Memoria Práctica 1

Sistemas de Percepción

Borja Peirotén y Juan Manuel Colmena

Tabla de contenido

1. Introducción

El objetivo de esta práctica es implementar una aplicación de realidad aumentada empleando las cámaras de un ordenador y un móvil aplicando los conocimientos obtenidos en clase. La realidad aumentada es la introducción de elementos visuales digitales en el entorno obtenido por la cámara, adaptándose correctamente dependiendo de la perspectiva, dando la sensación de que pertenece al entorno real.

Es necesario conocer las distintas tecnologías y algoritmos que se han empleado en los últimos años para entender el enfoque escogido para la solución y sus características respecto a otras soluciones.

En la práctica se empleará un patrón de calibración y se obtendrán los parámetros intrínsecos de la cámara, de forma que se conoce como se proyectan los puntos 3D en la imagen, es decir, calibrar la cámara. Una vez se conocen los parámetros intrínsecos para cada imagen se procede a calcular los parámetros extrínsecos a partir de los puntos conocidos del patrón de calibración, donde se encuentra el origen de coordenadas y se mostrarán los nombres Borja y Juan encima de este patrón. La solución desarrollada se explicará de una forma más extensa posteriormente.

Por otro lado, se procederá a realizar una experimentación que tratará de mostrar cómo afectan distintos parámetros en la precisión en la proyección de los puntos y el tiempo de procesamiento en la fase de calibración y en la fase de proyección del elemento digital.

2. Estado del arte

El campo de la realidad aumentada ha crecido mucho en las últimas décadas, sobre todo en el sector del entretenimiento en los últimos años. Todo su desarrollo ha sido acompañado por grandes avances en la tecnología como capacidad de procesamiento de imágenes, seguimiento de objetos tanto que en la actualidad existen muchos algoritmos, herramientas y soluciones de calidad para diferentes sistemas, entre los cuales se encuentran los siguientes cuya finalidad está relacionada con la realidad aumentada.

- **SLAM:** Es una técnica utilizada en robótica móvil para localizarse en un entorno desconocido a la vez que mapea el entorno. Se puede decir que al igual que en la realidad aumentada, es necesario obtener medidas del entorno y con dichos valores y la odometría del robot se localiza el robot en dicho entorno y se crea el mapa ya que se conocen los puntos detectados. Por lo tanto, se trata de un problema más sencillo en ciertos aspectos, ya que los sensores normalmente son sensores Lidar que se conoce la localización 3D de los puntos respecto al origen de coordenadas de la cámara por lo que se trata más el aspecto de localizarse y conocer la transformación que relaciona la posición inicial con la actual. El algoritmo más empleado y robusto es el EKF o filtro extendido de Kalman, el cual, tiene en cuenta los errores de los sensores y de la odometría desde el punto de vista probabilístico.

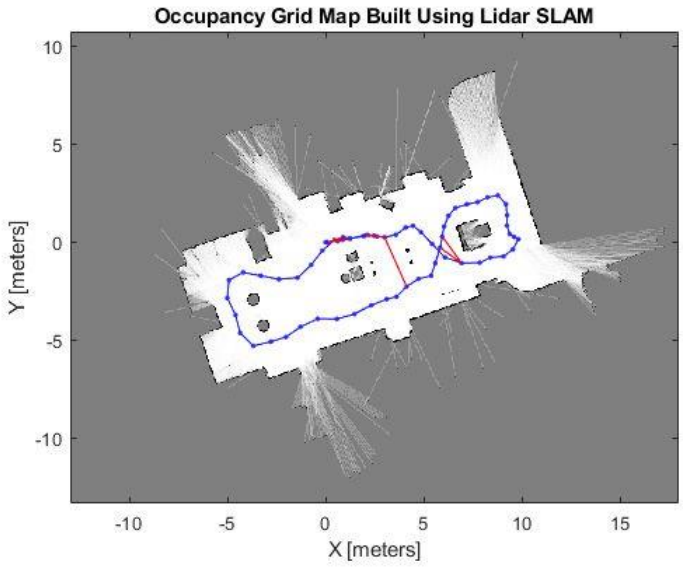

Figura 1: Ejemplo de SLAM

- **Soluciones basadas en Deep Learning:** La inteligencia artificial ha mejorado en mucho las aplicaciones relacionadas con la identificación y seguimiento de objetos, sobre todo las redes neuronales convolucionales y recurrentes. Esta herramienta puede obtener soluciones para obtener imágenes de profundidad y dependiendo de la perspectiva desde la que se observa un objeto predecir la transformación con El Deep Learning es un método de Machine Learning que busca imitar el funcionamiento de las neuronas humanas, entrenándose mediante

experiencia. La utilización del Deep Learning mejora en un gran porcentaje el reconocimiento y detección de distintos escenarios u objetos y facilita enormemente su seguimiento por lo que es una gran herramienta para reconocer objetos en tiempo real.

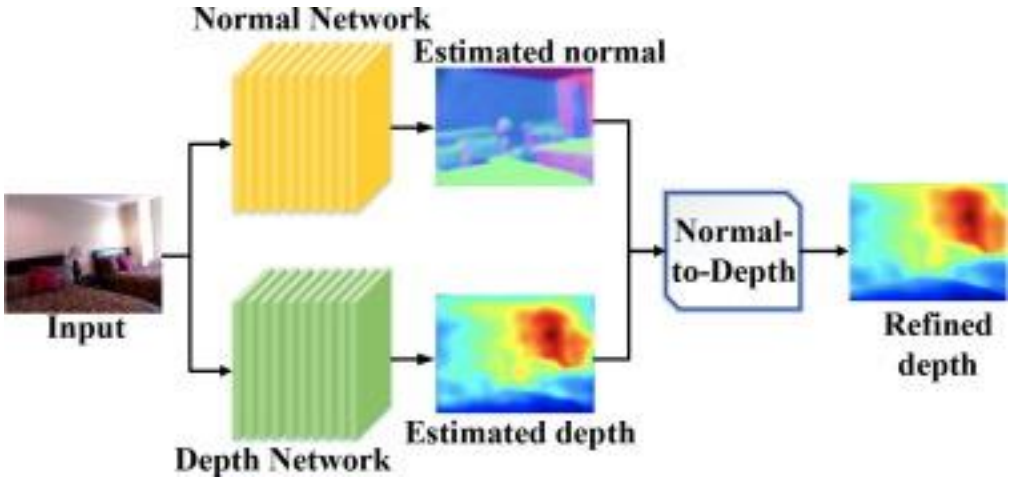

Figura 2: Ejemplo de Deep Learning

- **Realidad Virtual y Realidad Mixta:** En el caso de la realidad virtual, en la mayoría de los casos las gafas necesitan de dos sensores colocados en dos extremos de la habitación para poder localizarse, que puede mezclar la información proveniente de varios sensores mediante algoritmos de Sensor Fusion para poder reducir la incertidumbre de la localización. Además, este campo ha dado pie a la creación de realidad mixta, la cual se encuentra entre la realidad virtual y aumentada y trata de mostrar dichos elementos digitales en el entorno real y poder manipularlos sin necesidad de tener un patrón sobre el cual construir. Estos algoritmos permiten obtener soluciones muy realistas en casos donde el alineamiento con la realidad es fundamental partiendo de la información de sensores inerciales, algún sensor de profundidad y puntos característicos encontrados con las cámaras RGB repartidas por las gafas.

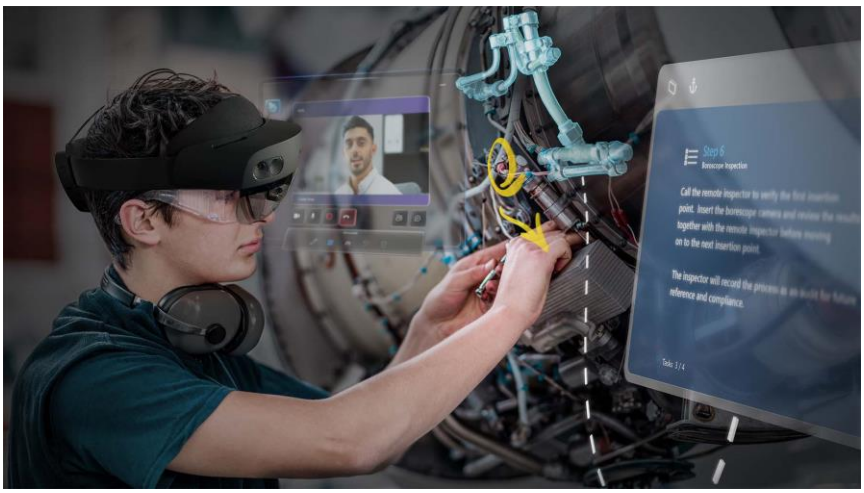

Figura 3: Ejemplo de Realidad Mixta

3. Solución desarrollada

Para esta práctica se han desarrollado dos archivos de código principales que serán los encargados de la calibración de la cámara y de la proyección de diversos puntos y dibujos en el patrón de calibración. El primer archivo llamado Video.py muestra la aplicación requerida en el enunciado mientras que el segundo archivo, Experimento_Calibración.py se calibra la cámara empleando distintos parámetros y calcula el error de reproyección medio de todas las imágenes que están contenidas en la carpeta de Ordenador, ya que la experimentación se ha llevado a cabo siempre con la misma cámara.

El procedimiento seguido por el primer archivo es calibrar la cámara inicialmente, tal y como se ha aprendido en teoría. Para determinar los parámetros intrínsecos se no variarán a lo largo de toda la ejecución se ejecuta la función calibrar. En esta función se seleccionan los nombres de los archivos de las imágenes pertenecientes a la carpeta seleccionada en las variables globales, con los que se calibrará la cámara. Una vez se tienen dichos nombres, para cada imagen, se escalará a la resolución deseada de forma que se mantenga un equilibrio entre el tiempo de ejecución y precisión. Tras esto, se buscarán las esquinas de las casillas del tablero mediante la función findChessboardCorners que después se refinan los puntos teniendo en cuenta los gradientes de las áreas cercanas a los puntos a través de la función cornerSubPix.

Cuando ya se han conseguido estos puntos y se han guardado en un vector, se conocen las correspondencias del sistema de coordenadas y la proyección de la imagen, por lo que se procede a pasar los valores a la función calibrateCamera de OpenCV para calibrar la cámara, la cual devuelve la matriz de los parámetros intrínsecos de la cámara, los coeficientes de distorsión y los vectores de rotación y traslación para cada una de las imágenes empleadas en la calibración. Finalmente, la función de calibración creada devolverá los parámetros intrínsecos de la cámara (matriz de la cámara y coeficientes de distorsión) dados por la función de calibración de OpenCV.

Con los parámetros intrínsecos conocidos, se abre el archivo de video a procesar o incluso la cámara del dispositivo para ejecutar la aplicación en tiempo real, donde se obtiene el fotograma correspondiente y se procede a ejecutar las mismas funciones que en el caso de la calibración aunque cuando se sabe la localización de las proyecciones de todos los puntos de calibración en la imagen se calculan los parámetros extrínsecos mediante la función solvePNP, el cual emplea el algoritmo de Levenberg-Marquardt para disminuir el error de reproyección.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates

Figura 4: Ecuación para proyectar puntos en un sistema de coordenadas en la imagen

Conociendo todos los parámetros que conforman la ecuación de la figura 4 es posible multiplicar cada punto que se quiere proyectar para obtener las coordenadas en la imagen que se consigue mediante la función projectPoints de OpenCV, pero debido a su

desconocimiento inicialmente se realizaron estas operaciones mediante numpy y transformando el vector de rotación de tipo Rodrigues a una matriz de rotación y creando la matriz de transformación correspondiente, consiguiendo el mismo resultado pero en un código más extenso.

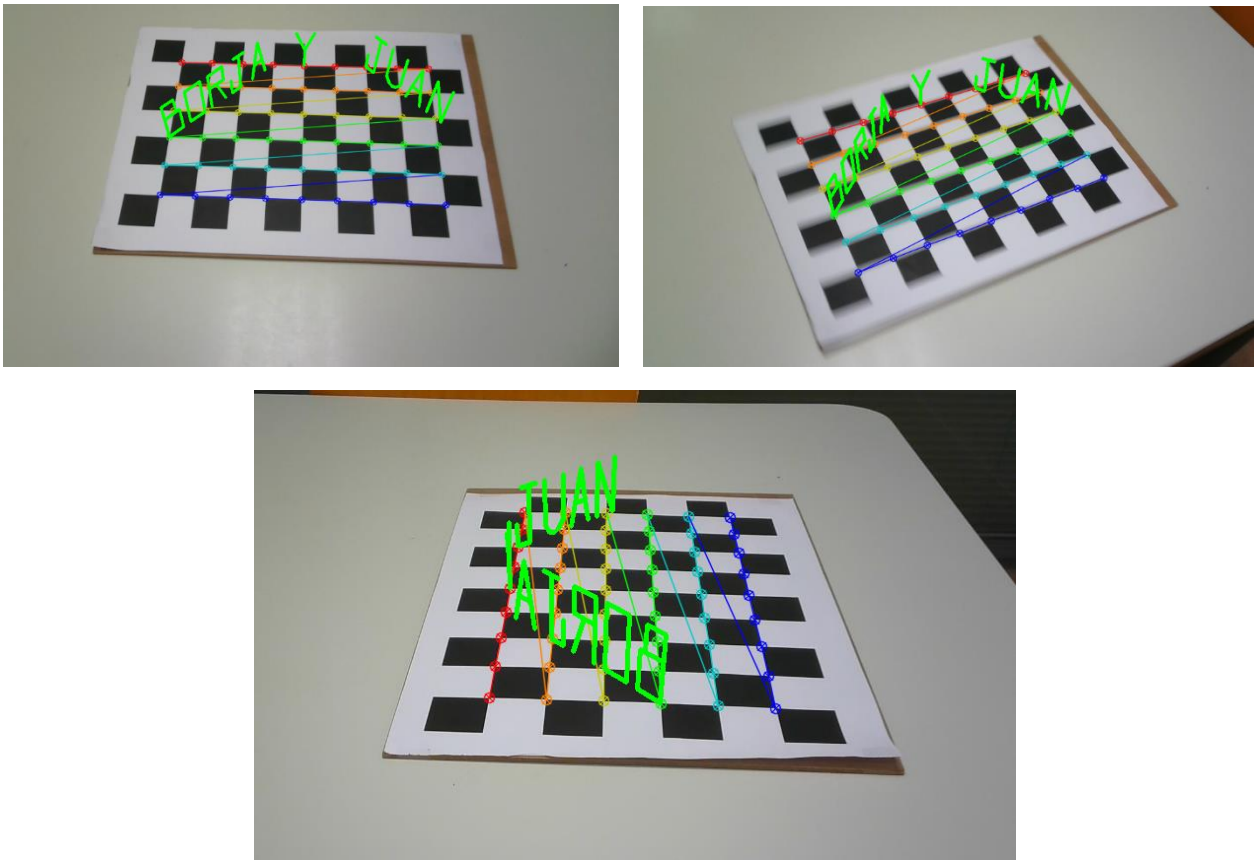

Figura 5: Resultado obtenido de la proyección de puntos

En la solución desarrollada se ha querido proyectar nuestros nombres en el tablero y para ello se ha realizado todo el procedimiento en la función DibujarNombres. Se importa al principio del código un archivo a modo de librería en el que están definidos todos los puntos de cada letra. También, antes de esta función, se habrán creado otras dos auxiliares para poder rotar cada letra un ángulo determinado y aplicar la transformación en el entorno 3D. Una vez se han obtenido los puntos 3D se proyectan mediante la función de OpenCV mencionada. Una vez se tienen las correspondencias, se unirán los puntos para formar las letras de una manera legible.

Al final se podrá observar una parte del código cuyo cometido es enviar los datos de los tiempos que tarda el programa en hacer diferentes funciones a un archivo CSV para poder calcular la media y desviaciones estándar de los tiempos de cada fase.

Media Encontrar Chessboard	Media Refinar esquinas	Media Determinar Matriz de Transformación	Media Proyectar Puntos
0.037744015	0.001269935	0.000356849	0.000448026
Desviación Típica Encontrar Chessboard	**Desviación Típica Refinar Esquinas**	**Desviación Determinar Matriz de Transformación**	**Desviación Típica Proyectar Puntos**
0.061908542	0.000460277	0.00010982	0.000140708

Tabla 1: Resultados de los datos de tiempo de las distintas fases

El proceso que sigue el segundo archivo es muy similar, donde primero de todo se calibra la cámara empleando un número de imágenes que va iterando y para todas las imágenes de la carpeta Ordenador se vuelven a buscar las esquinas de las casillas del tablero. Tras esto, se obtienen los parámetros extrínsecos de las imágenes con la función SolvePnP de OpenCV utilizando en ella el algoritmo de Levenberg-Marquardt. Teniendo los parámetros extrínsecos y los intrínsecos, se procede a proyectar los puntos de las esquinas. Cuando estos ya han sido proyectados, se procede a sacar la diferencia de los puntos proyectados y los hallados, dando así el error medio de reproyección que será lo que devuelva la función.

```
Tiempo de calibracion: 27.99510955810547
El error de calibracion con 1 imagenes, es 2.70496633569866 73
Tiempo de calibracion: 53.99918556213379
El error de calibracion con 2 imagenes, es 3.7551904528068465
Tiempo de calibracion: 74.06330108642578
El error de calibracion con 3 imagenes, es 0.5500861319141296
Tiempo de calibracion: 103.66988182067871
El error de calibracion con 4 imagenes, es 0.5008183558121003
```

Figura 6: Resultado del archivo Experimento_Calibración.py

4. Experimentación

En este apartado se procederá a analizar y explicar cómo influyen distintos parámetros que se emplean a lo largo de todo el procedimiento de la calibración como puede ser la resolución de las imágenes, el número de filas y columnas de puntos que contiene el patrón de tablero de ajedrez, o el número de imágenes empleado principalmente.

Resolución	Número de Puntos en el Patrón	Número de Imágenes	Error Medio de Reproyección	Error Medio de Reproyección Normalizado	Tiempo de Calibración (ms)
1920x1080	54	3	0,996	0,996	89,65
1920x1080	54	6	0,762	0,762	196,03
1920x1080	54	9	0,726	0,726	333,71
1920x1080	20	6	1,184	1,184	3761,15
1440 x810	54	3	0, 744	0,992	77,54
1440 x810	54	6	0, 572	0,762	169,59
1440 x810	54	9	0,545	0,726	301,75
1440 x810	20	6	0,862	1,150	1860,61
960x540	54	3	0,550	1,100	70,99
960x540	54	6	0,431	0,862	165,05
960x540	54	9	0,414	0,828	279,66
960x540	20	6	0,616	1,232	749,21

Tabla 2: Resultados de las distintas configuraciones en la calibración

Como se puede observar en la tabla, las distintas resoluciones tienen un impacto en el tiempo de calibración, ya que como las imágenes contienen más pixeles tardan más en buscar los puntos correspondientes al patrón de calibración, aunque inicialmente se puede pensar que se puede tener una mayor precisión. Sin embargo, aunque el error medio de reproyección aumente en el caso de aplicar una resolución mayor, hay que pensar que este mide la distancia euclídea de la posición de los pixeles por lo que es proporcional a la escala. En la siguiente columna donde los resultados están normalizados, es posible observar que el error de reproyección es mayor en el caso de una resolución menor, puesto que, al trabajar la misma imagen con menos pixeles, se pierde precisión.

En cuanto al número de puntos que contiene el patrón un mayor número de puntos ayuda a obtener un mejor resultado de los parámetros intrínsecos de la cámara, ya que de cada imagen se puede extraer más información y dependiendo de la distancia a la que se encuentre el tablero, se ocupa una mayor área. Igualmente, se puede observar que la calibración realizada con las imágenes del patrón pequeño tarda mucho más en ejecutarse y esto es debido a que para el algoritmo que busca las esquinas del patrón es más complejo encontrar un patrón pequeño que uno grande.

El número de imágenes empleado tiene un impacto significativo en el resultado puesto que inicialmente cuando se tratan de pocas imágenes, los parámetros intrínsecos obtenidos evolucionan de una forma más rápida hacia un resultado con un menor error de reproyección, pero cuando se trata de añadir una imagen más cuando en el conjunto de calibración ya se tienen veinte, la diferencia es mínima, pero ralentiza la calibración. Esto se puede observar en la figura 7.

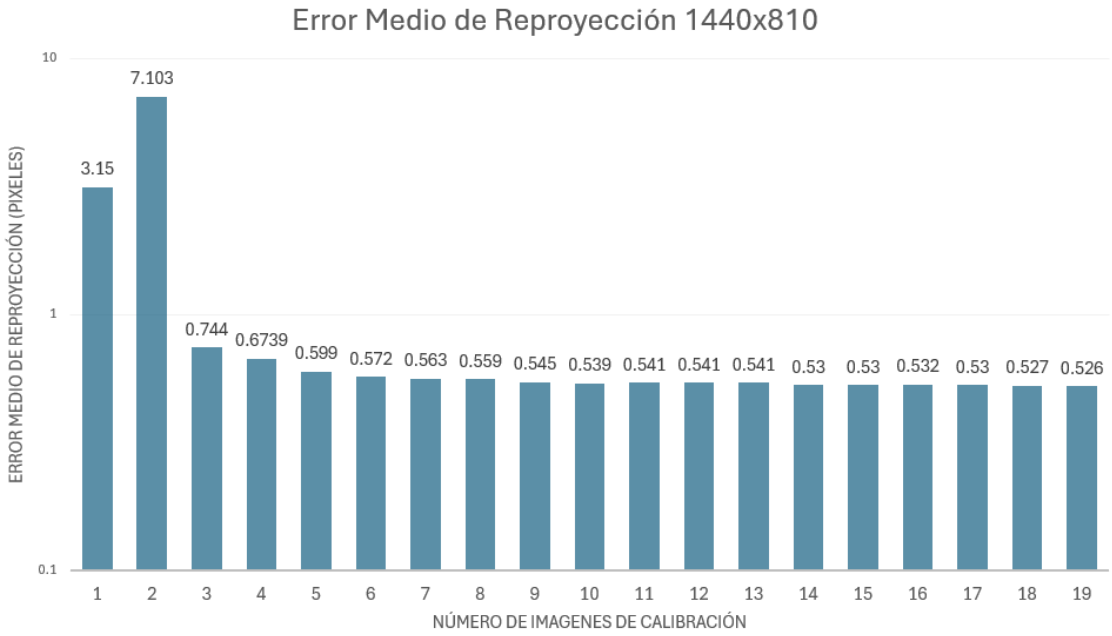

Figura 7: Error medio de reproyección

Por otro lado, se ha probado a variar el tamaño de la ventana de la función cornerSubPix de OpenCV junto con otros parámetros como el número de iteraciones máximo del algoritmo o la precisión buscada para finalizar antes el procesamiento en caso de conseguirlo. Esta función es empleada para refinar los puntos obtenidos inicialmente en la función findChessboardCorners, aportando decimales valiosos para la calibración. Este experimento se realizará para el patrón de seis filas por nueve columnas empleando seis imágenes para la calibración.

Resolución	Tamaño de la Ventana	Error Medio de Reproyección	Error Medio de Reproyección Normalizado	Tiempo de Calibración (ms)
1920x1080	21x21	0,7751	0,7751	230,10
1920x1080	15x15	0,7638	0,7638	196,07
1920x1080	11x11	0,7622	0,7622	200,10
1920x1080	7x7	0,7784	0,7784	190,09
960x540	21x21	2,2090	4,4180	227,03
960x540	15x15	0,7502	1,5004	204,02
960x540	11x11	0,4306	0,8612	193,74
960x540	7x7	0,3821	0,7642	161,83

Tabla 3: Resultados de la variación del tamaño de la ventana de cornerSubPix

Como se puede observar en la tabla 3, el hecho de modificar la ventana puede ser beneficioso, pero se debe tener en cuenta la resolución empleada, ya que, para una imagen mayor, más pixeles representaran el tablero y pudiendo necesitar un tamaño mayor de ventana, es decir, la ventana debe mantener una proporcionalidad con la resolución.

Resolución	Iteraciones Máximas	Precisión Máxima	Error Medio de Reproyección	Error Medio de Reproyección Normalizado	Tiempo de Calibración (ms)
1920x1080	5	0,005	0,7621	0,7621	193,13
1920x1080	5	0,0005	0,7621	0,7621	191,61
1920x1080	40	0,005	0,7622	0,7622	194,31
1920x1080	40	0,0005	0,7622	0,7622	194,17
960x540	5	0,005	0,4215	0,8430	188,06
960x540	5	0,0005	0,4214	0,8428	186,06
960x540	40	0,005	0,4306	0,8612	170,71
960x540	40	0,0005	0,4306	0,8612	183,13

Tabla 4: Resultados de la variación de parámetros de cornerSubPix con una ventana de 11x11

La modificación del resto de parámetros como la precisión o las iteraciones máximas para indicar la finalización del algoritmo no tiene ningún efecto destacable, aunque en el caso de la resolución baja, el empleo de menos iteraciones ha conseguido un error de reproyección menor, lo que podría ser porque nunca se alcanza la precisión buscada y tras finalizar las iteraciones se encuentra en un estado algo peor. En el caso de emplear cámaras de mayor calidad podría ser que tuviera cierto impacto en el caso de imágenes con una resolución muy alta.

5. Conclusiones

Para finalizar la memoria se podría decir que los parámetros más importantes a la hora de calibrar una cámara correctamente es el uso de unas seis o diez imágenes, las cuales contengan perspectivas diferentes del tablero y que se encuentre en distintos lugares de la imagen para poder definir de forma óptima la cámara, empleando la memoria y tiempo de procesamiento justo sin sacrificar precisión.

En cuanto a la velocidad a la que se procesa cada fotograma de los videos, no se ha conseguido una ejecución fluida puesto que en casos donde determinar las esquinas del patrón es algo complicado porque aparece borroso produce un mayor tiempo de ejecución del algoritmo de OpenCV, el cual puede tardar hasta 0,2 segundos, ralentizando mucho el proceso en estos casos. Este tiempo representa el 94% de todo el proceso de determinación de los parámetros extrínsecos y proyección de los puntos.

El repositorio de GitHub en el que se ha desarrollado la práctica se encuentra en el siguiente enlace, y para su ejecución únicamente es necesario ejecutar el archivo Video.py.

https://github.com/Borja418/P1_Percepcion