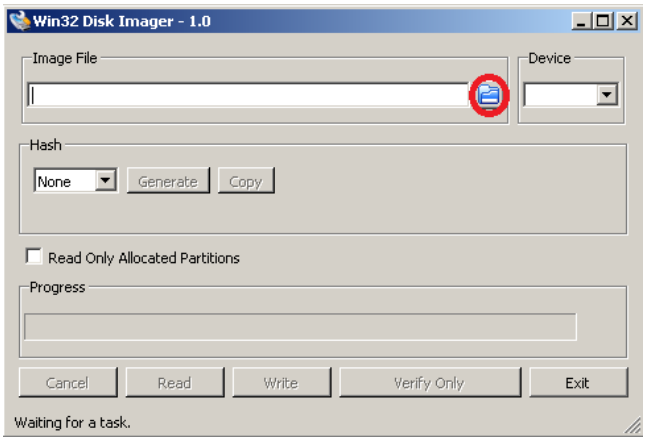

Figura 10.2.2.4 Win32 Disk Imager.

Se selecciona el archivo "tfg-rasp-iso" y se pulsa el icono "Write". Una vez finaliza el proceso la imagen queda grabada en la *microSD*. Esta imagen está configurada con las librerías necesarias para realizar compilaciones de aplicaciones Python con OpenCV.

10.2.3 Instalación firmware Pistola.

Para poder instalar el firmware de la pistola en un NodeMCU, se tener instalado previamente el *Arduino IDE*. Podemos descárgalo de la url :

https://www.arduino.cc/

También se debe descargar el archivo "pistola.ino", que contiene el código fuente del firmware de la pistola. Se puede descargar de la url:

http://turmandreams.es/archivos/pistola.ino

Se ejecuta Arduino IDE y se abre el archivo “pistola.ino”. Es necesario realizar una instalación para poder compilar aplicaciones para dispositivos basados en *ESP8266*, como se ha comentado en el Anexo 10.1. Se conecta el NodeMCU, por el puerto microusb, al ordenador. Una vez conectamos el NodeMCU, el ordenador detecta un puerto serie virtual. Desde la pestaña Herramientas seleccionamos el puerto serie virtual, como muestra la siguiente Figura:

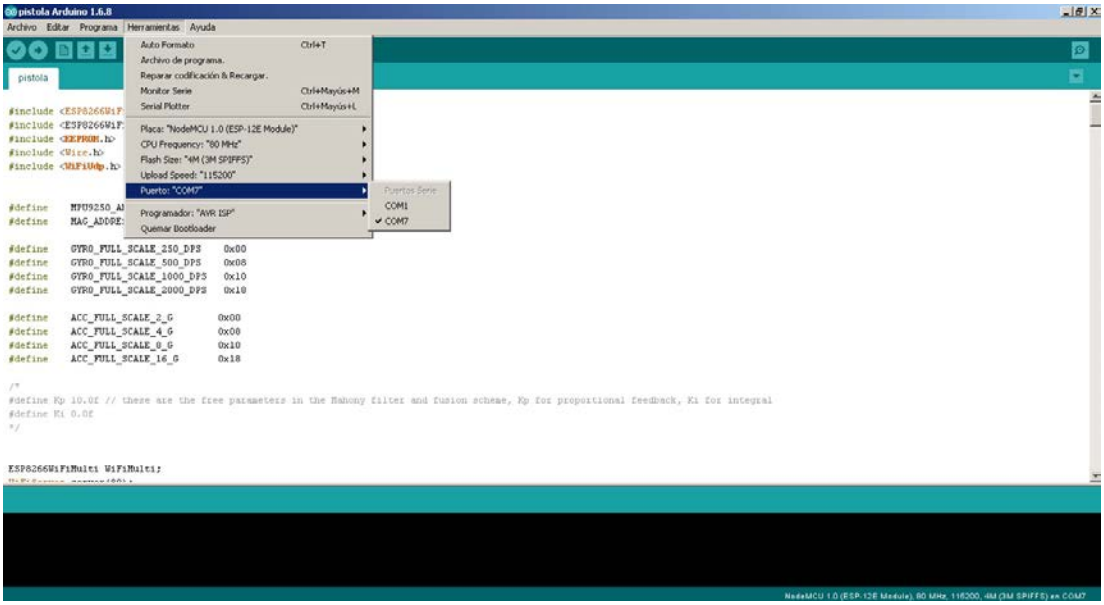

Figura 10.2.3.1 Selección puerto en Arduino IDE.

Además las opciones dentro de la pestaña Herramientas tienen que tener los siguientes valores:

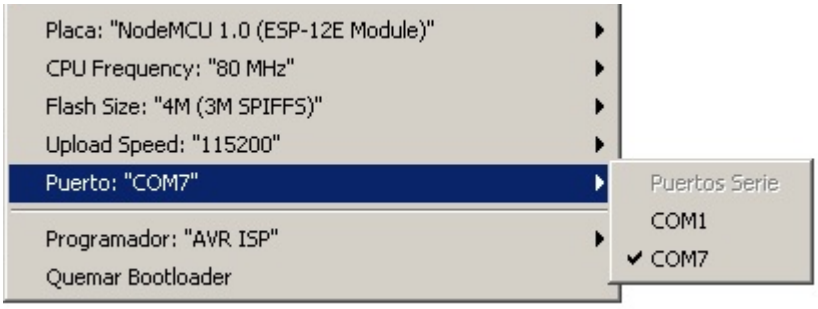

Figura 10.2.3.2 Valores de configuración Arduino.

Una vez está configurado Arduino IDE, podemos cargar el firmware de la pistola, en el NodeMCU, pulsando sobre el icono que se resalta en la siguiente figura:

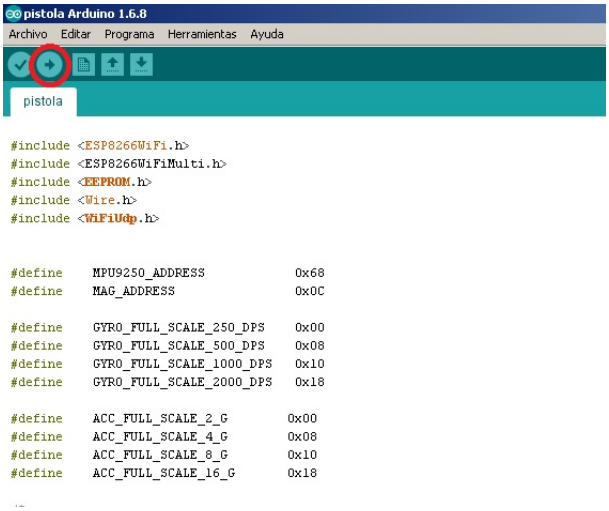

Figura 10.2.3.3 Cargar Firmware en NodeMCU.

Una vez finaliza el proceso el firmware ya está cargado en el NodeMCU y su modo de funcionamiento, es el indicado anteriormente.

10.3 ANEXO INSTALACIÓN OPENCV EN RASPBIAN

En este anexo se detallan los comandos que han sido necesarios ejecutar en el terminal del sistema Raspbian, instalado en la raspberry pi. Con estos comandos se instalan las librerías necesarias para poder compilar programas que incluyan OpenCV.

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install build-essential cmake pkg-config

sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev

sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

sudo apt-get install libxvidcore-dev libx264-dev

sudo apt-get install libgtk2.0-dev libgtk-3-dev

sudo apt-get install libatlas-base-dev gfortran

sudo apt-get install python3-dev

sudo apt-get install python3-pip

pip3 install opencv-python

sudo apt-get install libqtgui4

sudo modprobe bcm2835-v4l2

sudo apt-get install libqt4-test

10.4 ANEXO MANUAL DE USUARIO

En este apartado se explicarán los diferentes pasos que deben realizarse para poder realizar una demostración del funcionamiento del sistema de realidad virtual.

En primer lugar debe configurarse el dispositivo móvil, como punto de acceso wifi, como se ha explicado en el capítulo 4.2.4. Una vez configurado el móvil, como punto de acceso, se carga la aplicación previamente instalada ZombieVR.

Se puede observar que la pantalla del móvil, aparece divida en dos. De esta forma cuando se coloque el móvil en las *CardBoard*, el usuario podrá visualizar el juego en 3D.

Figura 10.4.1 Cardboard.

Encendemos la pistola, con el interruptor de encendido y automáticamente se conecta a la red creada por defecto "@epslinares_uja". Si se desea generar una red con distinto *SSID* y contraseña, es necesario configurar correctamente la pistola, como se comentaba en el capítulo 4.2.2.

Una vez el usuario observe que los movimientos que se realizan con la pistola, se reflejan en el entorno virtual, se procede a conectar la Raspberry Pi. Una vez se alimenta la Raspberry Pi por el puerto microUSB, está comienza a cargar el sistema operativo, que puede tardar sobre unos 20 segundos. Automáticamente ejecuta el programa en Python que reconoce los movimientos del jugador y envía la información al dispositivo móvil. Para confirmar que se está detectando al jugador, este debe moverse dentro de la estancia. Si los movimientos se reflejan en desplazamientos por el entorno virtual, esto indica que está funcionando correctamente.

Para poder visualizar la partida en una pantalla externa, se ha utilizado el dispositivo *Chromecast*. Se debe tener configurado el *Chromecast* para que se conecte a la red SSID creada por el dispositivo móvil. En la referencia [8], puede observarse el proceso de configuración. Para realizar el envío es necesario tener instalado la aplicación *"Google Home"*, que puede instalarse desde el *Play Store* de *Android*. Para comenzar a visualizar la pantalla del móvil en el chromecast se debe pulsa en la opción "Proyectar dispositivo", como puede observarse en la siguiente figura:

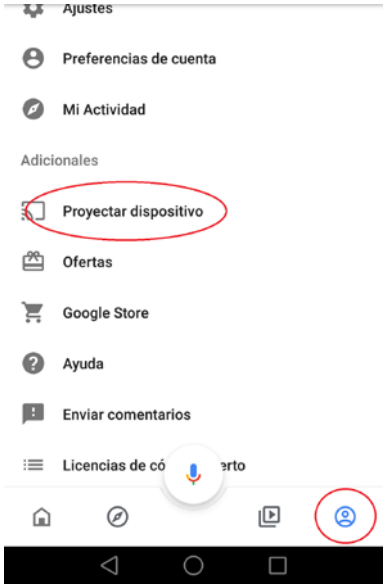

Figura 10.4.2 Proyectar dispositivo.

Una vez pulsada la opción "Proyectar dispositivo", se debe pulsar sobre el botón "Enviar Pantalla / Audio", como se muestra en la siguiente figura:

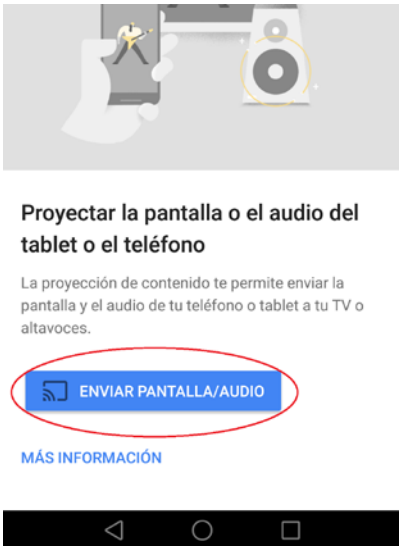

Figura 10.4.3 Enviar Pantalla / Audio.

Una vez se pulse sobre el botón “Enviar Pantalla / Audio”, se puede visualizar la pantalla del dispositivo móvil en el *Chromecast*. Para las demostraciones es interesante utilizar un proyector, para que todos los visitantes puedan ver lo que ocurre durante el juego.

El sistema está configurado correctamente y la partida puede comenzar. Para que comience la partida el jugador, solo tiene que pulsar el gatillo y empezarán a aparecer zombis. El jugador puede disparar a los zombies para conseguir no se eliminado y ganar puntos. El jugador dispone de 10 disparos seguidos, para poder seguir disparando debe recargar. Esto se realiza, disparando hacia el cielo del entorno virtual, durante la recarga se escucha un sonido característico de recarga de pistola.

El juego termina cuando el jugador pierde el total de la vida. Una vez ocurre esto el juego se reinicia.

10.5 ANEXO CODIGO PYTHON RASPBERRY PI

```
# USAGE
# python motion_detector.py
# python motion_detector.py --video videos/example_01.mp4

# import the necessary packages
import argparse
import datetime
import imutils
import time
import cv2
import socket
import numpy as np

ip="192.168.43.1"
puerto=9000


# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
ap.add_argument("-a", "--min-area", type=int, default=500, help="minimum area size")
args = vars(ap.parse_args())

# if the video argument is None, then we are reading from webcam
if args.get("video", None) is None:
        camera = cv2.VideoCapture(0)
        time.sleep(0.25)

# otherwise, we are reading from a video file
else:
        camera = cv2.VideoCapture(args["video"])

# initialize the first frame in the video stream
firstFrame = None

posx=200
posy=0




# loop over the frames of the video
while True:

        if not firstFrame is None:
               firstFrame=gray


        # grab the current frame and initialize the occupied/unoccupied
        # text
        (grabbed, frame) = camera.read()
        text = "Unoccupied"

        # if the frame could not be grabbed, then we have reached the end
```

```
# of the video
if not grabbed:
        break

# resize the frame, convert it to grayscale, and blur it
frame = imutils.resize(frame, width=400)

img=frame[150:300,0:400]
frame=img

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray,(21,21),0)

# if the first frame is None, initialize it
if firstFrame is None:
        firstFrame = gray
        continue

# compute the absolute difference between the current frame and
# first frame
frameDelta = cv2.absdiff(firstFrame, gray)
thresh = cv2.threshold(frameDelta,10,255,cv2.THRESH_BINARY)[1]

# dilate the thresholded image to fill in holes, then find contours
# on thresholded image
thresh = cv2.dilate(thresh, None, iterations=2)
(cnts, _) = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
        cv2.CHAIN_APPROX_SIMPLE)


# loop over the contours
for c in cnts:
        # if the contour is too small, ignore it
        #if cv2.contourArea(c) < args["min_area"]:
        #        continue

        # compute the bounding box for the contour, draw it on the frame,
        # and update the text
        (x, y, w, h) = cv2.boundingRect(c)
        p1x=x+(w/2)
        p1y=y+(h/2)

        #print str(p1x)+" , "+str(p1y)

        if p1x > posx:
                dif=p1x-posx
        else:
                dif=posx-p1x

        dif=dif*0.1

       # if dif>0.5:
       #         dif=0.5

       # posx=int((posx*(1-dif))+(p1x*dif))

        if p1x > posx:
```

```
                posx=posx+int(dif);
            else:
                 posx=posx-int(dif);


            if p1y > posy:
                 dif=p1y-posy
            else:
                 dif=posy-p1y

            dif=dif*0.05
            if dif>0.5:
                 dif=0.5

            posy=int((posy*(1-dif))+(p1y*dif))

            #print str(posx)+" , "+str(posy)

            #enviamos por udp los datos

            sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)

            sock.sendto("3;"+str(posx)+";",(ip,puerto))

            #sock.sendto("1;0;0;0;0;",(ip,puerto))


            break

        # show the frame and record if the user presses a key


       # cv2.rectangle(frame, (posx,posy),(posx+20,posy+20), (0, 255, 0), 2)
       # cv2.rectangle(frame, (posx,60),(posx+20,80), (0, 255, 0), 2)


       # cv2.imshow("Camara", frame)
       # cv2.imshow("Dilatado",thresh)
       # cv2.imshow("Recortada",frame2)


        key = cv2.waitKey(1) & 0xFF

        # if the `q` key is pressed, break from the lop
        if key == ord("q"):
            break

# cleanup the camera and close any open windows
camera.release()
cv2.destroyAllWindows()
```

10.6 ANEXO MANUAL MANTENIMIENTO

El sistema de realidad virtual planteado, requiere de poco mantenimiento. Únicamente cabe destacar que es importante tener cargada la pistola al máximo, ya que se ha observado que cuando la batería de la pistola, no alcanza un 30% de carga, esta no realiza correctamente el envío de datos a la aplicación móvil.

También cabe destacar que es importante realizar una correcta calibración de las lentes de la Cardboard. Ya que cada jugador requiere una colocación específica de las lentes.