Trabajo de Fin de Grado

Grado en Ingeniería Informática

Sistema de visión en 360º para personas con diversidad funcional usando Realidad Virtual

360º vision system for people with functional diversity using Virtual Reality

Kevin Miguel Rivero Martín

La Laguna, 2 de Julio de 2017

D. **Rafael Arnay del Arco**, con N.I.F. 78.569.591-G profesor Ayudante Doctor adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **José Demetrio Piñero Vera**, con N.I.F. 43.774.048-B profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"Sistema de visión en 360º para personas con diversidad funcional usando Realidad Virtual."

ha sido realizada bajo su dirección por D. **Kevin Miguel Rivero Martín**, con N.I.F. 42.199.315-L.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos, firman la presente en La Laguna a 2 de Julio de 2017.

Agradecimientos

A mis padres, a mi hermana, a mi pareja y a toda mi familia por sus esfuerzos y apoyo durante mis estudios, sin ellos todo esto no hubiera sido posible.

A Rafael Arnay del Arco por su magnífica labor como tutor durante el desarrollo del trabajo. Dispuesto a ayudar en todo momento y a resolver cualquier duda que me surgiera. A José Demetrio Piñero Vera por su labor como cotutor y por facilitar las gafas de realidad virtual para utilizarlas en este proyecto.

También, a:

- Jonay Tomás Toledo Carrillo por facilitar la placa de Arduino y su enseñanza de cómo utilizarla.
- Javier Hernández Aceituno por transmitir sus conocimientos en la utilización de la diadema Emotiv.
- Por último, pero no menos importante, a Manuel Fernández Vera por su ayuda en la construcción del par estéreo.

Licencia

Resumen

Este trabajo está enfocado en la creación y desarrollo de un sistema de visión en 360º basado en realidad virtual con el fin de facilitar una herramienta a personas con diversidad funcional para ser utilizada en una silla de ruedas autónoma.

La idea es que una persona que presente una discapacidad grave y que apenas pueda mover la cabeza, sea capaz de tener una visión amplia del entorno y ser más consciente de lo que le rodea. Los pequeños movimientos de cabeza que pueda hacer se traducirán en giros de un sistema de visión 3D conectado a unas gafas de realidad virtual que llevará puestas, permitiendo así, que vea lo que ocurre a su alrededor sin tener que girar la silla.

Palabras clave: Realidad virtual, Emotiv, Arduino, servomotor, par estéreo, Unity3D, OpenCV, C, C++, C#.

Abstract

This work is focused on the creation and development of a 360º vision system based on virtual reality to provide a tool for people with functional diversity to be used in an autonomous wheelchair.

The idea is that people who has a severe disability and can barely move their head, can have a broad vision of the environment and be more aware of what surrounds them. The little head movements that they can do will be translated into turns of a 3D vision system connected to virtual reality glasses that they will wear, allowing them to see what happens around them without having to steer the wheelchair.

Keywords: Virtual reality, Emotiv, Arduino, servomotor, stereo pair, Unity3D, OpenCV, C, C++, C#.

Índice General

Índice de Figuras

Índice de tablas

Capítulo 1. Introducción

1.1 Descripción, antecedentes y estado del arte

Actualmente, las personas que utilizan una silla de ruedas debido a una diversidad funcional grave, como puede ser la tetraplejia, ven reducida notablemente su percepción del entorno debido a que no pueden girar la silla de ruedas por sí mismos, teniendo que depender de otra persona para que les ayude. Lo que se pretende con este trabajo es proporcionar más libertad e independencia a estas personas mediante un sistema de visión en 360º basado en realidad virtual que les permita ver todo lo que les rodea y ser más conscientes de ello.

La idea es que una persona que sea capaz de realizar pequeños movimientos de cabeza pueda controlar la herramienta completamente. Para ello, los movimientos de cabeza que realice se traducirán en giros de un sistema de visión 3D que, a su vez, transmitirá las imágenes a unas gafas de realidad virtual que llevará puestas, dándole la posibilidad de ver su entorno como si lo estuviera viendo con su propia vista además de evitarle tener que girar la silla.

Existen múltiples proyectos de sillas de ruedas autónomas en la actualidad, como el proyecto de silla de ruedas inteligente del MIT [1], la silla de Ford que se sube o se baja del vehículo de forma autónoma [2] y el proyecto de una silla de ruedas autónoma de unos investigadores mexicanos en Alemania [3] a la cual se le indica el punto de destino y ella automáticamente realiza el viaje. En la mayoría de los casos, el tipo y la disposición de los sensores presentes en las sillas es similar, y los objetivos suelen consistir en el transporte de personas de movilidad reducida desde un punto a otro a través de comandos vocales. Sin embargo, no se ha encontrado ningún proyecto como el propuesto para su aplicación a sillas de ruedas inteligentes.

Lo que se plantea en este proyecto es aprovechar la irrupción de sistemas de realidad virtual para dotar a una silla de ruedas inteligente (en desarrollo actualmente por el Grupo de Robótica de la Universidad de La Laguna) de un sistema de visión en 360º, de forma que ayude al usuario a ser más consciente de su entorno.

En la realización de este proyecto es necesario el uso de gafas de realidad virtual, un par de cámaras web para construir un par estéreo, un giroscopio, un motor o motores para el movimiento del par estéreo, una placa Arduino para el control de éstos y un ordenador para procesar todos los datos y hacer

que los componentes funcionen conjuntamente. Actualmente, se cuenta con este material.

1.2 Objetivos a cumplir en el trabajo

Dentro de este trabajo hay varios objetivos a cumplir para considerarlo como finalizado. Están expuestos en un orden incremental, ya que, se considera que es la mejor forma de abordar el trabajo, comenzando por la base y añadiendo más funcionalidades.

- Construir el par estéreo con las dos cámaras web.
- Comunicar el par estéreo con las gafas de realidad virtual.
- Calibrar las imágenes obtenidas con el par estéreo para que se vean correctamente en las gafas de realidad virtual.
- Obtener los datos de los giroscopios de las gafas de realidad virtual utilizando C# y el SDK de las gafas.
- Mover dos servomotores con Arduino, C/C++ y los datos obtenidos de los giroscopios.

Capítulo 2. Elementos del proyecto

2.1 Gafas de realidad virtual

Figura 1: Gafas de realidad virtual

Son unas gafas de realidad virtual modelo XG PCVR diseñadas en el proyecto I AM Cardboard. Tiene una entrada micro USB y USB para la alimentación además de una entrada de vídeo HDMI y giroscopio. Su pantalla puede trabajar a una resolución de 1920x1080 px para garantizar una buena experiencia en la realidad virtual empleada en este trabajo. También, su giroscopio ofrece una precisión aceptable, es por ello que empleó este en lugar del que ofrece la diadema Emotiv que veremos posteriormente.

2.2 Par estéreo

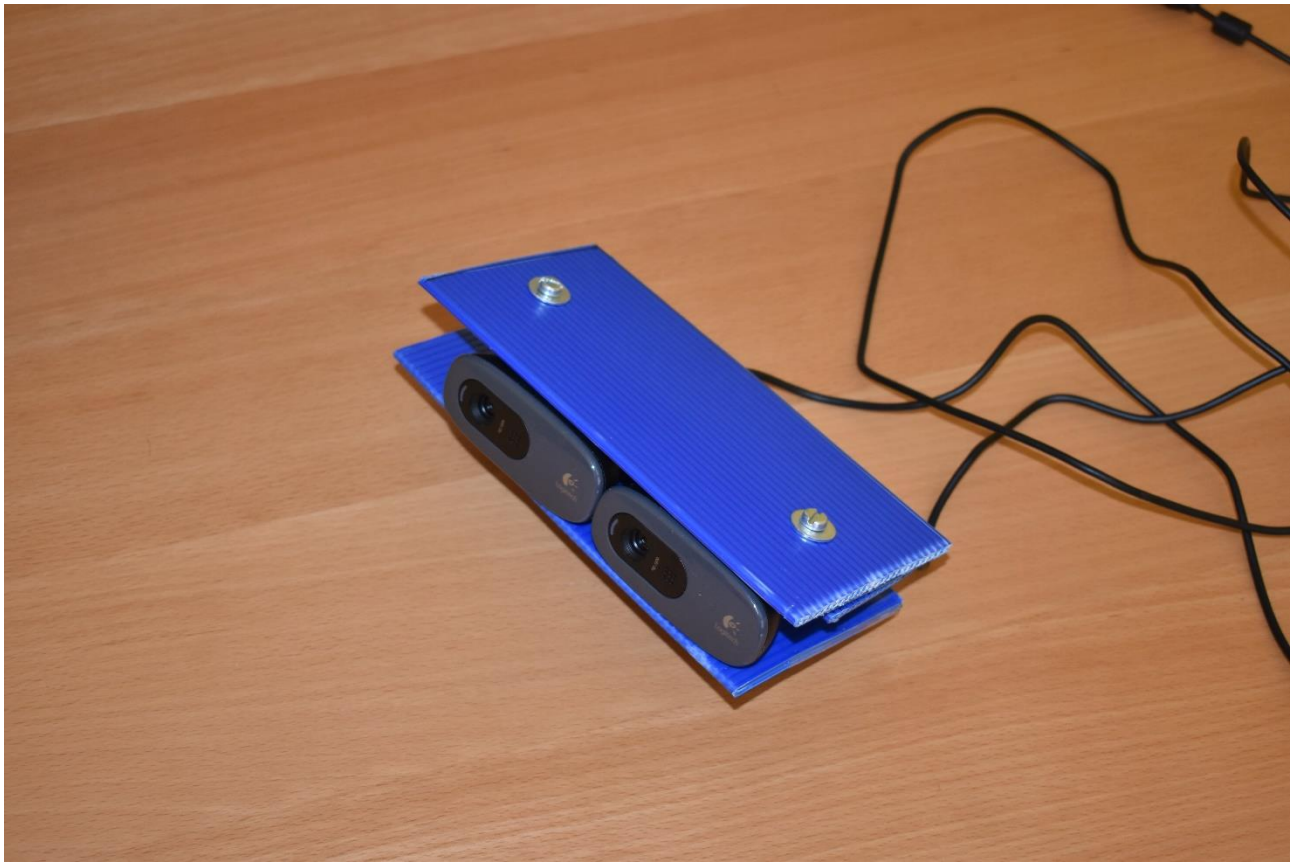

Figura 2: Par estéreo

Está compuesto por dos cámaras web Logitech C270 que obtienen una imagen de 1280x720px y se conectan a un ordenador por USB. Estas cámaras web están montadas en una plataforma, a una distancia de 7 cm entre objetivos, que se ha construido con el fin de mantenerlas fijas y tener un par estéreo fiable.

2.3 Diadema Emotiv

Figura 3: Diadema Emotiv

Esta diadema se coloca en la cabeza y se comunica con un ordenador mediante bluetooth. Lleva una batería incorporada que permite usarla por largos periodos de tiempo sin necesidad de cargarla. La diadema permite acceder, utilizando su SDK [4], a los giroscopios que lleva incorporados y, también, tiene una funcionalidad bastante interesante, la lectura de ondas cerebrales.

El acceso a los giroscopios nos devuelve los valores de la aceleración de los giros que, en función de la velocidad en que se realicen, serán mayores o menores y en la dirección en la que se gire serán positivos o negativos.

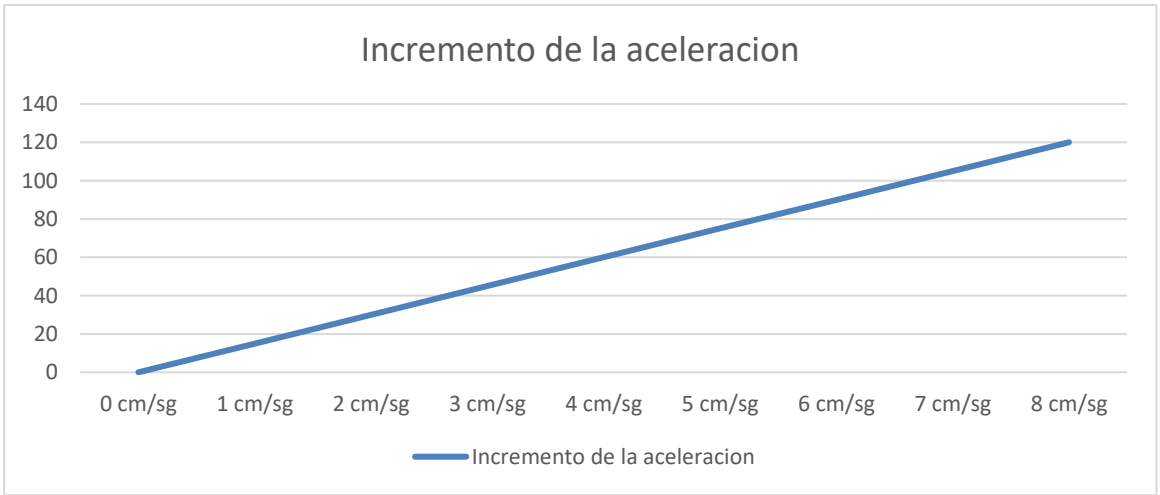

Figura 4: Incremento de los valores de aceleración Emotiv

Finalmente, la diadema no se ha utilizado, ya que los valores que devuelven sus giroscopios son muy variables, difíciles de controlar y no hay posibilidad de obtener los ángulos de giro que es lo que realmente se necesita en este proyecto. Como alternativa, se han utilizado los giroscopios incorporados en las gafas de realidad virtual.

2.4 Servomotores y placa Arduino

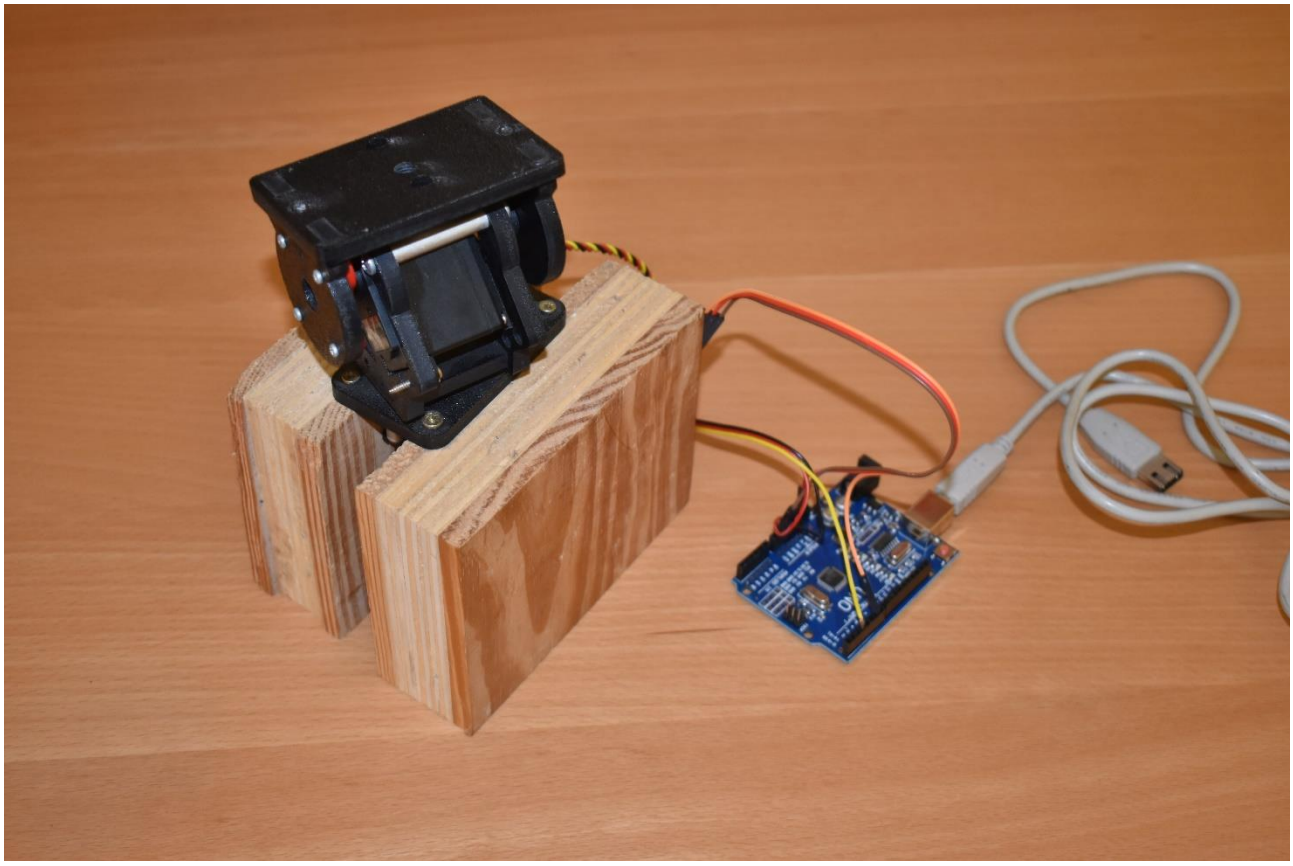

Figura 5: Servomotores con placa Arduino

Estos servomotores, dos concretamente, están conectados a una placa Arduino y, uno gira en horizontal mientras que el otro lo hace en vertical. La placa de Arduino, modelo Arduino Uno, cuenta con 14 entradas/salidas digitales, un conector USB mediante el cual se realiza la conexión al ordenador, un conector de alimentación para utilizarlo en caso de que la potencia suministrada por el USB no sea suficiente y un botón de reseteo.

El servomotor que gira en horizontal permite un giro que va desde 2º a 180º.

Figura 6: Representación del giro del servomotor en horizontal

Asimismo, el servomotor que gira en vertical permite un giro que va desde 0º a 138º.

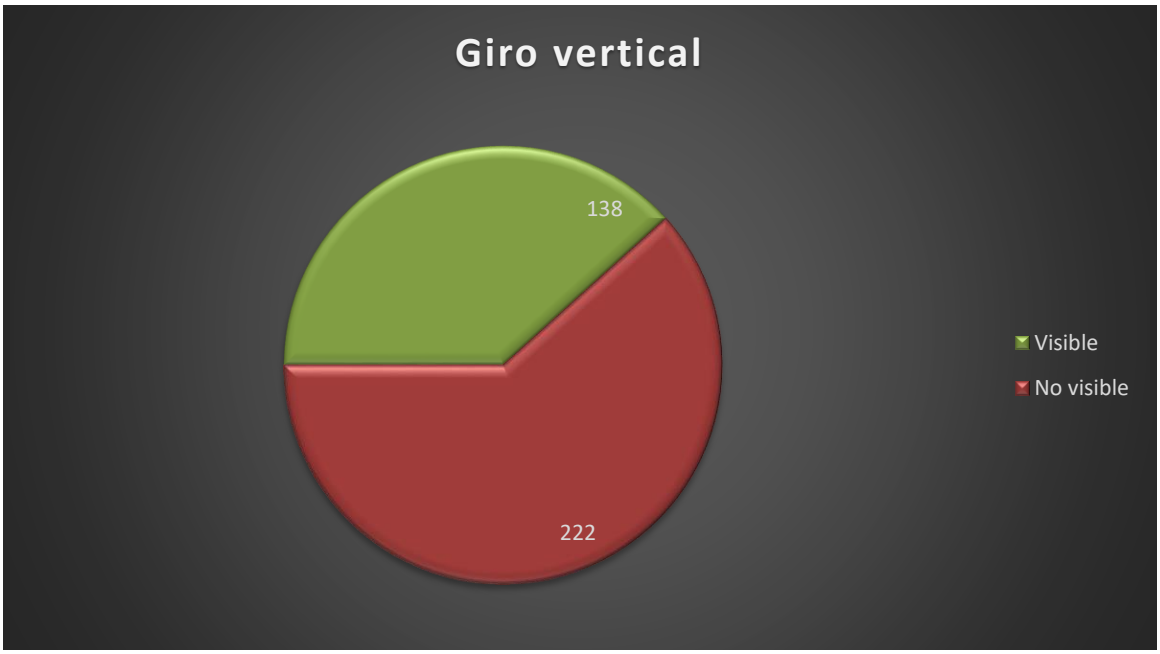

Figura 7: Representación del giro del servomotor en vertical

Capítulo 3. Fases del desarrollo del proyecto

3.1 Fase 1: Comunicación del par estéreo con las gafas de realidad virtual

En esta fase se pretende obtener la señal de vídeo de las cámaras y mostrarlas en las gafas de realidad virtual. Para ello, se ha creado una aplicación con el motor Unity3D [5].

Este motor está enfocado en el desarrollo de videojuegos multiplataforma, siendo ese su punto fuerte, sin embargo, también es posible utilizarlo en otro tipo de aplicaciones ya que las herramientas que proporciona lo hacen una opción bastante viable.

Para este trabajo se ha hecho uso de las diferentes herramientas que proporciona para la lectura de vídeo desde una entrada USB y mostrarla en la aplicación creada. También, uno de los motivos para la utilización de este motor es que, en un futuro, se podría añadir al proyecto nuevas funcionalidades empleando la realidad virtual y no solo mostrar las imágenes obtenidas.

Como punto de partida se comenzó montando las cámaras en su estructura para crear el par estéreo y, posteriormente, se desarrolló la aplicación de Unity.

Esta aplicación consta de una escena con varios objetos:

- Dos objetos “Plane” con el fin de proyectar en ellos las imágenes de vídeo obtenidas.
- Un objeto “Camera” para mostrar lo que ocurre en la escena al usuario, en este caso, los dos planos con la señal de vídeo.
- Un objeto “Light” para iluminar la escena.
- Un objeto “Canvas” con un objeto “Text” como hijo para mostrar información al usuario.

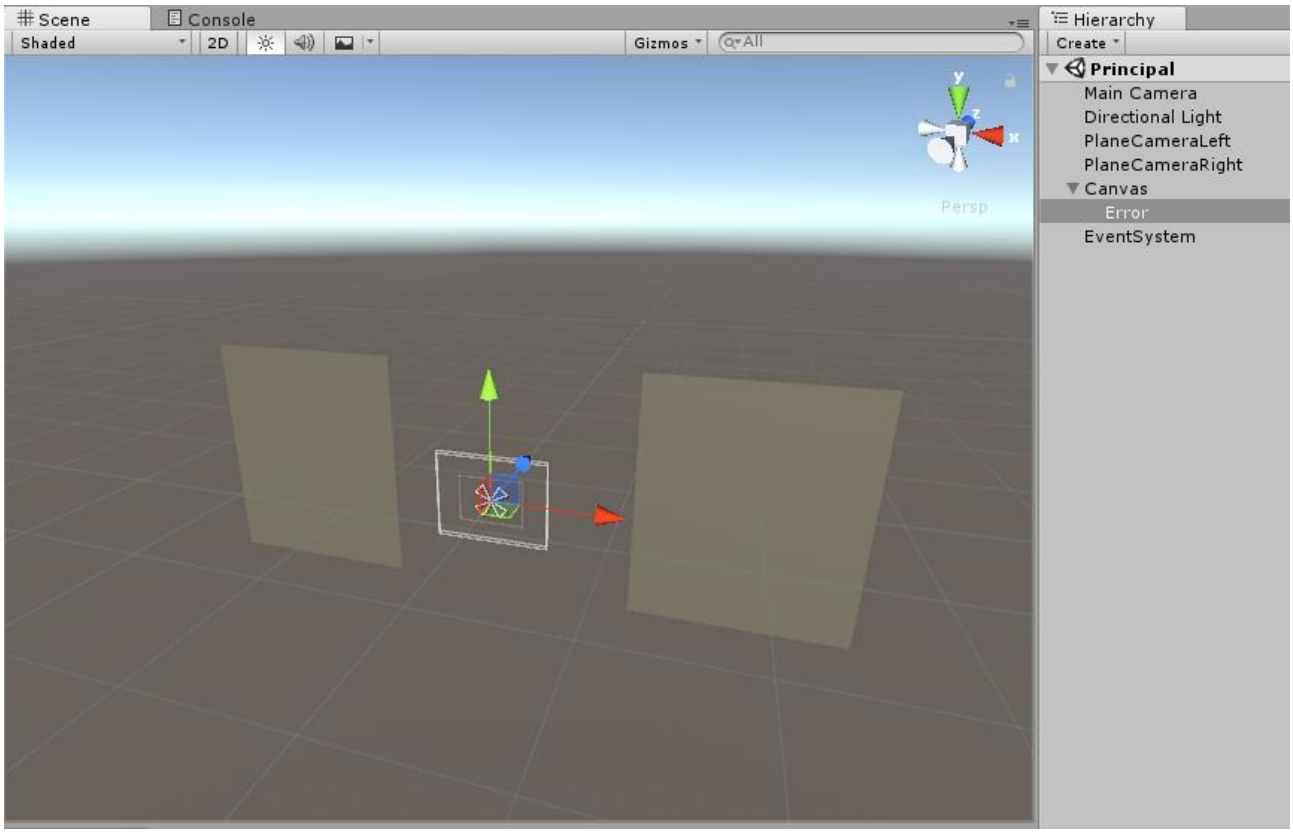

Figura 8: Escena de Unity3D

Luego se pasó a desarrollar la clase que realizaría todo el trabajo, es decir, obtener las imágenes y aplicarlas a los paneles.

```
public class VideoInput : MonoBehaviour {
    // Left Eye
    public Material inputMaterial_leftEye;
    private WebCamTexture camText_leftEye;
    // Right Eye
    public Material inputMaterial_rightEye;
    private WebCamTexture camText_rightEye;

    void Start() {
        WebCamDevice[] devices = WebCamTexture.devices;
        Debug.Log ("Detecto " + devices.Length + " dispositivos conectados");

        if (devices.Length < 2) {
            GameObject.Find ("Error").GetComponent<Text> ().enabled = true;
        } else {
            // Left Eye
            camText_leftEye = new WebCamTexture ();
            camText_leftEye.deviceName = devices[0].name;
            camText_leftEye.Play ();
            inputMaterial_leftEye.mainTexture = camText_leftEye;
            // Right Eye
            camText_rightEye = new WebCamTexture ();
            camText_rightEye.deviceName = devices[1].name;
            camText_rightEye.Play ();
            inputMaterial_rightEye.mainTexture = camText_rightEye;
        }
    }

}
```

Esta clase consta de dos atributos “Material”, aplicados uno a cada panel de la escena, y dos atributos “WebCamTexture”, uno para cada cámara, que permiten obtener la señal de vídeo. Una vez se pone en ejecución la aplicación, se obtienen los dispositivos WebCams disponibles y se asignan a los atributos “WebCamTexture” para, una vez se pone en marcha las señales de vídeo (camText_X.Play()), asignarlos a los “Material” aplicados a cada panel. Como resultado final obtenemos el vídeo de las cámaras en cada panel.

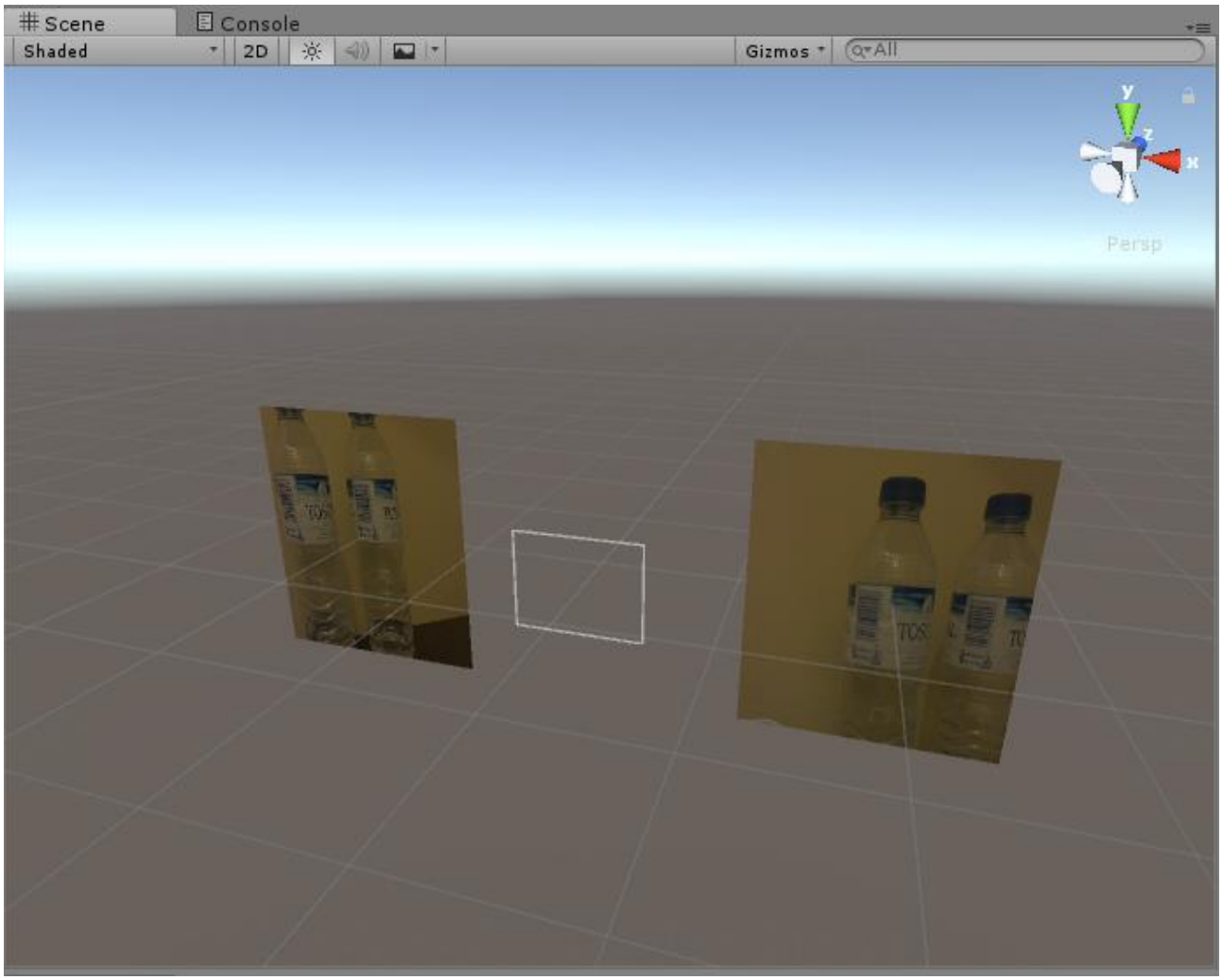

Figura 9: Resultado en la escena de la obtención del vídeo

Una vez se logró obtener la señal de vídeo y mostrarla en la aplicación, había que aplicarle una deformación para que al mirarla a través de las gafas de realidad virtual las imágenes se mostrarán correctamente. Para este propósito se ha utilizado el SDK de Google, Google VR SDK for Unity [6], al que hubo que realizarle unas modificaciones puesto que está preparado para funcionar en una aplicación Android y no en una aplicación para ordenador.

Una vez importado el SDK nos crea en la estructura del proyecto dos carpetas:

- GoogleVR: que contiene todos los scripts, recursos, materiales, etc., que necesita el SDK para funcionar.
- Plugins: que contiene las diferentes librerías y recursos para funcionar en las distintas plataformas (Android, iOS y el editor Unity3D).

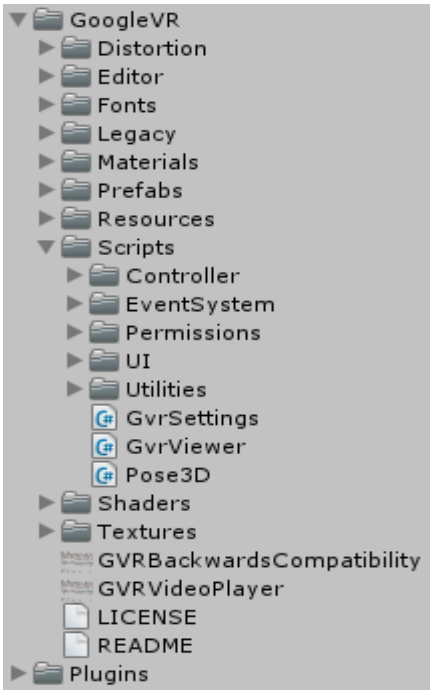

Figura 10: Directorios creados por el SDK GoogleVR

Para que funcione el SDK hay que incorporar a la cámara principal de la escena, “Main Camera”, el script “GvrViewer” que se ve en la imagen anterior, ya que éste es el que crea en la vista de la cámara los diferentes ajustes para la realidad virtual.

Una vez hecho esto, cuando se pone en ejecución la aplicación en el entorno Unity3D, el SDK crea dentro de la “Main Camera” cuatro cámaras más, de las cuales, dos son las que nos interesan a nosotros (Main Camera Left y Main Camera Right) ya que son las que muestran la imagen final al usuario e, inicialmente, no están correctamente colocadas para enfocar a los dos paneles de la escena.

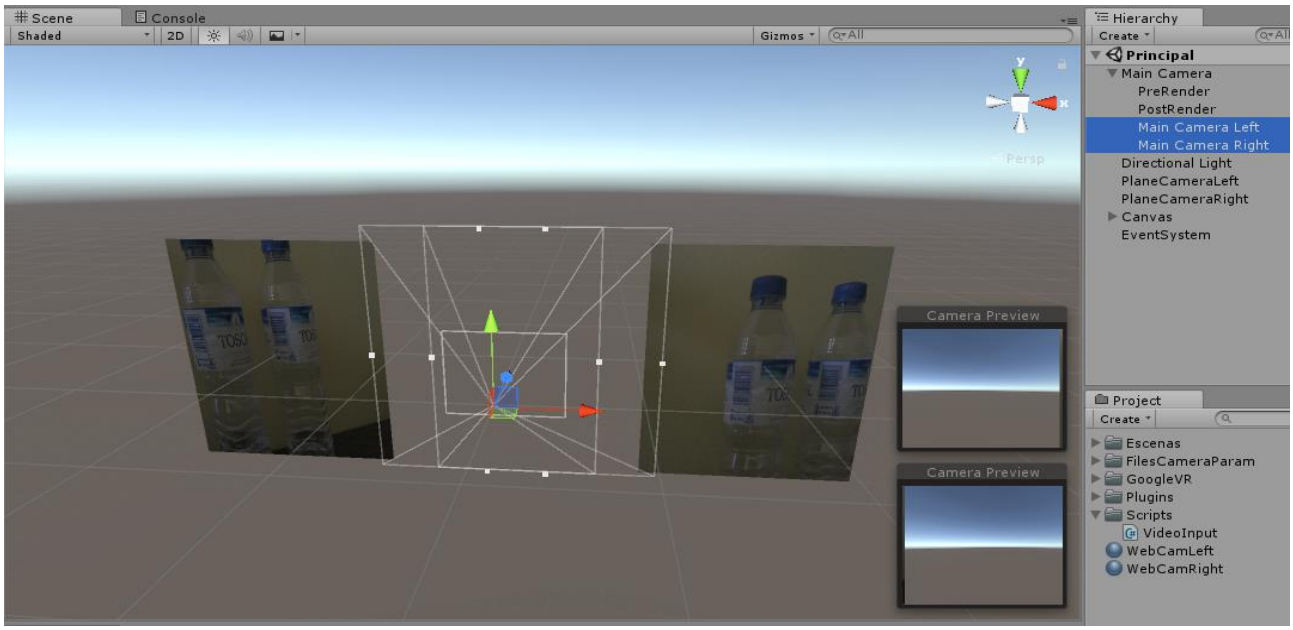

Figura 11: Cámaras del SDK GoogleVR mal posicionadas

Para solucionar esto, se buscó en los scripts de configuración del SDK los ajustes correspondientes, pero no se encontraron, por lo tanto, se optó por modificar la posición de las cámaras en tiempo de ejecución añadiendo a la clase “VideoInput”, vista anteriormente, una función que obtiene las dos cámaras y modifica su posición a otra definida por nosotros en la que se sitúan correctamente sobre los paneles.

```
public class VideoInput : MonoBehaviour {
    // Left Eye
    public Material inputMaterial_leftEye;
    private WebCamTexture camText_leftEye;
    private Vector3 positionEyeLeft = new Vector3 (-2158, 0, 0);
    // Right Eye
    public Material inputMaterial_rightEye;
    private WebCamTexture camText_rightEye
    private Vector3 positionEyeRight = new Vector3 (2158, 0, 0);

    void Start() {
        WebCamDevice[] devices = WebCamTexture.devices;
        Debug.Log ("Detecto " + devices.Length + " dispositivos conectados");

        if (devices.Length < 2) {
            GameObject.Find ("Error").GetComponent<Text> ().enabled = true;
        } else {
            // Left Eye
            camText_leftEye = new WebCamTexture ();
            camText_leftEye.deviceName = devices[0].name;
            camText_leftEye.Play ();
            inputMaterial_leftEye.mainTexture = camText_leftEye;
            // Right Eye
            camText_rightEye = new WebCamTexture ();
            camText_rightEye.deviceName = devices[1].name;
            camText_rightEye.Play ();
            inputMaterial_rightEye.mainTexture = camText_rightEye;

            Invoke ("adjustPositionEyes", 1f);
        }
```

```
    }

    private void adjustPositionEyes() {
        GameObject.Find ("Main Camera Left").transform.position =
positionEyeLeft;
        GameObject.Find ("Main Camera Right").transform.position =
positionEyeRight;
    }
  }
```

Esto soluciona el problema de la localización de las cámaras, pero aún queda otro problema, el SDK de Google no permite funcionar en una aplicación compilada para ordenador. Para solucionar este problema se investigó un poco por Internet y se encontró unas modificaciones a realizar en ciertos archivos de configuración del SDK que permiten que funcione correctamente.

BaseVRDevice.cs

```
#if UNITY_EDITOR || UNITY_STANDALONE
device = new EditorDevice();
#elif ANDROID_DEVICE
// Rest of the code
#endif
```

EditorDevice.cs

```
#if UNITY_EDITOR || UNITY_STANDALONE
using UnityEngine;

private bool RemoteCommunicating {
    get {
        if (!remoteCommunicating) {
            remoteCommunicating = EditorApplication.isRemoteConnected;
            // You have to comment that line
            //remoteCommunicating = EditorApplication.isRemoteConnected;
        }
        return remoteCommunicating;
}
// Rest of the code
#endif
```

EditorDevice.cs

```
#if UNITY_EDITOR || UNITY_STANDALONE
/// Restores level head tilt in when playing in the Unity Editor after you
/// release the Ctrl key.
public bool autoUntiltHead = true;
// The rest of the code
#endif
```

Con estas modificaciones el SDK puede funcionar en una aplicación de ordenador correctamente, consiguiendo así completar los dos primeros objetivos y la primera fase del proyecto. Aun así, las imágenes no se veían correctamente a través de las gafas de realidad virtual debido a que había pequeñas diferencias entre las obtenidas por una cámara y la otra. Esto es debido a que, si la posición de las cámaras en la estructura creada no es exactamente la misma en cuanto a inclinación, las líneas horizontales de las imágenes obtenidas no coinciden entre ellas y producen una sensación de mareo al usuario. Para solucionar esto, es necesario realizar una calibración de las cámaras, lo que nos lleva a la fase siguiente.

Figura 12: Ejecución con pequeñas diferencias en las imágenes

3.2 Fase 2: Calibración de las imágenes de las cámaras

En esta fase lo que se pretende es realizar una calibración a las cámaras para obtener las imágenes correctamente niveladas, es decir, que las líneas horizontales de las imágenes tomadas por una cámara coincidan con las líneas horizontales tomadas por la otra.

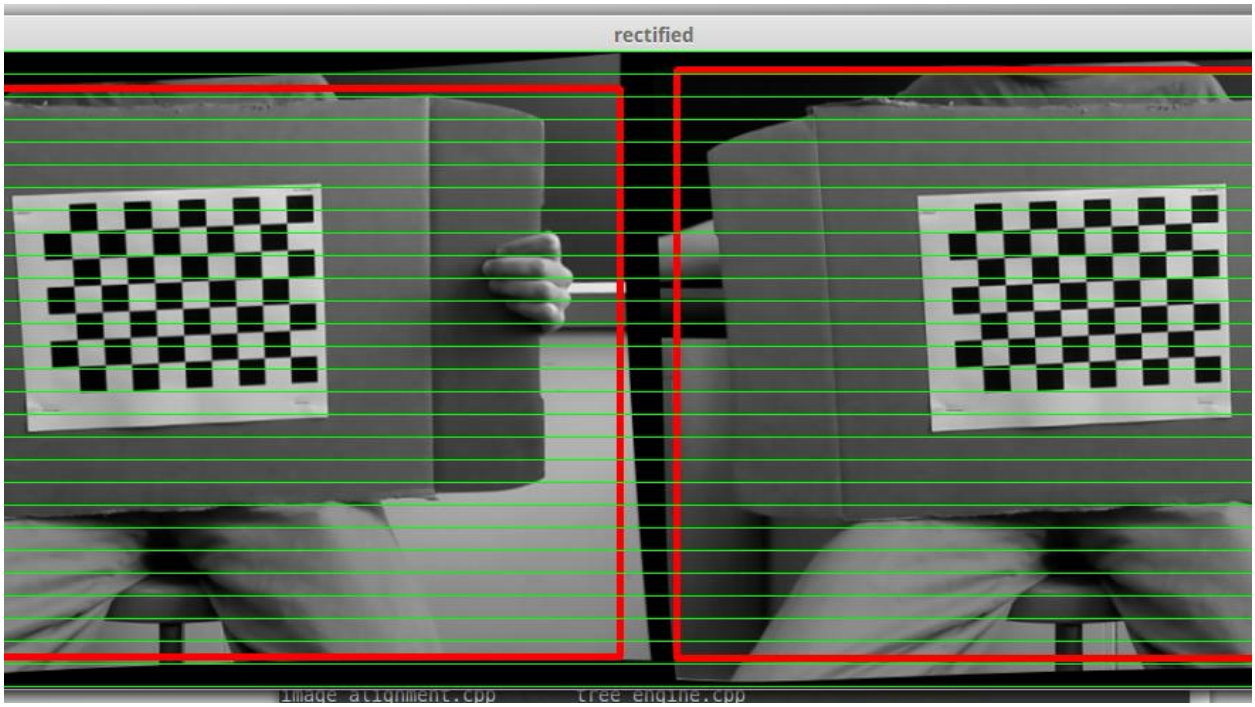

Figura 13: Líneas horizontales coincidiendo en un par estéreo calibrado

Esto hace que las imágenes se puedan ver a través de las gafas de realidad virtual sin producir la sensación de mareo. Para ello, se hizo uso de la librería OpenCV [7], que es una librería libre de visión artificial que ofrece múltiples funciones para trabajar con imágenes. Es empleada para sistemas de detección de movimiento, reconocimiento de objetos, etc. Dentro de las funciones de que dispone, tiene las que se necesitan para este proyecto, es decir, para realizar la calibración de un par estéreo.

El programa creado para realizar la calibración fue desarrollado con los lenguajes de programación C y C++.

Figura 14: Menú del programa de calibración en ejecución

```
class CalibrationTools {
  private:
      bool calibrationComplete = false;
      int numBoards = 10;           // Number of images to take
      int board_w = 7;         // Horizontal corners
      int board_h = 5;         // Vertical corners
      int board_n = board_w * board_h;          // Counter for loops
      Size board_sz = Size (board_w, board_h); // Size of board
      vector<vector<Point3f> > object_points;   // Real location of corners in 3D
      vector<vector<Point2f> > imagePoints1, imagePoints2; // Location of the
corners in the image
      vector<Point2f> corners1, corners2;
```

```
    vector<Point3f> obj;    // To insert elements in object_points
    Size imageSize;
    Mat CM1 = Mat (3, 3, CV_64FC1);     // Camera matrix 1
    Mat CM2 = Mat (3, 3, CV_64FC1);     // Camera matrix 2
    Mat D1, D2;                  // Distotion coefficient
    Mat R, T, E, F;              // Rotation matrix and traslation matrix
    Mat R1, R2, P1, P2, Q;              // Parameters to stereoRectify
    Mat map1x, map1y, map2x, map2y;     // RMAPS parameters to undistort

public:
    CalibrationTools ();
    void Menu ();
    void ClearScreen ();
    void PressKeyToContinue ();
    void CaptureAndFindChessBoardCorners ();
    void Calibrate ();
    void Rectify ();
    void WriteFiles ();
    void UndistortRectify ();
    void PlayLiveVideo ();

};
```

En dicho programa se implementaron varias funciones:

- La función para la calibración.
- Ver el vídeo en directo una vez calibrado para comprobar si el proceso se realizó correctamente.
- Guardar los parámetros de calibración en archivos para su reutilización u observación.

El proceso de calibración contiene varias funciones para realizarlo correctamente:

```
case '1':
{
      CaptureAndFindChessBoardCorners ();
      Calibrate ();
      Rectify ();
      UndistortRectify ();
      calibrationComplete = true;

} break;
```

1. *CaptureAndFindChessBoardCorners*: Esta función toma un cierto número de imágenes (definido en el código), en tiempo de ejecución a medida que el usuario pulsa una tecla, de un tablero de ajedrez y detecta los cruces entre celdas con la función de OpenCV “findChessboardCorners()”.

```
while (success < numBoards) {
      cap1 >> img1;
      cap2 >> img2;
      cvtColor (img1, gray1, CV_BGR2GRAY);
      cvtColor (img2, gray2, CV_BGR2GRAY);

      found1 = findChessboardCorners (img1, board_sz, corners1,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);
```

```
      found2 = findChessboardCorners (img2, board_sz, corners2,
CV_CALIB_CB_ADAPTIVE_THRESH | CV_CALIB_CB_FILTER_QUADS);

      if (found1) {
            cornerSubPix (gray1, corners1, Size (11, 11), Size (-1, -1),
TermCriteria (CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
            drawChessboardCorners (gray1, board_sz, corners1, found1);
      }

      if (found2) {
            cornerSubPix (gray2, corners2, Size (11, 11), Size (-1, -1),
TermCriteria (CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 30, 0.1));
            drawChessboardCorners (gray2, board_sz, corners2, found2);
      }

      imshow ("image1", gray1);
      imshow ("image2", gray2);

      k = waitKey (10);
      if (k == ' ' && found1 != 0 && found2 != 0) {
            imagePoints1.push_back (corners1);
            imagePoints2.push_back (corners2);
            success++;
            cout << "Image " << success << " of " << numBoards << " and corners
stored" << endl;

            if (success >= numBoards) {
                  break;
            }
      }

  }
```

Esta función recibe como parámetros la imagen tomada, el tamaño del tablero (número de celdas horizontales y verticales), un vector para guardar la posición de las esquinas de las celdas y unos parámetros necesarios para realizar las operaciones. Una vez detectados los cruces entre celdas, dibuja, en las imágenes, las líneas que se ven en la imagen a continuación y se las muestra al usuario para, posteriormente, almacenar el vector de las posiciones de las esquinas de ambas imágenes.

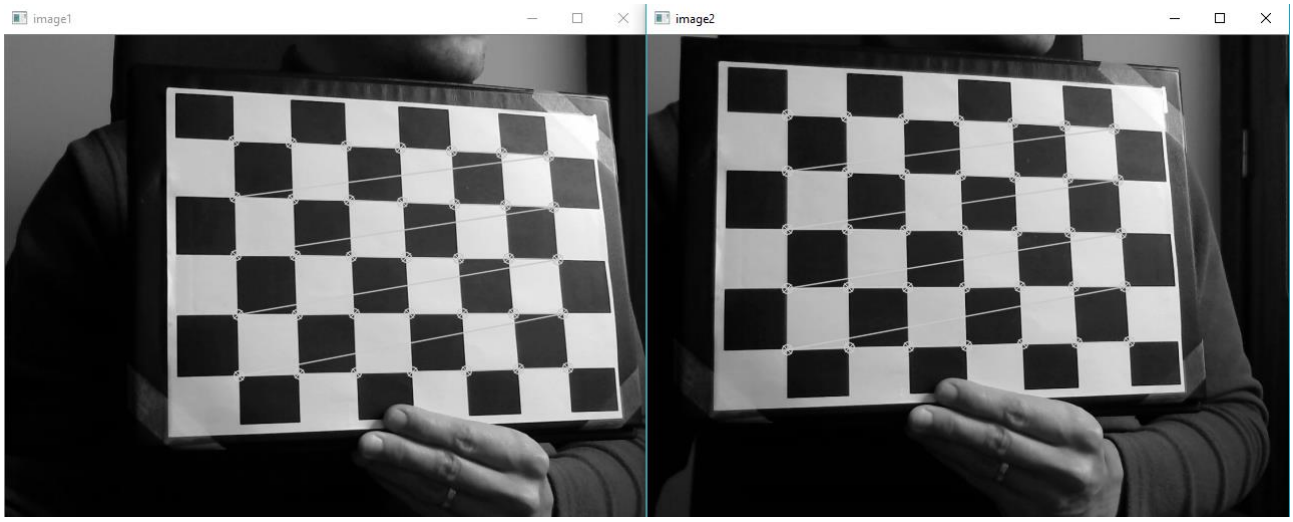

Figura 15: OpenCV detectando esquinas de tablero

2. *Calibrate*: Esta función realiza, utilizando los vectores de salida de la función anterior, el proceso de calibración. Para ello se hace uso de la función de OpenCV "stereoCalibrate" que recibe como parámetros:
    a. Un vector de puntos en 3D que simulan la localización real de los cruces del tablero (object_points).

```
for (int n = 0; n < numBoards; n++) {
	for (int i = 0; i < board_h; i++) {
		for (int j = 0; j < board_w; j++) {
			object_points[n].push_back (Point3f ((float)j*1.0, (float)i*1.0, 0));
		}
	}
}
```

    b. Los vectores de las posiciones de las esquinas de las imágenes obtenidos en la función anterior.
    c. Una matriz de salida que almacenará los parámetros de calibración de una de las cámaras.
    d. Una matriz de salida que almacenará los coeficientes de distorsión de la cámara anterior.
    e. Otras dos matrices de salida para almacenar los parámetros de calibración de la otra cámara y sus coeficientes de distorsión.
    f. El tamaño de las imágenes capturadas.
    g. Una matriz para almacenar la rotación entre los sistemas de coordenadas de las cámaras.
    h. Una matriz para almacenar la traslación entre los sistemas de coordenadas de las cámaras.
    i. Dos matrices para almacenar, en una, parámetros esenciales, y en otra, parámetros fundamentales.
    j. Otros parámetros necesarios para realizar el proceso de calibración.

```
void CalibrationTools::Calibrate () {
	cout << "-> Starting calibration" << endl;

	stereoCalibrate (object_points, imagePoints1, imagePoints2,
		CM1, D1, CM2, D2, imageSize, R, T, E, F,
		CV_CALIB_SAME_FOCAL_LENGTH | CV_CALIB_ZERO_TANGENT_DIST,
		cvTermCriteria (CV_TERMCRIT_ITER + CV_TERMCRIT_EPS, 100, 1e-5));

	cout << "-> Done!" << endl;
	PressKeyToContinue ();

}
```

3. *Rectify*: Esta función realiza el cálculo de las transformaciones de rectificación para cada cámara, una vez están calibradas. Para esto,

se hace uso de la función de OpenCV “stereoRectify”, que recibe como parámetros todas las matrices de salida de la función anterior (salvo los parámetros esenciales y fundamentales), el tamaño de las imágenes y:

a. Dos matrices de salida R1 y R2 para almacenar los parámetros de rectificación de la rotación de cada cámara.
b. Dos matrices de salida P1 y P2 para almacenar los parámetros de proyección rectificados de los sistemas de coordenadas de cada cámara.
c. Una matriz de salida Q para almacenar parámetros de mapeo disparidad-profundidad.

```
void CalibrationTools::Rectify () {
	cout << "-> Starting rectification" << endl;

	stereoRectify (CM1, D1, CM2, D2, imageSize, R, T, R1, R2, P1, P2, Q);

	cout << "-> Done!" << endl;
	PressKeyToContinue ();

}
```

4. *UndistortRectify*: Esta función calcula los mapas de rectificación y eliminación de la distorsión para transformar las imágenes obtenidas por las cámaras. Para ello, se hace uso de la función de OpenCV “initUndistortRectifyMap” una vez para cada cámara. Recibe como parámetros:

a. La matriz de parámetros para la calibración de la cámara (CMX).
b. La matriz de los coeficientes de distorsión (DX).
c. La matriz de rectificación de la rotación (RX).
d. La matriz de parámetros de proyección rectificados (PX).
e. El tamaño de las imágenes.
f. El tipo de los datos que se almacenarán en las matrices de salida (mapxX y mapyX).
g. Mapa de salida para el eje x (mapx).
h. Mapa de salida para el eje y (mapy).

```
void CalibrationTools::UndistortRectify () {
	cout << "-> Applying undistort" << endl;

	initUndistortRectifyMap (CM1, D1, R1, P1, imageSize, CV_32FC1, map1x, map1y);
	initUndistortRectifyMap (CM2, D2, R2, P2, imageSize, CV_32FC1, map2x, map2y);

	cout << "-> Done!" << endl;
	PressKeyToContinue ();

}
```

Una vez ha terminado el proceso de calibración ya se puede acceder a las otras dos funciones del programa, ver el vídeo en directo y guardar los parámetros.

La función que realiza el proceso para mostrar el vídeo en directo se llama “PlayLiveVideo” y consiste en un bucle que, en cada iteración, obtiene una imagen de las cámaras, les aplica una función de OpenCV llamada “remap” y la muestra al usuario para que compruebe si el proceso de calibración se realizó correctamente. Esta función “remap” lo que realiza es una transformación geométrica de la imagen utilizando los mapas obtenidos en el proceso de calibración. Recibe como parámetros:

a) La matriz que contiene la imagen de origen tomada desde la cámara.

b) La matriz que contendrá la imagen de destino una vez transformada.

c) Los mapas generados anteriormente para la cámara en cuestión.

d) Tres parámetros más (interpolación, modo de borde de la imagen y valor de ese borde), que se dejaron por defecto.

```
void CalibrationTools::PlayLiveVideo () {
	cout << "-> Playing live video" << endl;
	cout << "-> Press ESC to finish" << endl;
	VideoCapture cap1 = VideoCapture (0);
	VideoCapture cap2 = VideoCapture (1);
	Mat img1, img2, imgU1, imgU2;
	int key = 0;

	while (1) {
		cap1 >> img1;
		cap2 >> img2;

		remap (img1, imgU1, map1x, map1y, INTER_LINEAR, BORDER_CONSTANT,
Scalar ());
		remap (img2, imgU2, map2x, map2y, INTER_LINEAR, BORDER_CONSTANT,
Scalar ());

		imshow ("image1", imgU1);
		imshow ("image2", imgU2);

		key = waitKey (5);

		if (key == 27) {
			break;
		}
	}
	destroyAllWindows ();
	cap1.release ();
	cap2.release ();

}
```

Por último, quedaría la función que realiza el guardado de los parámetros en archivos. Esta función se llama “WriteFiles” y lo que realiza, utilizando la clase de OpenCV “FileStorage”, es el guardado, en un archivo, de todas las matrices almacenadas en el programa durante el proceso de calibración y

que contienen los parámetros para su reutilización u observación, por otro lado, en otro archivo, los mapas generados para obtener las imágenes calibradas.

```
void CalibrationTools::WriteFiles () {
	cout << "-> Saving parameters" << endl;
	FileStorage fs1 ("mystereocalib.yml", FileStorage::WRITE);
	if (fs1.isOpened ()) {
		fs1 << "CM1" << CM1;
		fs1 << "CM2" << CM2;
		fs1 << "D1" << D1;
		fs1 << "D2" << D2;
		fs1 << "R" << R;
		fs1 << "T" << T;
		fs1 << "E" << E;
		fs1 << "F" << F;
		fs1 << "R1" << R1;
		fs1 << "R2" << R2;
		fs1 << "P1" << P1;
		fs1 << "P2" << P2;
		fs1 << "Q" << Q;
		fs1.release ();
	} else {
		printf ("Error: Could not open mystereocalib file\n");
	}

	fs1.open ("rmaps.yml", FileStorage::WRITE);
	if (fs1.isOpened ()) {
		fs1 << "M00" << map1x << "M01" << map1y << "M10" << map2x << "M11"
<< map2y;
		fs1.release ();
	} else {
		cout << "Error: Could not open rmaps file" << endl;
	}
	cout << "-> Done!" << endl;
	PressKeyToContinue ();

}
```

Finalizado este paso, se tienen los ajustes necesarios para obtener unas imágenes calibradas y ver a través de las gafas de realidad virtual correctamente, pero, para ello, hay que modificar el programa de Unity3D para que implemente la función de remapeo (remap) de las imágenes obtenidas.

Este paso no fue tarea fácil, ya que OpenCV no está disponible para Unity3D ni tampoco para C# de forma oficial, por lo que se realizó una búsqueda y se encontró una librería que es una adaptación de OpenCV para utilizarlo en Unity3D cuyo nombre es “OpenCV for Unity” [8]. Después de ver cómo funcionaba la librería e implementar parte del código del programa, llegó un punto en el que no funcionaba y no se encontraba solución, por lo que se tomó como un problema de la librería y se pasó a buscar alternativas para conseguir el objetivo.

Después de buscar alternativas se encontró otra librería, esta vez para C#, llamada “OpenCVSharp” [9] que es otra adaptación de OpenCV para C#.

Se procedió a modificar el código que ya se tenía de Unity3D para añadirle la lectura de los mapas generados en la calibración. Para ello se creó una variable de tipo “string” que contendría la ruta del archivo a leer, otra de tipo “CvMat” (un tipo de matriz que se utiliza en OpenCV) y se creó una función que, utilizando las clases “CvFileStorage” y “CvFileNode” proporcionadas por OpenCVSharp, leería los mapas guardados en un archivo .yml [10]. Además, se añadió una condición para, en caso de que el proceso de lectura se realizase correctamente, continuar con la ejecución del programa, en caso contrario, mostrar un mensaje de error.

```
private string pathRmapsFile = "./Assets/FilesCameraParam/rmaps.yml";
private CvMat[,] rmap = new CvMat[2, 2];

void Start() {
    WebCamDevice[] devices = WebCamTexture.devices;
    Debug.Log ("Detecto " + devices.Length + " dispositivos conectados");

    if (devices.Length < 2) {
        GameObject.Find ("Error").GetComponent<Text> ().enabled = true;
    } else {
        // Read Rmaps to modify the images
        bool correct = readRmapParameters ();

        if (correct) {
            // Left Eye
            camText_leftEye = new WebCamTexture ();
            camText_leftEye.deviceName = devices[0].name;
            camText_leftEye.Play ();
            inputMaterial_leftEye.mainTexture = camText_leftEye;
            // Right Eye
            camText_rightEye = new WebCamTexture ();
            camText_rightEye.deviceName = devices[1].name;
            camText_rightEye.Play ();
            inputMaterial_rightEye.mainTexture = camText_rightEye;

            Invoke ("adjustPositionEyes", 1f);
        }
    }

}

private bool readRmapParameters() {
        try {
          // Read M00
            CvFileStorage read = new CvFileStorage (pathRmapsFile, null,
FileStorageMode.Read);
            CvFileNode param = read.GetFileNodeByName (null, "M00");
            rmap[0, 0] = read.Read<CvMat> (param);

            // Read M01
            read = new CvFileStorage (pathRmapsFile, null, FileStorageMode.Read);
            param = read.GetFileNodeByName (null, "M01");
            rmap[0, 1] = read.Read<CvMat> (param);

            // Read M10
            read = new CvFileStorage (pathRmapsFile, null, FileStorageMode.Read);
            param = read.GetFileNodeByName (null, "M10");
            rmap[1, 0] = read.Read<CvMat> (param);

            // Read M11
            read = new CvFileStorage (pathRmapsFile, null, FileStorageMode.Read);
            param = read.GetFileNodeByName (null, "M11");
            rmap[1, 1] = read.Read<CvMat> (param);
            return true;
        } catch (Exception e) {
            GameObject.Find ("Error").GetComponent<Text> ().enabled = true;
```

```
            GameObject.Find ("Error").GetComponent<Text> ().text = "Error al leer el
   archivo rmap " + e;
            return false;
        }

    }
```

Una vez leídos los mapas, se procedió a realizar el remapeo de las imágenes obtenidas por las cámaras en tiempo real.

```
private IEnumerator editStreamingVideo() {
        // Convert image to CvMat
        Color[] pixels_left = camText_leftEye.GetPixels ();
        Color[] pixels_right = camText_rightEye.GetPixels ();
        CvMat imageConvertedLeft = new CvMat (imgHeight, imgWidth, MatrixType.U8C3);
        CvMat imageConvertedRight = new CvMat (imgHeight, imgWidth, MatrixType.U8C3);

        for (int i = 0; i < imgHeight; i++) {
            for (int j = 0; j < imgWidth; j++) {
                Color pixel_left = pixels_left[j + i * imgWidth];
                Color pixel_right = pixels_right[j + i * imgWidth];
                CvScalar scalar_left = new CvScalar (pixel_left.b * 255, pixel_left.g *
255, pixel_left.r * 255);
                CvScalar scalar_right = new CvScalar (pixel_right.b * 255, pixel_right.g
* 255, pixel_right.r * 255);
                imageConvertedLeft.Set2D ((imgHeight - 1) - i, (imgWidth - 1) - j,
scalar_left);
                imageConvertedRight.Set2D ((imgHeight - 1) - i, (imgWidth - 1) - j,
scalar_right);
            }
        }

        // Remapping
        CvMat outImg_left = new CvMat (imgHeight, imgWidth, MatrixType.U8C3);
        CvMat outImg_right = new CvMat (imgHeight, imgWidth, MatrixType.U8C3);
        Cv.Remap (imageConvertedLeft, outImg_left, rmap[0, 0], rmap[0, 1],
Interpolation.Linear);
        Cv.Remap (imageConvertedRight, outImg_right, rmap[1, 0], rmap[1, 1],
Interpolation.Linear);

        // Convert image to Texture2D
        IplImage converted_left = Cv.GetImage (outImg_left);
        IplImage converted_right = Cv.GetImage (outImg_right);
        Texture2D videoTexture_left = new Texture2D (imgWidth, imgHeight,
TextureFormat.RGB24, false);
        Texture2D videoTexture_right = new Texture2D (imgWidth, imgHeight,
TextureFormat.RGB24, false);
        for (int i = 0; i < imgWidth; i++) {
            for (int j = 0; j < imgHeight; j++) {
                float b_left = (float) converted_left[j, i].Val0;
                float g_left = (float) converted_left[j, i].Val1;
                float r_left = (float) converted_left[j, i].Val2;
                float b_right = (float) converted_right[j, i].Val0;
                float g_right = (float) converted_right[j, i].Val1;
                float r_right = (float) converted_right[j, i].Val2;
                Color color_left = new Color (r_left / 255.0f, g_left / 255.0f, b_left /
255.0f);
                Color color_right = new Color (r_right / 255.0f, g_right / 255.0f,
b_right / 255.0f);
                videoTexture_left.SetPixel (imgWidth - i, imgHeight - j, color_left);
                videoTexture_right.SetPixel (imgWidth - i, imgHeight - j, color_right);
            }
        }
        videoTexture_left.Apply ();
        videoTexture_right.Apply ();
        inputMaterial_leftEye.mainTexture = videoTexture_left;
        inputMaterial_rightEye.mainTexture = videoTexture_right;
        yield return null;

}
```

Esta función lo que hace es:

1. Obtiene los pixeles de cada imagen de las cámaras y los almacena en dos arrays tipo “Color”. Este tipo “Color” lo que almacena es un color RGB.
2. Crea dos matrices tipo “CvMat” para almacenar el contenido de los arrays anteriores y operar con ellas en el resto del procedimiento. Para ello se utilizan dos bucles anidados que obtienen de cada array, en la posición indicada, el valor “Color” almacenado, luego se transforma en un tipo “CvScalar” que es el que almacenan los “CvMat” y posteriormente se insertan en las matrices en la posición correspondiente (en el proyecto se realiza de la posición (n,m) a la (0,0) porque, al obtener las imágenes, éstas están invertidas.
3. Crea dos matrices del mismo tipo que las anteriores para almacenar la salida de la función de remapeo. Luego, se realiza la propia función de remapeo que aplica las transformaciones correspondientes indicadas por los mapas leídos anteriormente y almacena el resultado.
4. Para finalizar, esas matrices obtenidas con las imágenes correctamente calibradas se transforman en “Texture2D” para aplicarlas al material que llevan los paneles de la escena y que el usuario las pueda ver.

Una vez hecho esto, como resultado se obtiene que el usuario puede ver a través de las gafas de realidad virtual sin que le produzca mareo el hacerlo, por tanto, se da por completado el tercer objetivo del proyecto y por finalizada la segunda fase.

3.3 Fase 3: Obtención de los datos de la diadema Emotiv y movimiento de los servomotores

En esta fase se pretendía obtener los datos de los giroscopios de la diadema Emotiv para interpretarlos y, posteriormente, mover los servomotores.

El movimiento que realizarían los servomotores sería una traducción del movimiento de la cabeza del usuario, es decir, cuando se detecta que el usuario está realizando un movimiento de cabeza, el programa comienza a modificar los ángulos de los motores para ir traduciendo la posición de la cabeza del usuario a los servomotores en función de un incremento que se puede ajustar para adaptar la herramienta a diferentes usuarios. En cuanto a los límites alcanzables por el movimiento de la cabeza del usuario, corresponderían a los límites alcanzables por los servomotores.

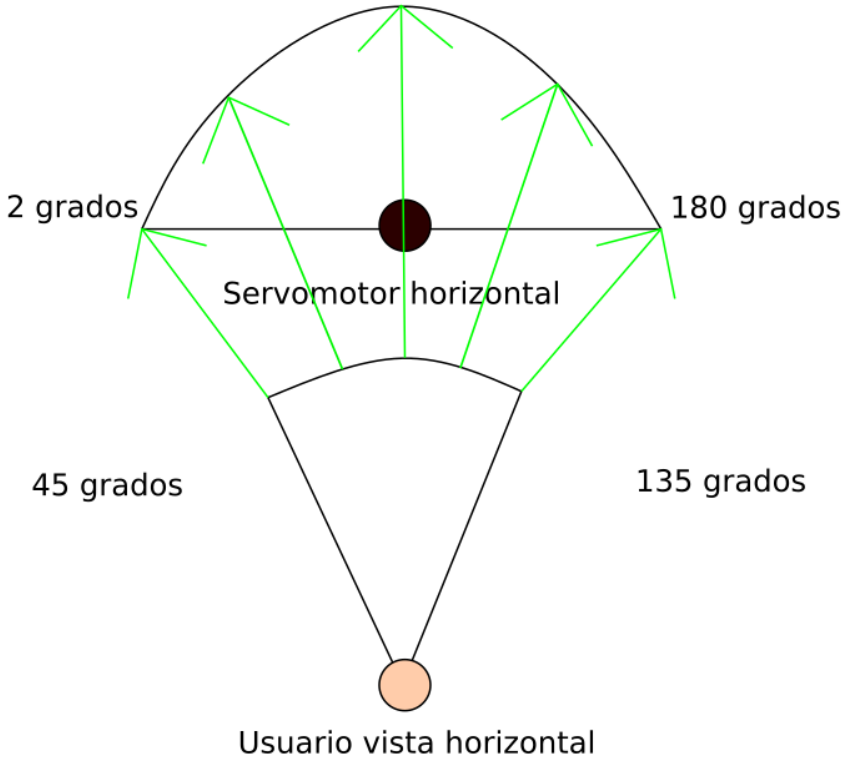

Figura 16: Traducción del movimiento horizontal al servomotor

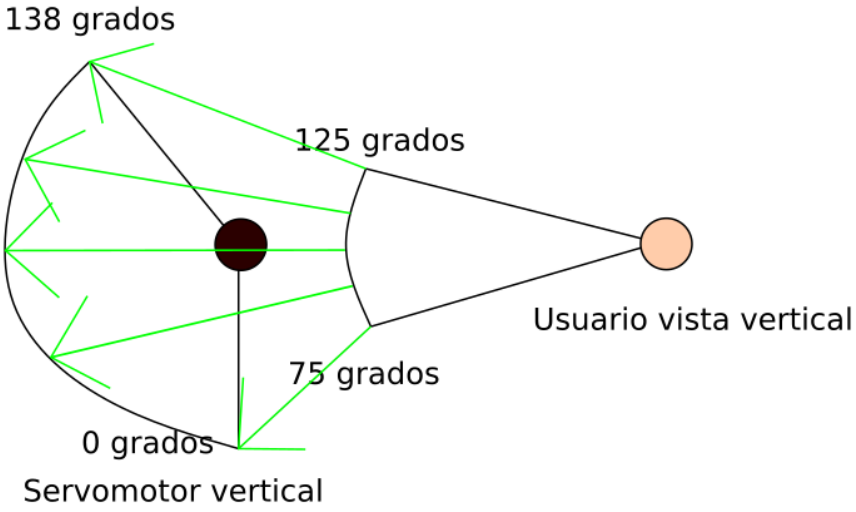

Figura 17: Traducción del movimiento vertical al servomotor

Como ejemplo, supongamos que el usuario solo puede realizar un movimiento en horizontal en el rango de 45º a 135º, estos ángulos extremos corresponderían al rango máximo alcanzable por el servomotor que realiza el movimiento en horizontal, en este proyecto, la limitación física del servomotor solo permite de 2º a 180º por lo que el rango del usuario equivaldría a esos límites (si el servomotor tuviera un rango más amplio correspondería igualmente al rango máximo alcanzable por este). El caso del movimiento vertical sería similar.

Actualmente, el proyecto cuenta con un valor de incremento modificable que permite incrementar/disminuir el ángulo de los servomotores con mayor o menor velocidad permitiendo adaptar la herramienta a diferentes usuarios y grados de discapacidad.

Para la realización de esta fase se comenzó creando una clase en C y C++, haciendo uso del SDK de Emotiv, con el objetivo de obtener los datos de los giroscopios y ver de qué forma se podrían utilizar para mover los servomotores.

```
class Detection {
	private:
		EmoEngineEventHandle eventHandle;
		unsigned int userID;
	public:
		Detection();
		void startProgram();
		bool connectEmotiv();
		bool calibrateEmotiv();
		bool exitProgram();
		void pauseProgram();

};
```

Esta clase tiene una función principal “startProgram” que lleva el flujo del programa encargándose de llamar a las otras cuando sea necesario. También, una vez se han realizado los procesos de conexión a la diadema y calibración de la misma, se encarga, mediante un bucle, de obtener los datos.

```
void Detection::startProgram() {
	system("CLS");
	bool correct = connectEmotiv();

	if (correct) {
		Sleep(2000);
		system("CLS");

		correct = calibrateEmotiv();
		if (correct) {
			Sleep(2000);
			system("CLS");

			int dx, dy;

			while (true) {
				if (IEE_HeadsetGetGyroDelta(userID, &dx, &dy) == EDK_OK)
				{
					printf("x: %d	y: %d \n", dx, dy);
				}
				Sleep(20);
			}
		}
	}
}
```

Nada más comenzar la función, lo que realiza es la conexión a la diadema Emotiv, llamando a la función correspondiente “connectEmotiv”. Esta función

se encarga, además de indicarle al usuario lo que está sucediendo en el programa, de realizar la propia conexión, a través de Bluetooth, mediante las funciones proporcionadas por el SDK y un bucle que itera hasta que la conexión se realiza correctamente. Una vez sucede esto, retorna “true” para que la función principal sepa que el proceso se realizó correctamente y siga con el flujo del programa.

Esta parte dio algunos problemas al inicio porque no se conectaba con la diadema, necesitaba un ordenador con Bluetooth 4.0 e inicialmente no se disponía de él. Cuando tuve el ordenador ya disponible, aún seguía dando problemas, se conectaba de forma aleatoria, es decir, algunas veces sí y otras no. Después de estar investigando un tiempo, se encontró, inexplicablemente, que la solución era abrir el panel de control del SDK, realizar la conexión a la diadema para que el programa se pusiera a buscar el dispositivo, cerrar el panel de control y ya mi programa detectaba, para esa ejecución, el dispositivo.

Este proceso hay que realizarlo siempre que se quisiera conectar a la diadema puesto que creo que el problema viene por parte del SDK.

```
bool Detection::connectEmotiv() {
    printf("======= CONECTANDO EMOTIV =======\n");
    if (IEE_EngineConnect() != EDK_OK) {
          printf("Error en la conexion \n");
          return exitProgram();
    }

    int state = 0;
    bool ready = false;
    while (!ready) {
          state = IEE_EngineGetNextEvent(eventHandle);

          if (state == EDK_OK) {
                IEE_Event_t eventType=IEE_EmoEngineEventGetType(eventHandle);
                IEE_EmoEngineEventGetUserId(eventHandle, &userID);

                if (eventType == IEE_UserAdded) {
                      printf("Dispositivo conectado \n\n");
                      userID = 0;
                      ready = true;
                }
          } else {
                printf("Buscando dispositivo.. \r");
          }

          Sleep(200);
    }
    return true;

}
```

Una vez que la conexión se realiza correctamente, se pasa a la calibración de la diadema. Para ello, al igual que en el paso anterior, se llama a su función correspondiente “calibrateEmotiv”, esta le indica al usuario lo que

está sucediendo y realiza el proceso de calibración devolviendo “true” una vez realizado el proceso correctamente.

```
bool Detection::calibrateEmotiv() {
    printf("======= CALIBRANDO EMOTIV =======\n");
    printf("\nPor favor, asegura que tu dispositivo esta encendido y no muevas
la cabeza \n\n");

    bool calibrated = false;
    while (!calibrated) {
          int gyroX = 0;
          int gyroY = 0;
          int err = IEE_HeadsetGetGyroDelta(userID, &gyroX, &gyroY);

          if (err == EDK_OK){
                printf("\nDispositivo calibrado \n");
                calibrated = true;
          }else if (err == EDK_GYRO_NOT_CALIBRATED){
                printf("El giroscopio no esta calibrado. Por favor, no te
muevas \r");
          }else if (err == EDK_CANNOT_ACQUIRE_DATA){
                printf("No se pueden obtener datos \r");
          }else{
                printf("No hay un dispositivo conectado \r");
          }
          Sleep(200);
    }
    return true;

}
```

Esta parte también dio algunos problemas ya que, con la diadema colocada en la cabeza, no se conseguía realizar la calibración, aunque no se realizara ningún movimiento. Afortunadamente, este problema no llevó tanto tiempo como el anterior puesto que su solución es colocar la diadema en una superficie estable e inmóvil de forma que la diadema quede en una posición horizontal y no reciba ningún movimiento.

Realizada la calibración, ya estaba todo preparado para obtener los datos y empezar a definir la estrategia para su utilización.

La estrategia empleada fue la definición de unos límites fijos en el código que, al girar la cabeza, si los datos que devolvía la diadema superaban esos límites, significaría que los servomotores se tendrían que mover en la dirección indicada. Para implementarla se creó una función “moveCameras”, llamada desde el bucle que obtiene los datos, que recibiría como parámetros los valores de la diadema y los compararía con los límites a ver si son superados.

```
void Detection::moveCameras (const int x, const int y) {
      printf("x: %d   y: %d\r", x, y);
      if (x >= limitGoRight) {
            // Right
      } else if (x <= limitGoLeft) {
            // Left
      }
      if (y >= limitGoUp) {
```

```
                        // Up
                } else if (y <= limitGoDown) {
                        // Down
                }

        }
```

Esta estrategia no fue del todo bien puesto que, los valores obtenidos de la diadema variaban demasiado y, además, según la posición/inclinación en la que se haya calibrado la diadema, por mínima que fuera la variación de una vez a otra, los límites posiblemente no estarían bien fijados. Para solucionar esto, se creó otra función "defineMovementLimitsCameras" que se encargaría de definir los límites en tiempo de ejecución, permitiendo así que el usuario los estableciera donde más cómodo le resultara a la vez que se intentaba controlar esa variación de los valores obtenidos.

Esta función es llamada una vez se ha conectado correctamente el programa a la diadema y antes de entrar al bucle que mueve las cámaras.

```
void Detection::defineMovementLimitsCameras() {
      printf("======= DEFINIENDO LIMITES PARA ACTIVACION DEL MOVIMIENTO
      =======\n\n");
      bool exit = false;
      int movement = 0;
      while (!exit) {
            switch (movement) {
                  case 0: {
                        printf("-> Limite derecho: Pulsa una tecla para
                        establecer el limite de activacion indicado \n");
                        bool defined = false;
                        int x = 0, dx, dy;
                        while (!defined) {
                              if (IEE_HeadsetGetGyroDelta(userID, &dx, &dy) ==
                              EDK_OK) {
                                    x += dx;
                                    printf("--> %d\r", x);

                                    if (_kbhit()) {
                                          _getch();
                                          limitGoRight = x;
                                          defined = true;
                                          movement = 1;
                                          printf("Limite establecido.\n\n");
                                    }
                              }
                              Sleep(200);
                        }
                        pauseProgram();
                  } break;
```

La función establece los límites para los movimientos de izquierda, derecha, arriba y abajo. El usuario giraría la cabeza en la dirección indicada y cuando quisiera establecer el límite pulsaría una tecla.

Con esto, se solucionó el problema de los límites y solamente quedaría conectar a la placa de Arduino y enviarle los datos que, por otro lado, recibiría otro programa en dicha placa que también faltaría crear.

Para la conexión y el envío de los datos se creó una clase "Serial" sencilla

```
class Serial
{
    private:
        HANDLE hSerial;
        bool connected;
        COMSTAT status;
        DWORD errors;

    public:
        Serial(const char *portName);
        ~Serial();
        bool WriteData(const char *buffer, unsigned int nbChar);
        bool IsConnected();

};
```

que es instanciada dentro de la clase “Detection” y, en su constructor, realiza la conexión a la placa Arduino a través del puerto serie. También, tiene una función “WriteData” que se encargaría de enviar los datos a la placa.

```
bool Serial::WriteData(const char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    //Try to write the buffer on the Serial port
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //In case it don't work get comm error and return false
        ClearCommError(this->hSerial, &this->errors, &this->status);

        return false;
    }
    else
        return true;

}
```

Esta función recibe como parámetros la cadena de caracteres a enviar y el número de caracteres que la componen. Luego, realiza el intento de enviar la información y, en caso de enviarla correctamente, retorna “true”, si no, retorna “false”. Por último, esta función es llamada desde “moveCameras” dentro de las condiciones correspondientes.

El programa creado para la placa Arduino [11] realiza la conexión a los servomotores en los puertos 3 (para rotación) y 6 (para inclinación). Luego, se creó un bucle en el que se realiza una lectura, en cada iteración, del puerto serie de la placa con el fin de comprobar si hay parámetros. En caso afirmativo, lo clasifica, realiza la comprobación de si el ángulo recibido está dentro de los límites físicos de giro del motor y, si lo está, realiza el movimiento oportuno de los motores en la dirección correspondiente utilizando las funciones “rotateServo” e “inclineServo”.

```
void setup(){
  servoRotation.attach(3);
  servoInclination.attach(6);
  Serial.begin(9600);
}

void loop(){
  unsigned char command=0;
```

```
  if(Serial.available()){
    command=Serial.read();

    switch(command) {
      case 'a': {
        if (angleRotation > LIMIT_LOWER_ROTATION) {
          angleRotation-=INCREMENT;//decrementamos 10
          rotateServo(servoRotation, angleRotation);
        }
      } break;
      case 'd': {
        if (angleRotation < LIMIT_UPPER_ROTATION) {
          angleRotation+=INCREMENT;//incrementamos 10
          rotateServo(servoRotation, angleRotation);
        }
      } break;
      case 's': {
        if (angleInclination < LIMIT_UPPER_INCLINATION) {
          angleInclination+=INCREMENT;//incrementamos 10
          inclineServo(servoInclination, angleInclination);
        }
      } break;
      case 'w': {
        if (angleInclination > LIMIT_LOWER_INCLINATION) {
          angleInclination-=INCREMENT;//decrementamos 10
          inclineServo(servoInclination, angleInclination);
        }
      } break;
    }
  }
  delay(1);

}

void rotateServo(Servo servo, int angle) {
  angle=constrain(angle,LIMIT_LOWER_ROTATION,LIMIT_UPPER_ROTATION);
  servo.write(angle);
}

void inclineServo(Servo servo, int angle) {
  angle=constrain(angle,LIMIT_LOWER_INCLINATION,LIMIT_UPPER_INCLINATION);
  servo.write(angle);

}
```

Una vez finalizado el programa para la placa, ya se movían los motores en función del giro de la diadema, por tanto, el trabajo estaba aparentemente finalizado a falta de intentar ajustar un poco más el movimiento de los servomotores en función de los datos. El problema surgió con esto, la diadema ofrecía unos datos muy variables y difíciles de controlar, lo que provocaba que los servomotores no se movieran correctamente y no fuera un sistema fiable. Entonces, a estas alturas del trabajo, que estaba prácticamente finalizado, nos topamos con este problema al cual no le encontraba solución, pero, se descubrió que las gafas de realidad virtual tenían un SDK [12] disponible y permitía acceder a unos giroscopios incorporados. Se tomó la decisión de investigar por ese camino a ver si se podrían utilizar en sustitución de la diadema Emotiv.

El resultado de investigar el SDK fue que, las funciones que accedían a los giroscopios, devolvían un cuaternión de rotación, algo que era bastante

bueno porque se podría transformar para obtener los ángulos de rotación directamente en lugar de trabajar con la aceleración que devolvía Emotiv. Visto esto, se dejó de lado la diadema y sus programas y se empezó a trabajar con las gafas de realidad virtual.

Se desarrolló una clase en C# que, primeramente, se conectaba a Arduino y a las gafas para ver cómo eran los datos que devolvía. Estos datos eran un cuaternión, como ya se había dicho, pero las gafas tenían una imprecisión ya que éstos, con las gafas sin movimiento, siempre disminuían a una velocidad constante. Esta imprecisión no supuso un problema puesto que se me ocurrió una solución para contrarrestarla una vez el cuaternión estuviera convertido a ángulos de Euler.

Para realizar la conversión, después de realizar búsquedas por la red para aprender sobre cuaterniones, se crearon tres funciones:

1. Una que construye un cuaternión a partir del devuelto por las gafas.

```
private void quaternion(float w, float x, float y, float z) {
        q[0] = w;
        q[1] = x;
        q[2] = y;
        q[3] = z;

}
```

2. Una que se encarga de realizar la conversión del cuaternión a ángulos de Euler. Para almacenar los ángulos de Euler, se creó una clase “EulerAnglesStruct” que únicamente tiene tres atributos, uno para cada eje, es decir, X, Y y Z.

```
private EulerAnglesStruct getEulerAngles() {
        EulerAnglesStruct eulerAnglesStruct = new EulerAnglesStruct();

        eulerAnglesStruct.x = radiansToDegrees (Math.Atan2(2.0d * (q[2] * q[3] -
        q[0] * q[1]), 2.0d * q[0] * q[0] - 1.0d + 2.0d * q[3] * q[3]));

        eulerAnglesStruct.y = radiansToDegrees (-Math.Atan ((2.0d * (q[1] * q[3]
        + q[0] * q[2])) / Math.Sqrt (1.0d - Math.Pow ((2.0d * q[1] * q[3] + 2.0d
        * q[0] * q[2]), 2.0d))));

        eulerAnglesStruct.z = radiansToDegrees (Math.Atan2 (2.0d * (q[1] * q[2]
        - q[0] * q[3]), 2.0d * q[0] * q[0] - 1.0d + 2.0d * q[1] * q[1]));

        return eulerAnglesStruct;
}
```

3. Una que es llamada desde la anterior para convertir de radianes a grados.

```
private double radiansToDegrees(double radians) {
        return 57.2957795130823f * radians;

}
```

Una vez obtenidos los ángulos sobre los ejes X, Y y Z (el eje Z no se utiliza puesto que el motor únicamente se mueve en X e Y) la solución consiste en

almacenar el valor del ángulo anterior y el valor del actual y realizar dos comparaciones:

1. Si el valor actual es mayor que el anterior significa que el usuario está mirando hacia la derecha (en el caso del eje X) y hacia arriba (en el caso del eje Y). Por el contrario, si el valor actual es menor que el anterior, significa que el usuario mira hacia la izquierda (en el caso del eje X) y hacia abajo (en el caso del eje Y).
2. Si el valor absoluto de la resta de ambos ángulos (tanto para el eje X como Y) es mayor que cierto valor, significa que el movimiento lo ha realizado el usuario y, por tanto, se soluciona la imprecisión de las gafas. Este valor se puede modificar para darle mayor o menor sensibilidad al movimiento del usuario, es decir, que detecte más o que detecte menos los pequeños movimientos.

Continuando con el resto del programa,

```
while (true) {
      driver.update ();
      rotation = driver.getFloatInformation (selectedDevice, 4,
      OpenHMD.ohmd_float_value.OHMD_ROTATION_QUAT);
      quaternion (rotation[0], rotation[1], rotation[2], rotation[3]);
      euler = getEulerAngles ();

      currX = euler.x;
      currY = euler.y;

      if (currX > prevX && Math.Abs (prevX - currX) >= 0.09d) {
                angleRotation += 0.6;
                ArduinoPort.Write ("r" + sendAngle (angleRotation));
      }
      if (currX < prevX && Math.Abs (prevX - currX) >= 0.13d) {
                angleRotation -= 0.6;
                ArduinoPort.Write ("r" + sendAngle (angleRotation));
      }
      if (currY > prevY && Math.Abs (prevY - currY) >= 0.08d) {
                angleInclination += 0.6;
                ArduinoPort.Write ("i" + sendAngle (angleInclination));
      }
      if (currY < prevY && Math.Abs (prevY - currY) >= 0.09d) {
                angleInclination -= 0.6;
                ArduinoPort.Write ("i" + sendAngle (angleInclination));
      }

      prevX = currX;
      prevY = currY;
      System.Threading.Thread.Sleep (10);

}
```

consta de un bucle que, en cada iteración, obtiene el cuaternión “rotation” de las gafas, realiza la conversión “getEulerAngles” y comparaciones descritas anteriormente “if” y, por último, en función de la condición en la que entra, realiza la operación de incrementar o disminuir el ángulo para luego realizar el envío a la placa de Arduino. En esta operación de incremento es donde se puede modificar la cantidad a añadir o disminuir para modificar la velocidad

de giro de los servomotores en función de la necesidad del usuario que lo vaya a utilizar.

Como la placa lo que recibe son cadenas de texto, se creó una función “sendAngle” que transforma los ángulos de tipo “double” en enteros y, posteriormente, en cadena de texto. También, para facilitar el proceso y hacer la lectura de los ángulos más rápida por parte de la placa Arduino, se pensó que los ángulos como máximo van a tener 3 dígitos (100 en adelante), entonces si tienen menos de 3 (100 hacia atrás) añadir a la cadena de texto 0 en las primeras posiciones hasta alcanzar la longitud de 3 caracteres. Así, se puede indicar en el programa de la placa que tiene que leer ciertos bytes cada vez y no pierde tiempo de procesamiento en comprobar cuando ha finalizado la cadena de texto recibida. Finalmente, el ángulo quedaría codificado de la siguiente manera:

- Si es un ángulo de giro en horizontal se envía “r003”. La “r” para indicar que el ángulo es de rotación y “003” para indicar que tiene que rotar hasta alcanzar la posición correspondiente a 3º del servomotor.
- Si es un ángulo de inclinación se envía “i120”. La “i” para indicar que el ángulo es de inclinación y “120” para indicar el ángulo.

```
private string sendAngle(double angle) {
        string returnAngle = ((int)angle).ToString ();
        while (returnAngle.Length < 3) {
            returnAngle = "0" + returnAngle;
        }
        return returnAngle;

}
```

Con esto realizado, solo quedaría modificar el bucle del programa de Arduino, ya que estaba preparado para el programa de Emotiv y no para éste.

```
void loop(){
  char command[4];

  if(Serial.available()){
    Serial.readBytes(command, 4);
    setAngle = (String(command[1])+String(command[2]) +
String(command[3])).toInt();

      switch(command[0]) {
      case 'r': {
        if (setAngle <= LIMIT_UPPER_ROTATION && setAngle >=
LIMIT_LOWER_ROTATION) {
          rotateServo(servoRotation, setAngle);
        }
      } break;
      case 'i': {
        if (setAngle <= LIMIT_UPPER_INCLINATION && setAngle >=
LIMIT_LOWER_INCLINATION) {
          inclineServo(servoInclination, setAngle);
        }
      } break;
    }
  }
```

```
  delay(1);
}
```

El bucle, en cada iteración, comprueba si el puerto serie está disponible. Si es así, realiza una lectura de 4 bytes y lo almacena en un array de caracteres de tamaño 4 para luego unir los caracteres de la posición 1 a la 3 y convertirla a entero con el fin de obtener el ángulo a indicar al servomotor correspondiente. Luego, obtiene el carácter de la posición 0 y lo clasifica para saber si el ángulo leído es de rotación o de inclinación. Una vez clasificado, comprueba que el ángulo recibido está dentro de los rangos físicos de los servomotores y, de ser así, realiza el movimiento del servomotor.

Una vez finalizado esto, y de realizar unos ajustes en la sensibilidad de la detección del movimiento y de la velocidad de giro de los servomotores, el trabajo se dio por funcional y finalizado con lo que pasaríamos a ver los resultados.

Capítulo 4. Resultados del proyecto

La imagen a continuación corresponde a los diferentes elementos del proyecto montados en la silla autónoma del laboratorio de robótica.

Figura 18: Componentes montados en la silla de ruedas

Para poner el proyecto en funcionamiento se requiere realizar los siguientes pasos:

1. Conectar el par estéreo al ordenador y realizar la calibración en caso de que haya recibido movimientos/golpes externos a los servomotores.
2. Copiar el archivo “rmaps” generado por el proceso de calibración dentro de la carpeta “FilesVS3D” que está localizada junto a la aplicación de Unity3D.
3. Conectar las gafas de realidad virtual al ordenador y ejecutar la aplicación de Unity3D.
4. Conectar la placa de Arduino al ordenador, comprobar el puerto “COMx” asignado a la placa y, en caso de ser diferente al establecido en el programa, cambiarlo por el asignado a la placa para que coincidan. Posteriormente, ejecutar el programa que realiza la obtención de los datos de los giroscopios y el movimiento de los servomotores.

En caso de que ya se haya ejecutado anteriormente y el par estéreo no haya recibido movimientos/golpes, únicamente se realizarían los pasos 3 y 4.

Capítulo 5.
Conclusiones y líneas futuras

Viendo los resultados obtenidos en este trabajo, es posible concluir que esta herramienta tiene bastante potencial para ayudar y facilitar a las personas con diversidad funcional la experiencia de estar en una silla de ruedas, dándoles la capacidad de ser más conscientes de su entorno.

Por otro lado, el manejo de esta herramienta, una vez instalada en la silla de ruedas, es bastante sencillo e intuitivo, teniendo únicamente que mover la cabeza de forma natural y la herramienta haría el resto del trabajo, lo que le da otro punto a favor para su utilización.

Para finalizar, como trabajo futuro que podría realizarse a partir de este trabajo tenemos:

- La incorporación del manejo de los motores a través de la lectura de las ondas cerebrales. Esto daría aún más libertad para su utilización, puesto que, no haría falta tener que mover la cabeza y podría ser utilizada en casos muy graves de discapacidad.
- La mejora de la interfaz en la realidad virtual. Se podrían añadir diferentes indicadores para alertar al usuario o hacer la experiencia más agradable. Algunos ejemplos pueden ser, entre otros, la detección de bordillos, el reconocimiento de personas y la velocidad actual de la silla.
- Una vez incorporado la lectura de ondas cerebrales, también se podrían añadir diferentes comandos para interactuar con el sistema, como, por ejemplo, establecer una velocidad máxima a la silla, bloquear su movimiento, etc.

Capítulo 6.
Conclusions and future work

Seeing the results obtained in this work, it’s possible to conclude that this tool has enough potential to help and make it easier to people with functional diversity the experience of being in a wheelchair, giving them the ability to be more aware of their environment.

On the other hand, the handling of this, once installed in the wheelchair, is quite simple and intuitive, having only to move the head naturally and the tool would do the rest of the work, which gives another point in favor if its use.

Finally, as future work that could be done from this work we have:

- The incorporation of the handling of the motors through the reading of the brain waves. This would give even more freedom to use, since it wouldn't require having to move the head and could be used in very severe cases of disability.
- Improve interface in virtual reality. Different indicators could be added to alert the user or make the experience more enjoyable. Some examples can be, among others, the detection of curbs, the recognition of people and the current speed of the wheelchair.
- Once the brain wave reading has been incorporated, different commands can also be added to interact with the system, such as setting a maximum speed to the wheelchair, blocking their movement, etc.

Capítulo 7. Presupuesto

En el presupuesto de este trabajo se incluyen los diferentes costes de los componentes a utilizar, así como las horas de desarrollo necesarias para llevarlo a cabo. El presupuesto para el software a utilizar no es necesario puesto que es completamente gratuito.

Componentes	Presupuesto
Gafas de realidad virtual	250 €
Placa Arduino	30 €
Cámara Logitech x2	60 €
Estructura del par estéreo	20 €
Servomotor x2	100 €
TOTAL	**460 €**

Tabla 1: Presupuesto de los componentes del proyecto

Tarea	Horas de trabajo	Presupuesto
Montaje del par estéreo	2 h (15€/h)	30 €
Montaje de los servomotores con la placa Arduino	2 h (15€/h)	30 €
Desarrollo de los programas e investigación de las herramientas	450 h (20€/h)	9.000 €
TOTAL	**454 h**	**9.060 €**

Tabla 2: Presupuesto de las horas de trabajo en el proyecto

Después del desglose anterior, el presupuesto total necesario para este proyecto sería de 9.520€ aproximadamente, ya que las horas necesarias para su desarrollo no es un dato fiable y pueden variar.

Bibliografía

[1] http://rvsn.csail.mit.edu/wheelchair/

[2] http://www.adaptado.es/ford-echair/

[3] http://www.conacytprensa.mx/index.php/tecnologia/robotica/11939-mexicanos-desarrollan-en-alemania-silla-de-ruedas-autonoma

[4] https://github.com/Emotiv/community-sdk

[5] https://unity3d.com/es

[6] https://developers.google.com/vr/unity/

[7] http://opencv.org/

[8] http://diskokosmiko.mx/ZEROGIGA/unity-5-13468/opencv-for-unity-v1-2-3,69077.unitypackage

[9] https://github.com/shimat/opencvsharp

[10] https://es.wikipedia.org/wiki/YAML

[11] https://programarfacil.com/tutoriales/fragmentos/servomotor-con-arduino/

[12] http://www.openhmd.net/