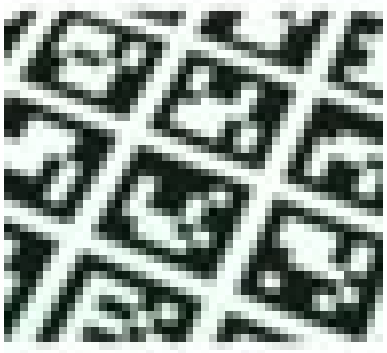
Sistemas Multimedia Interactivos e Inmersivos

Práctica de introducción a la Realidad Aumentada con ArUco y OpenCV



Manuel Agustí
Curso 2k20/2k21
Grado de Ingeniero en Informática
Escola Tècnica Superior d'Enginyeria Informàtica
DISCA - UPV

En la presente práctica, se aborda el uso de ArUco, un módulo de OpenCV para realizar la detección y seguimiento de marcas binarias que utilizaremos como base para explorar los mecanismos de desarrollo de aplicaciones de Realidad Aumentada (RA).

En este curso vamos a utilizar los archivos de código de ejemplo que vienen con la instalación de ArUco como módulo de OpenCV. Habrá que escoger entre dos versiones con pequeñas diferencias entre ambas debidas a la versión de OpenCV sobre la que están construidas (*Detection of ArUco Markers* usando OpenCV 3.2.0 [9] o *Detection of ArUco Markers* usando OpenCV 4.2.0 [1]) y por tanto en función de la tenga disponible en su equipo de trabajo:

- Con OpenCV 3.2, la que tenemos disponible en la máquina virtual (construida a partir de la imagen de la instalación del laboratorio) que utiliza la distribución de *Linux Ubuntu 18.04*. En este caso deberá utilizar los ejemplos de código contenidos en el PoliformaT de SMII en "Recursos/Prácticas/PL04 Introdución a la RA con ArUco y OpenCV/ examples_aruco_v3.1__OpenCV_v3.2.zip".

- Con OpenCV 4.2, la que tenemos disponible en las instalaciones más recientes como la que viene en los repositorios de *Linux Ubuntu 20.04* y que se puede obtener del Github de OpenCV[1] o utilizar los ejemplos de código contenidos en el PoliformaT de SMII en "Recursos/Prácticas/PL04 Introdución a la RA con ArUco y OpenCV/ examples_aruco_v3.4__OpenCV_v4.X.zip".

De los ejemplos de código se obtendrá la experiencia en el uso de ArUco para RA a través de comprobar los resultados obtenidos como respuesta a los ejercicios propuestos. Recuerde que hay una actividad que ha de guardar su resultado para añadirlo al porfolio.

1 Introducción

Las aplicaciones de Realidad Aumentada (en adelante RA) [1] ofrecen un ejemplo de sistema interactivo e inmersivo, en tanto que:

- La aplicación responde a eventos que suceden en tiempo de ejecución por el análisis visual (en la mayor parte de las aplicaciones desarrolladas) de la escena del mundo real al que tiene acceso la cámara del equipo que está ejecutando la aplicación al encontrar en la escena una imagen predefinida.
- La respuesta del sistema ofrece al usuario una visión del mundo real a la que se añaden objetos sintéticos con comportamientos estáticos o dinámicos en función de la aplicación, que le confieren la característica de inmersión.

Ejemplos de aplicaciones finales de estas técnicas[2] de uso de la RA se pueden ver en la Figura 1 y se pueden encontrar otros en una búsqueda rápida por la Web[3]. En ellos podemos ver aplicaciones que, como la mayor parte de trabajos de RA utilizan la imagen como medio de acceso al mundo real.

1 El Github de OpenCV <https://github.com/opencv/opencv_contrib/tree/master/modules/aruco/samples >.

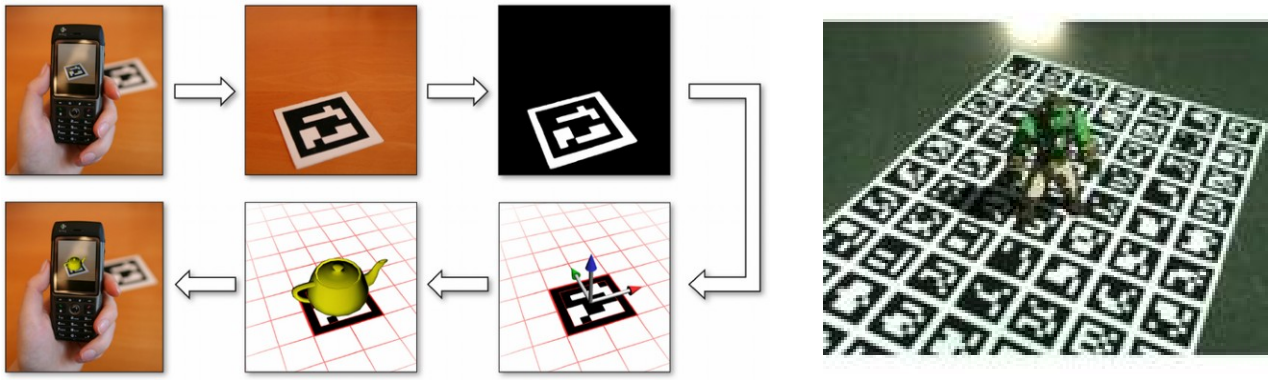
2 Como se puede ver en [2] y en Juegos RA: Realidad Aumentada <https://www.nintendo.es/Familia-Nintendo-3DS/Software-instantaneo/Juegos-RA-la-realidad-aumentada/Juegos-RA-la-realidad-aumentada-115169.html >.

3 Por ejemplo, en "How augmented and virtual reality are enhancing customer service and satisfaction for many top brands" <https://www.cxnetwork.com/cx-experience/articles/how-augmented-and-virtual-reality-are-enhancing-customer-service-and-satisfaction-for-many-top-brands> o How to Use AR (Augmented Reality) to Improve the Customer Experience <https://blog.hubspot.com/service/augmented-reality-customer-experience>.

a) b)

Figura 1: Ejemplos de uso de aplicaciones de RA. Imágenes de Yelp (2009) [2] y de juegos para Nintendo 3DS (2011).

OpenCV es [3] la herramienta que hemos venido utilizando para analizar las imágenes y desde la versión 3.1 incorpora el módulo ArUco[4] [4] que está especializado en la detección y seguimiento de marcas binarias, así como para la obtención de la posición relativa de la cámara y la marca. Estas operaciones son básicas para la realización de aplicaciones de RA. Vamos a explorar el API de este módulo de OpenCV, véase Figura 2, para tener una experiencia en la realización de aplicaciones de RA de carácter multiplafaforma y con la posibilidad de conectarlo a cualquier API[5] de visualización que queramos emplear. El Anexo I. Otras fuentes muestra un ejemplo de uso sobre plataformas móviles.



a) b)

Figura 2: Ejemplos básicos de RA con OpenCV.

1.1 Materiales

ArUco utiliza marcas binarias[6] para hacerlas servir de referencia y, si se le pide, también para el cálculo de la posiciones relativas de la cámara y la marca. Un ejemplo de las diferentes variantes que es capaz de detectar ArUco las podemos ver en la Figura 3. Veamos cómo generarlas y qué

4 Las referencias de investigación relativas al trabajo de ArUco las podéis encontrar en [3] y [4]. También podéis encontrar las versiones estables en *SourceForge* [6].

5 Al respecto del API de ArUco se puede encontrar documentación en el sitio web de OpenCV <https://docs.opencv.org/3.2.0/d9/d6a/group__aruco.html> y explicaciones de uso de las operaciones que se definen en ese API en [9]. y en "ArUco: An efficient library for detection of planar markers and camera pose estimation" <https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#>.

6 Conocidas como "fiduciales". este es un nombre genérico que se utiliza en diferentes contextos de análisis de imagen que se utiliza para designar un objeto que se utiliza como marca de referencia en la imagen. Véase "Marcador de referencia" <https://es.wikipedia.org/wiki/Marcador_de_referencia>.

operaciones nos ofrece el API de ArUco para trabajar con ellas.

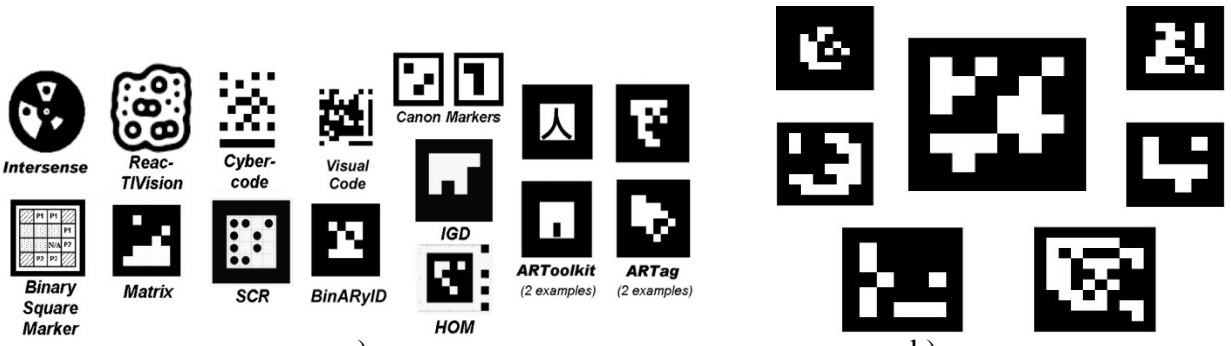


Figura 3: Ejemplos de marcadores. Imágenes extraídas de <https://medium.com/deemaze-software/augmented-reality-a-simple-technical-introduction-83d5e77206b9> y de [9]

A la hora de usar marcas, cada aplicación ha diseñado las suyas con unos contenidos (unas formas) que considera más rápidas para el proceso de detección, buscando tanto la eficiencia como la robustez en este proceso. Por ello, algunas soluciones se basan en el uso de marcas aisladas, esto es proponer una marca a buscar en la escena y sobre su aparición o no, generar los eventos que precisa una aplicación para dirigir su comportamiento. En otros casos, se han definido marcas compuestas, formadas por varias marcas aisladas y que se disponen en un mismo plano (Figura 4a), siendo el dibujo de un tablero de ajedrez (*chessboard*) el utilizado en las primeras soluciones de este tipo y qu e ha ido evolucionando en otros más complejos como el tablero ArUco o el ChArUco propios del API que estamos explorando.

Otros sistema de marcas compuestas que se disponen en el espacio 3D (Figura 4b) también se utilizan para aplicaciones de guiado y elaboración de mapas para sistemas autónomos.

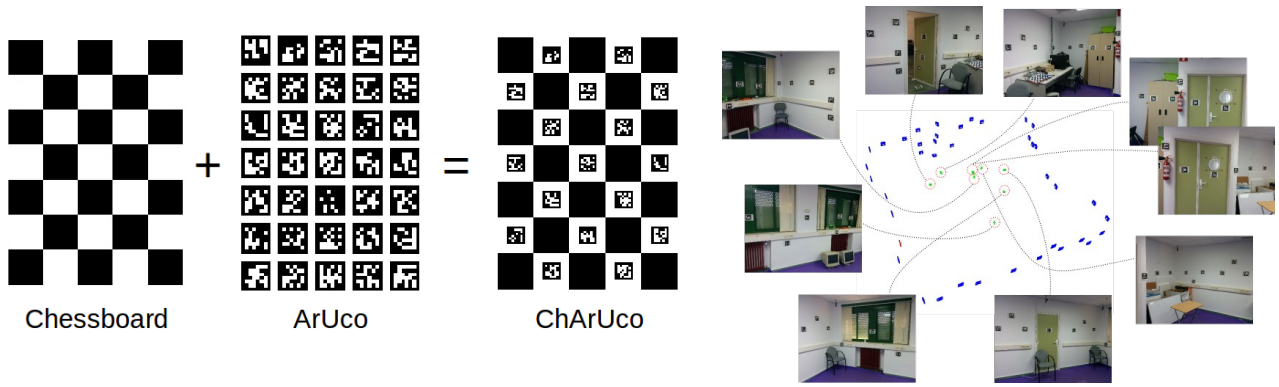


Figura 4: Ejemplos de sistemas compuestos de marcas 2D (a) y 3D (b). Imágenes extraída de [1] y Mapping and Localization from Planar Markers <http://www.uco.es/investiga/grupos/ava/node/57>.

En el laboratorio y en la máquina virtual, está instalado el módulo de ArUco para la versión de OpenCV que hay disponible, vamos a crear un subdirectorio para ir realizando sobre él los apartados de esta práctica. Yo lo he llamado "examples_aruco", sobre él copiaremos los ejemplos para poder tener una versión de los ejemplos que podamos compilar y modificar.

El Listado 1 muestra la secuencia de órdenes para generar las versiones ejecutables de los ejemplos de ArUco disponibles en la versión de OpenCV 3.2 que tenemos en el laboratorio y la máquina virtual., así como la que puede tener en su equipo si ha instalado OpenCV 4.2.

```
$ mkdir examples_aruco
$ cp -rp /usr/share/doc/opencv-doc/examples/aruco/* examples_aruco
$ cd examples_aruco
$ for i in *.gz ; do gzip -d $i; done
$ for i in *.cpp; do g++ $i -o `basename $i .cpp` `pkg-config opencv --cflags --libs`; done
```

Listado 1: Secuencia de órdenes para generar los ejemplos de ArUco en el laboratorio con OpenCV 3.2.

```
$ mkdir examples_aruco
$ cp -rp /usr/share/doc/opencv-doc/examples/aruco/* examples_aruco
$ cd examples_aruco
$ for i in *.cpp; do g++ $i -o `basename $i .cpp` `pkg-config opencv4 --cflags --libs`; done
```

Listado 2: Secuencia de órdenes para generar los ejemplos de ArUco en el laboratorio con OpenCV 4.2.

Ahora, que ya los tenemos en nuestro espacio de usuario, solo nos queda ponerlos en marcha y aprender con ellos, vamos allá.

2 Creación y detección de marcas aisladas

Como se ha comentado, ArUco puede trabajar con varios tipos de marcas [7], por lo que para empezar es necesario definir el conjunto de estas que se vaya a utilizar. Para ello, en la terminología de ArUco se habla de "diccionarios", que contienen la descripción de un conjunto de marcas.

Cada marca es una rejilla cuadrada de puntos a negro o blanco (las hemos visto en la Figura 3.b), de forma que el borde exterior está siempre a negro y la parte interna codifica su identificador: una secuencia de 1s y 0s, con casillas a blanco o negro respectivamente. Así se habla de marcas de tamaño 4x4 (el de la rejilla interna) que definen identificadores (también llamados *codewords*) de 16 bits.

Los diccionarios, no contienen los mapas de bits que representan a cada marca, sino su codificación interna, en las cuatro posibles rotaciones en que se puede encontrar cada marca en una imagen. Y su nombre indica el número de bits y de marcas que contiene: así, "DICT_4X4_50" es un diccionario de cincuenta entradas de códigos de 16 bits (4x4). Aunque existen algunos otros[7] nombres, como p. ej. "DICT_ARUCO_ORIGINAL" que el que propone ArUco, que tiene 1024 marcas de 5x5 bits con una distancia mínima de 0 entre sus códigos.

El proceso de detección de las marcas solo identificará las que defina el diccionario que se le haya dicho que utilice y obtendrá el identificador que el código de bits interno (la secuencia de casillas internas a blanco o negro, véase la Figura 5) de la marca tenga asignado, esto es, la posición en el diccionario en que se haya generado esa marca. Puesto que estas son imágenes binarias y se

7 La lista de diccionarios se puede consultar en la URL <https://docs.opencv.org/trunk/d9/d6a/group__aruco.html#ggac84398a9ed9dd01306592dd616c2c975a6c2a6d747864ef4daad8cdeffa07f7c0>.

construyen con unas restricciones geométricas impuestas para facilitar su detección y decodificación en condiciones de imágenes con deformaciones (p .ej. debidas a la perspectiva) o con ruido (p. ej. debidas a las condiciones de iluminación o a los parámetros de segmentación) su robustez es grande y el tiempo de respuesta es corto.

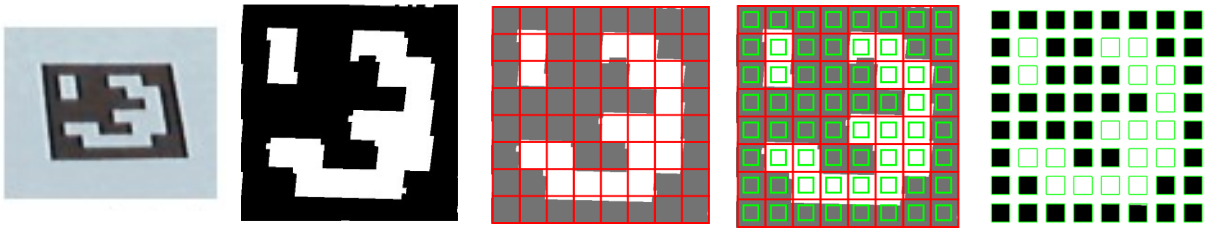


Figura 5: Proceso de decodificación de una marca. Imágenes extraídas de [9].

2.1 Creación de marcas simples

Para poder elaborar una aplicación de RA es necesario generar una de estas marcas y utilizar un soporte que permita su visualización, el papel o la pantalla de un dispositivo, al que podamos dirigina nuestra cámara. Para ello utilizaremos una de las aplicaciones que hemos compilado al ejecutar las órdenes del Listado 1. Utilizaremos[8] *create_marker*, p,. ej. indicando que queremos que se emplee la marca en posición 1 del diccionario "DICT_ARUCO_ORIGINAL" y que se guarde en el fichero "marca.png" con :

```
$ create_marker -d=16 --id=1    marca.png
```

Si se ejecuta sin parámetros podrá observar todos los parámetros que puede recibir esta orden y porque los que hemos escogido. Hágalo y compruebe cómo se indica con el valor 16 el diccionario a utilzar y con 1 el número de la marca de ese diccionario. Para visualizar la marca se puede hacer con

```
$ gwenview marca.png
```

Ejercicio 1. Visualizar la marca que corresponde con el número del computador del equipo del laboratorio en que se hace la práctica, de los diccionarios DICT_4X4_50 y DICT_ARUCO_ORIGINAL.

Dado el tamaño de algunos diccionarios es difícil mostrarlos al completo en una imagen, pero se pueden generar una a una sus marcas en un directorio y, después, visualizarlas.

2.2 Detección de marcas simples

'Ahora que ya tenemos una marca en pantalla o en papel, ya podemos utilizar otra de las utilidades que hemos compilado para su detección: es *detect_markers* y se puede ejecutar con

```
$ detect_markers -d=16
```

De esta forma le decimos que estamos empleando el diccionario 16 (DICT_ARUCO_ORIGINAL), de donde extraerá los valores para el tamaño de la rejilla que debe utilizar para identificar el código, los bits, de esta marca. La Figura 6 muestra dos ejemplos de detección y como se resalta la imagen de la marca y la esquina superior izquierda de la misma, en caso de correspondencia de la marca

8 Existen versiones en línea que podrían también utilizarse en algunos casos como: "ArUco markers generator!" <http://chev.me/arucogen/> y "Generate ArUco Markers for printing" <https://tn1ck.github.io/aruco-print/>.

con el diccionario especificado.

a) b)

Figura 6: Ejemplo de marca detectada (a) y no detectada (b).

Ejercicio 2. Mostrar la marca que corresponde con el número del computador del equipo del laboratorio, en que se hace la práctica (si está en el laboratorio) o la que resulte de contar las letras que componen su nombre completo (en caso contrario)de los diccionarios DICT_4X4_50 y DICT_ARUCO_ORIGINAL para comprobar si se detectan ambas marcas o no.

2.3 Marcas compuestas: tableros (*boards*)

Como ya se mostró en la o (Figura 4a), otros tipos de marcas compuestas se pueden emplear para tener mayor robustez en el proceso de detección y localización de las referencias. Así, es posible crear un tablero de los que se denominan *ArUco board*. Un ejemplo de este tipo de tablero se muestra en la Figura 7, junto al resultado de su detección. Para ello se han de ejecutar la pareja de órdenes:

```
$  create_board -d=16 -h=4 -w=4 --si=true --l=100 --s=110 board__DICT_ARUCO_ORIGINAL.png
$ detect_markers -d=16
```

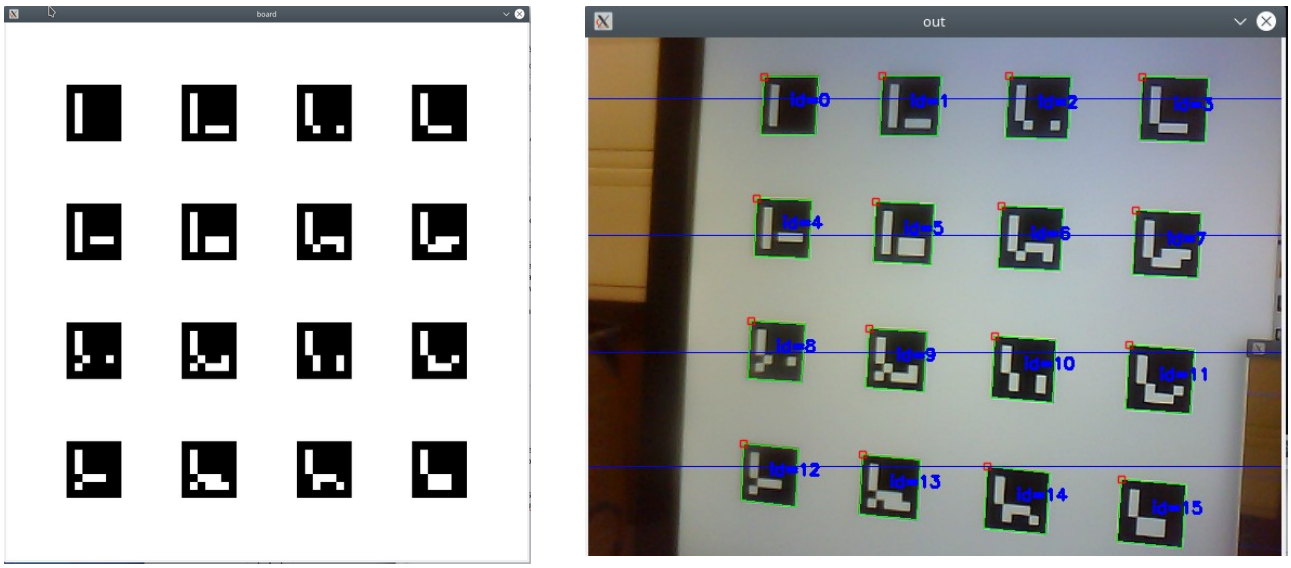


Figura 7: ArUco board (izquierda) y el resultado de su detección (derecha).

Ejercicio 3. Copie el significado de los parámetros de la invocación de la orden *create_board* que se acaba de proponer. Para obtenerlos ejecute *create_board* sin parámetros.

También se pueden crear otros tableros denominados ChArUco, como el que se muestra en la Figura 8 junto al resultado de su detección. Para ello se han de ejecutar la pareja de órdenes:

```
$ create_board_charuco -d=16 -h=4 -w=4 --si=true --ml=200 –sl=210  charuco__DICT_ARUCO_ORIGINAL.png
$ detect_markers -d=16
```

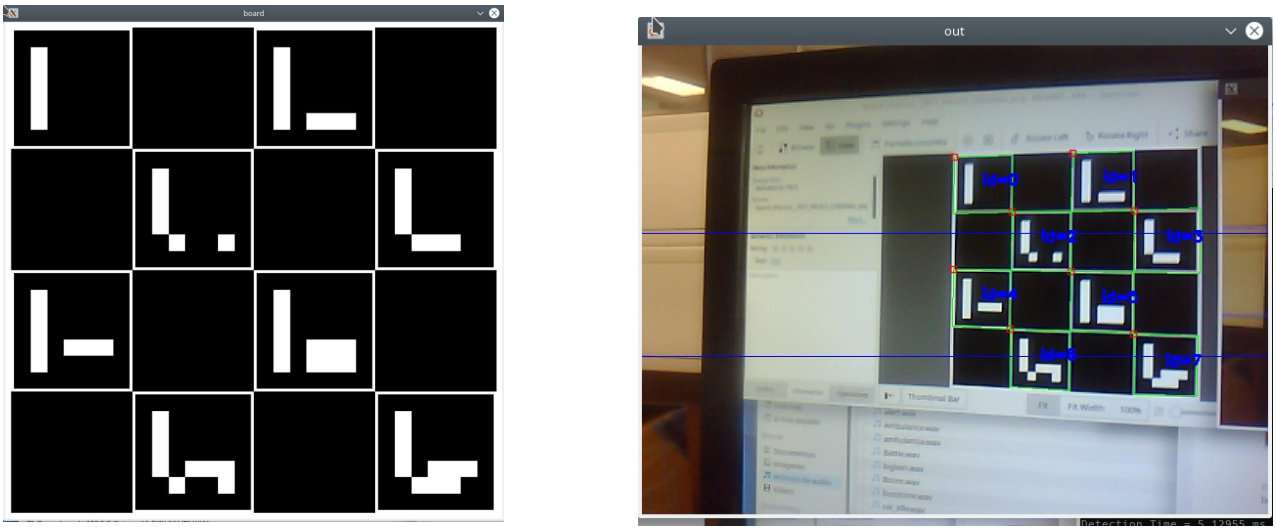


Figura 8: ChArUco board (izquierda) y el resultado de su detección (derecha).

2.4 Calibración

La calibración de la cámara es[9] el proceso que permite inferir la relación entre el sistema de coordenadas de la cámara (2D) y el del mundo real (3D). La Figura 9 muestra la relación entre el punto *P* de coordenadas *(X, Y, Z)* del mundo real, con el punto del plano de la cámara en coordenadas *(u,v)*. Conocida esta relación será posible determinar la medida de un objeto o la distancia entre dos puntos del mundo real a partir de la distancia observada en píxeles en una imagen.

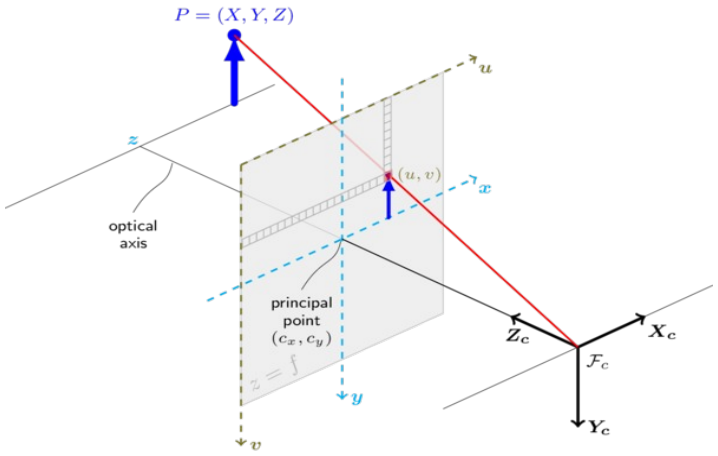


Figura 9: Relación entre los sistemas de coordenadas dentro y fuera de la cámara. Imagen extraída de OpenCV "Camera Calibration and 3D Reconstruction".

9 "Camera Calibration and 3D Reconstruction" <https://docs.opencv.org/3.2.0/d9/d0c/group__calib3d.html>.

Para obtenerla es necesario determinar la relación geométrica de la proyección perspectiva (lo que se denomina parámetros extrínsecos) y la posible deformación que tenga la óptica o el sensor de la cámara (denominados parámetros intrínsecos), como se muestran en la Figura 10.

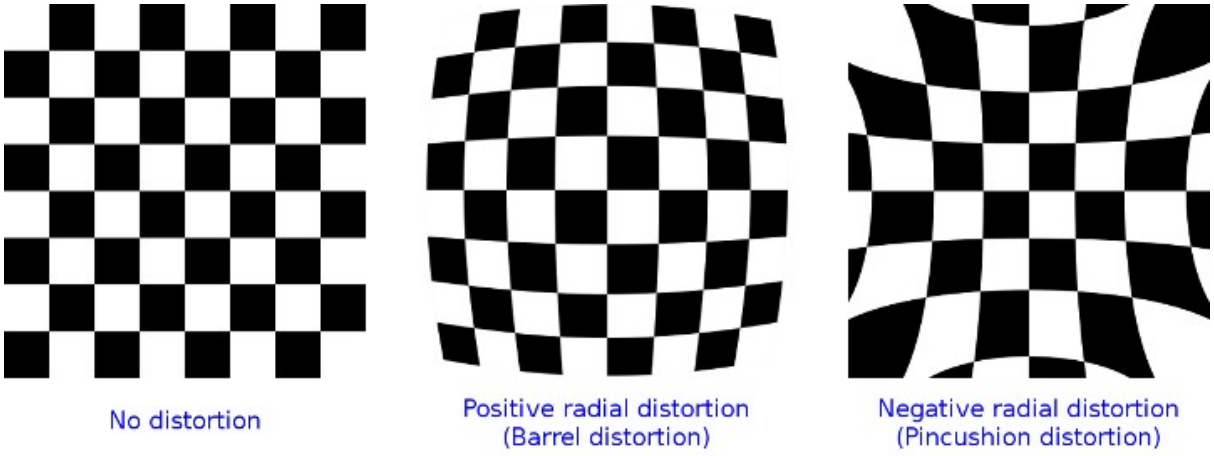


Figura 10: Ejemplos de distorsión óptica en una cámara.

Para facilitar la correspondencia entre puntos de un objeto 3D y la imagen 2D se recurre a objetos denominados "patrones de calibración", de geometría conocida y que sean bien observables en una imagen. Estos se colocan en diferentes posiciones con la cámara fija (o también se puede hacer al revés) y se obtendrá un conjunto de ecuaciones cuya solución son los parámetros de la calibración. OpenCV incorpora para esta tarea el uso de tableros de ajedrez (*chessboard*) y su detección a través de operaciones como *findChessboardCorners()*[10], así como *solvePnP()* o *solvePnPRansac()* para determinar la solución al sistema de ecuaciones que relaciona la correspondencia de puntos 3D y 2D..

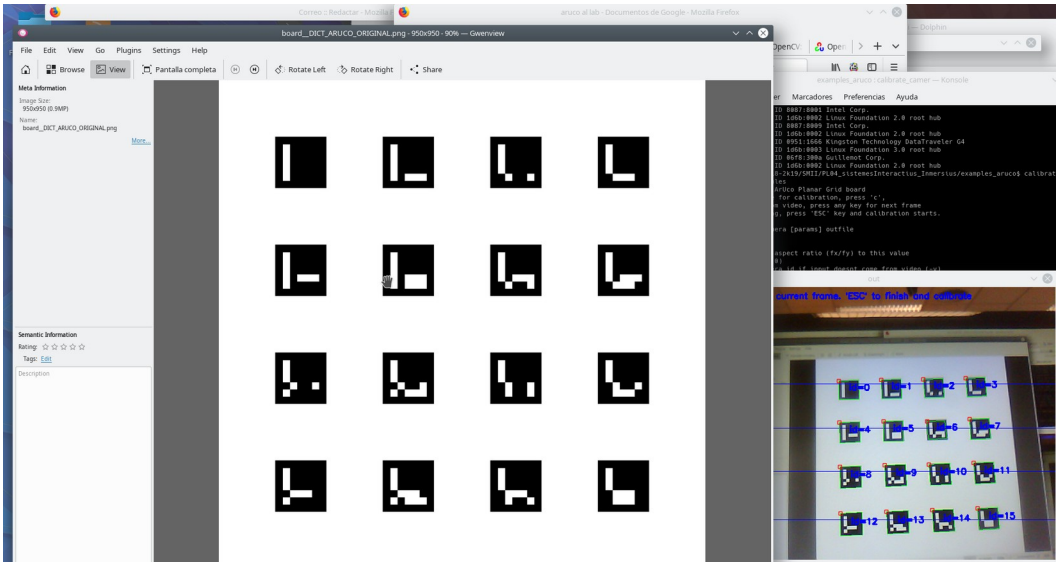


Figura 11: Calibrando la cámara mediante un tablero de marcas de ArUco.

10 Véase <https://docs.opencv.org/3.2.0/d9/d0c/group__calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a >.

```
%YAML:1.0
---
calibration_time: "Tue Mar 12 17:18:35 2019"
image_width: 640
image_height: 480
flags: 0
camera_matrix: !!opencv-matrix
  rows: 3
  cols: 3
  dt: d
  data: [ 3.472084653423415e+02, 0., 3.1925693404345191e+02, 0.,
          3.7439242160984975e+02, 3.9849814228474810e+02, 0., 0., 1. ]
distortion_coefficients: !!opencv-matrix
  rows: 1
  cols: 5
  dt: d
  data: [ 1.3421330070041737e+00, -3.1557957089408442e+00,
          -2.6691366957747483e-02, -5.3727924286901221e-02,
          2.3329511200063351e+00 ]
avg_reprojection_error: 2.2234105643994287e+01
```

Listado 3: Ejemplo de fichero YAML resultante de la calibración de la cámara del laboratorio.

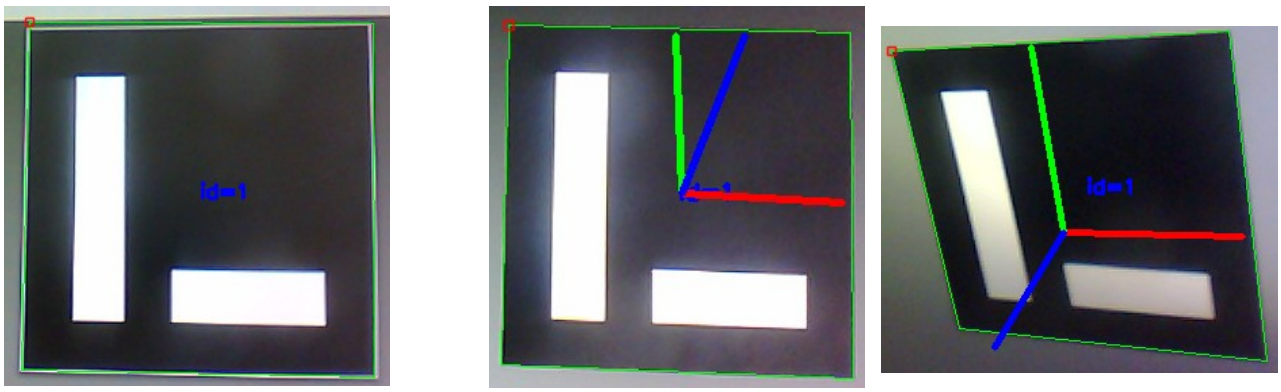
Para esta tarea, en ArUco se utiliza *calibrate_camera* y un tablero, a modo de patrón de calibración, uno realizado con marcas de uno de sus diccionarios. En la Figura 12 se observa un patrón fijo que está siendo observado desde diferentes puntos con la cámara que se quiere caracterizar. La orden utilizada es:

```
$ calibrate_camera calibracio_hercules.yml -d=16 -h=4 -w=4 -l=100 -s=110
```

Se deberían tomar imágenes desde diferentes puntos, barriendo el espacio donde va estar la cámara. Cuanto más preciso sea este muestreo, más preciso será el resultado. Para ello, esta aplicación permite tomar una imagen cada vez que se pulsa la tecla ‘c’ y cuando se considere, al pulsar la tecla ESC se procede a calcular la calibración. Lo que genera un fichero en formato YAML, con un contenido equivalente al que muestra el Listado 3.

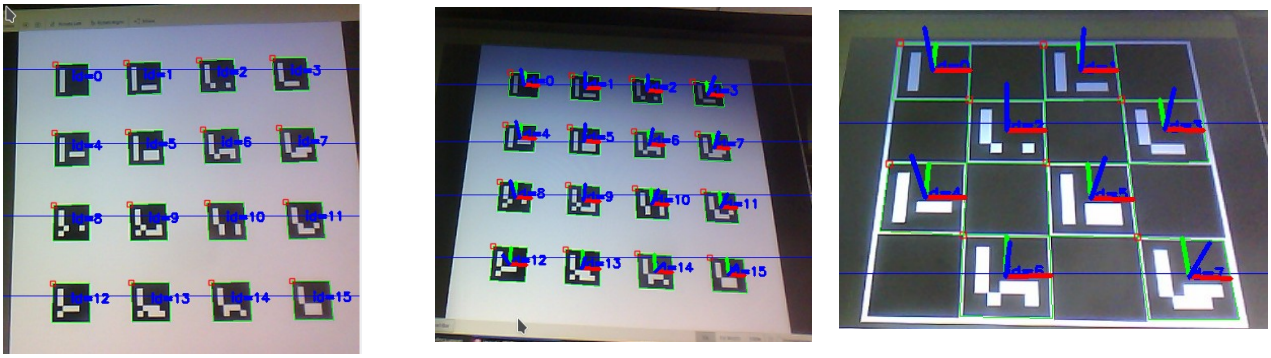
De esta forma, ahora la detección de las marcas se puede enriquecer con información de la posición en el mundo real (3D) de las mismas. La muestra ejemplos de detección como los vistos en la Figura 6 , ahora con la información de posición añadida al ejecutar el detect_markers sobre marcas simples (Figura 12) o compuestas:

```
$ detect_markers -d=16 -c=calibracio_hercules.yml
```



a) b)

Figura 12: Detección básica de una marca (a) y utilizando la calibración de la cámara (b) para mostrar que se puede establecer la posición de los objetos.



a) b)

Figura 13: Detección de marcas en un tablero (s) y posicionamiento (b) resultado de la calibración.

3 Actividades

Tras leer este boletín y experimentar con los ejercicios del mismo, deberá realizar el ejemplo "Augmented Reality using ArUco Markers in OpenCV" [12]. Se puede consultar la referencia y también la copia local (en PDF y ZIP) por si falla la red.

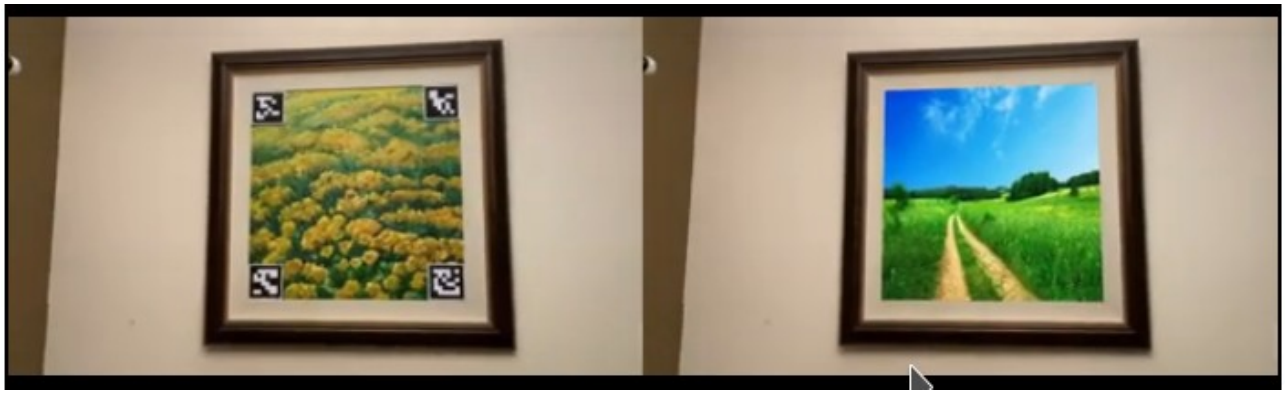


Figura 14: Un instante de la ejecución del ejemplo original de [12].

El código de este ejemplo consiste en dos ficheros:

1. Uno (*generate_aruco_markers.cpp*) que muestra cómo generar una de las marcas que se emplean en la parte.
2. Otro (*augmented_reality_with_aruco.cpp*) que utiliza las marcas 25, 33, 30 y 23 para definir un espacio en el que situar una imagen y que graba, en disco, un vídeo donde se ve, como en la Figura 14, a la izquierda la imagen que obtiene la cámara del mundo real y a la derecha la que resulta de "ampliar" el mundo con la sobreimpresión de un objeto virtual, en este caso una imagen estática..

El ejemplo no trabaja con la cámara web, en su lugar puede:

1. Utilizar un fichero estático, *test.jpg*, que contiene la imagen de un cuadro en la pared con cuatro marcas. Este modo se ejecuta al lanzar la aplicación con
   $ augmented_reality_with_aruco --image=test.jpg
   lo que mostrará la imagen de la Figura 14 y que sirve para comprobar que funciona la detección de las marcas.
2. Utilizar un fichero de vídeo, *test.mp4*, que contiene una secuencia grabada de imágenes del mismo cuadro con las marcas del caso anterior. Ese modo se ejecuta con
   $ augmented_reality_with_aruco. –video=test.mp4
   y permite comprobar que la posición y perspectiva de la cámara y el cuadro se utilizan para ajustar la imagen sobrepuesta a las variaciones de distancia y perspectiva que provocan el movimiento de la cámara.

El código del ejemplo se puede compilar con una versión de OpenCV mayor o igual que la 3.2.0 que utilizamos en el laboratorio (o en la máquina virtual), en OpenCV 3.2 con:

```
$ g++ -std=c++11 augmented_reality_with_aruco.cpp -o augmented_reality_with_aruco `pkg-config --cflags --libs opencv`
```

y en OpenCV 4.2 con:

```
$ g++  augmented_reality_with_aruco.cpp -o augmented_reality_with_aruco `pkg-config opencv4 --cflags –libs`
```

Recordad que una de las metas de nuestra asignatura es ver como el escritorio virtual del computador se expande/conecta con el escritorio real (el área de acción) del usuario. Así que **queremos desarrollar la parte de uso de la cámara web** de este ejemplo para poder hacer este pequeño divertimento de inmersión con RA, de modo que podamos situar las marcas en cualquier lugar, p. ej, sobre la mesa de trabajo, no hace falta pegarlas sobre ningún cristal o pared y pensad que podría ser una manera de interactuar con el computador, que es lo que nos interesa.

Como respuesta a esta práctica se deberán realizar las oportunas modificaciones en el código del proyecto reseñado para alcanzar los siguientes objetivos:

1. Ampliar el código de generación de las marcas (*generate_aruco_markers*) para obtener los cuatro ficheros con las cuatro marcas que se utilizan en el ejemplo de partida en cuatro ficheros con el esquema de nombres que utiliza para el único que genera el código original.
2. Ampliar el código de *augmented_reality_with_aruco* para que tome como fuente de vídeo la cámara que se indique con
   $ *augmented_reality_with_aruco.out –device=0*
   para la cámara por defecto y cuyo valor numérico se puede utilizar para referenciar otras cámaras conectadas en el equipo.
3. Modificar el ejemplo original (*augmented_reality_with_aruco*) para que la imagen estática que se sobrepone a las marcas, se pueda cambiar por un fichero en disco. Se

creará uno con la cara de quien/quienes entregan el porfolio y se adjuntará en el porfolio y se ejecutará con cualquiera de estas tres órdenes
$ augmented_reality_with_aruco –image=test.jpg –iOverlay=foto.jpg
$ augmented_reality_with_aruco –video=test.mp4 –iOverlay=foto.jpg
$ augmented_reality_with_aruco –device=0 –iOverlay=foto.jpg
para mostrar la imagen foto.jpg, Figura 15a), sobrepuesta a la entrada de vídeo seleccionada, .

4. Modificar el ejemplo *augmented_reality_with_aruco* para que pueda mostrar un vídeo sobre las marcas en lugar de una imagen estática. Utilizaremos el vídeo "laser.avi", Figura 15b), utilizado en la práctica 2 de introducción a OpenCV y se ejecutará con cualquiera de estas tres órdenes
$ augmented_reality_with_aruco –image=test.jpg – vOverlay=laser.avi
$ augmented_reality_with_aruco –video=test.mp4 – vOverlay=laser.avi
$ augmented_reality_with_aruco –device=0 –vOverlay=laser.avi

5. Modificar *augmented_reality_with_aruco* para que la imagen de este tercer objetivo se quede en la escena cuando termine de reproducirse el vídeo anclado a las marcas y mientras el usuario no decida terminar pulsando la tecla ESC y que se ejecutará, p. ej con
$ augmented_reality_with_aruco –device=0 –iOverlay=XYZ.jpg –vOverlay=laser.avi

Con ello, el interfaz del ejemplo quedará como:

```
const char* keys =
"{help h usage ? | | Usage examples: \n\t\t./augmented_reality_with_aruco.out --image=test.jpg \n\t\t./augmented_reality_with_aruco.out --video=test.mp4}"
"{image i       |<none>| input image   }"
"{video v       |<none>| input video   }"
"{device d      |<none>| num. camara }"
"{iOverlay y    |<none>| imatge per a sobrepossar (overlay) }"
"{vOverlay o    |<none>| vídeo per a sobrepossar (overlay) }"
;
```

a) *b)*

Figura 15: Ejemplos de salida del ejemplo augmented_reality_with_aruco: (a) con una imagen estática y (b) con un fotograma de un vídeo.

Se habrá de indicar, en el porfolio, las órdenes de compilación y de uso para los objetivos propuestos.

4 Bibliografía y referencias

[1] M. Isberto. (2018). “The History of Augmented Reality”. URL <https://www.colocationamerica.com/blog/history-of-augmented-reality >.

[2] D. Silverman. (2017). , Houston Chronicle. “The strange tale of Monocle, the pioneering AR app you've never heard of”. URL <https://www.houstonchronicle.com/techburger/article/The-strange-tale-of-Monocle-the-AR-pioneer-12371889.php>.

[3] *OpenCV Library*. URL <https://opencv.org/>.

[4] *ArUco: a minimal library for Augmented Reality applications based on OpenCV*. URL <https://www.uco.es/investiga/grupos/ava/node/26>.

[5] "Speeded up detection of squared fiducial markers", Francisco J.Romero-Ramirez, Rafael Muñoz-Salinas, Rafael Medina-Carnicer, Image and Vision Computing, vol 76, pages 38-47, year 2018 URL <https://www.researchgate.net/publication/325787310_Speeded_Up_Detection_of_Squared_Fiducial_Markers >.

[6] "Generation of fiducial marker dictionaries using mixed integer linear programming",S. Garrido-Jurado, R. Muñoz Salinas, F.J. Madrid-Cuevas, R. Medina-Carnicer, Pattern Recognition:51, 481-491,2016. URL <https://www.researchgate.net/publication/282426080_Generation_of_fiducial_marker_dictionaries_using_Mixed_Integer_Linear_Programming>

[7] R. Muñoz. *ArUco Library Documentation*. URL <https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#>.

[8] *ArUco. Augmented reality library based on OpenCV*. URL <https://sourceforge.net/projects/aruco/files/>

[9] *Detection of ArUco Markers (Versión OpenCV 3.2)* . URL <https://docs.opencv.org/3.2.0/d5/dae/tutorial_aruco_detection.html>.

[10] *Detection of ChArUco Corners*. URL <https://docs.opencv.org/3.2.0/df/d4a/tutorial_charuco_detection.html>.

[11] *Detection of ArUco Markers (Versión OpenCV 34.2)* . URL <https://docs.opencv.org/4.2.0/d5/dae/tutorial_aruco_detection.html>.

[12] S. Nayak. (2020). Augmented Reality using ArUco Markers in OpenCV (C++ / Python). URL <https://learnopencv.com/augmented-reality-using-aruco-markers-in-opencv-c-python/>-

Anexo I. Otras fuentes

Puede curiosear con el uso de Aruco en una aplicación sobre plataforma móvil con la app de ArUco Test de Rafael Muñoz Salinas y en https://play.google.com/store/apps/details?id=com.uco.ava.appcv&hl=es:

The program allows testing the ArUco library for marker detection and camera pose estimation. Additionally, you can calibrate your phone camera and send the calibration file to by email.

En la Figura 16 y Figura 17vemos algunos instantes de su uso.

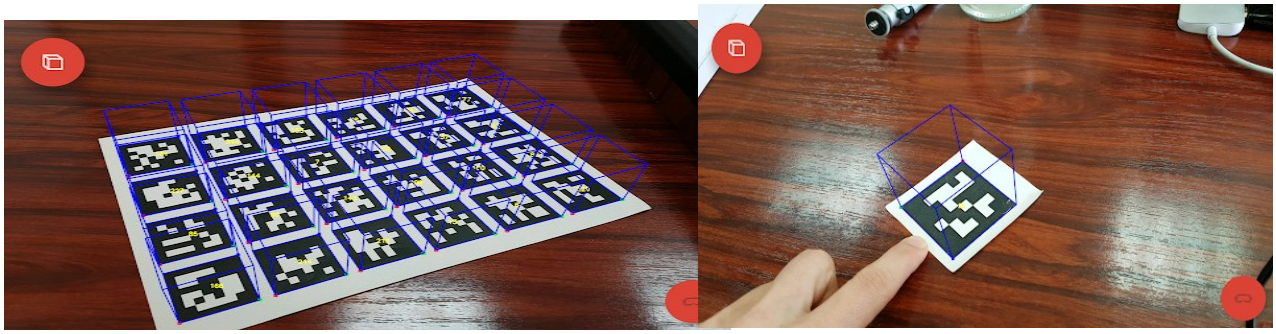


Figura 16: Ejemplo de calibración con Aruco Test.

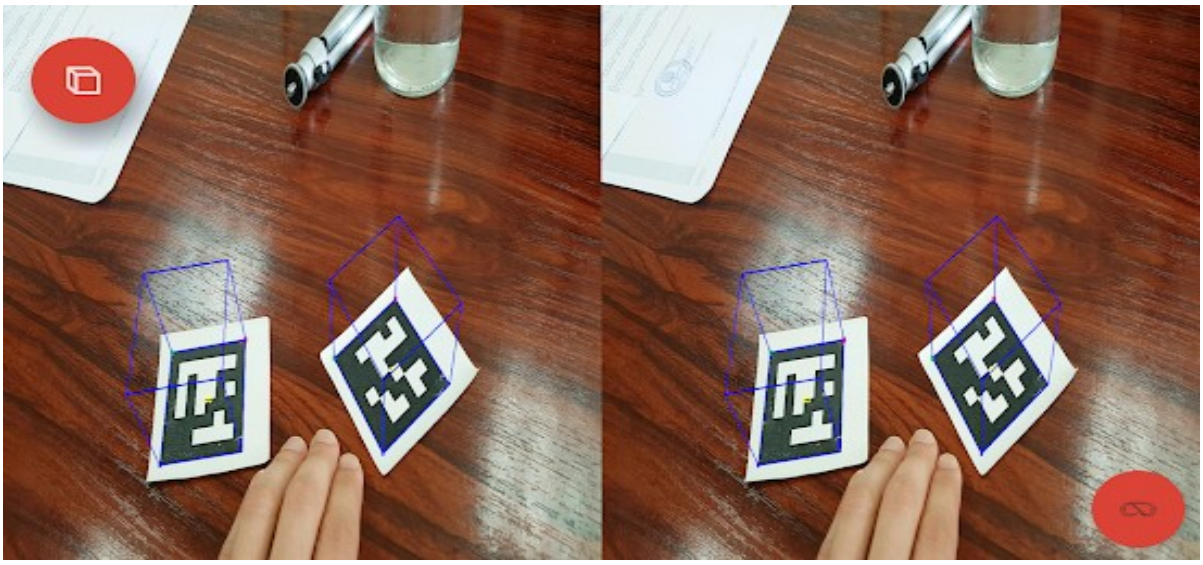


Figura 17: Modo "Split View" de Aruco Test para utilizarla con un visor estereoscópico.