UNIVERSITAT POLITÈCNICA DE VALÈNCIA

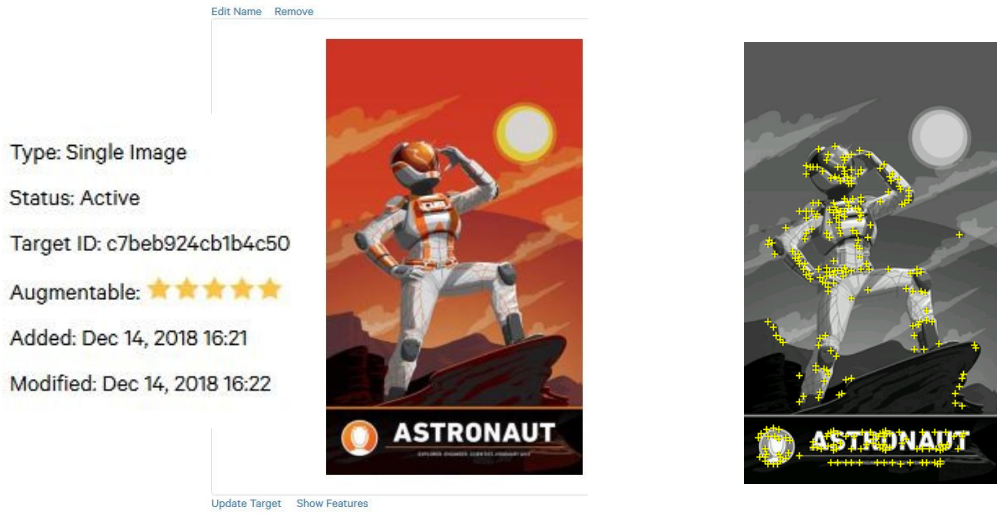
Introducción a la detección de puntos característicos con OpenCV

Apellidos, nombre	**Agustí i Melchor, Manuel** (magusti@disca.upv.es)
Departamento	**Departamento de Informática de Sistemas y Computadores**
Centro	Universitat Politècnica de València

1 Resumen de las ideas clave

Actualmente los computadores son capaces de analizar imágenes para encontrar en ellas patrones de referencia que les permitan identificar la similitud exacta o el grado de parecido entre dos imágenes, así como la presencia de objetos complejos (manos, rostros, personas, animales, y un largo etcétera) en una imagen. En el caso de las aplicaciones de realidad aumentada (en adelante RA) que utilizan la imagen de una cámara para establecer el posicionamiento del usuario en la escena, se pueden encontrar: las que ofrecen el uso de un patrón prefijado, como una marca de referencia y las que permiten al usuario asignar una imagen de su elección, para que la aplicación la busque en la escena y a partir de la que se establece la relación entre el mundo real y el espacio virtual en la pantalla del computador.

Revisando el uso de una de las librerías más populares en el campo de la RA, encontramos el caso de Vuforia [1], en la que podemos hacer uso de un asistente que nos permite escoger la imagen que queramos utilizar de referencia y el sistema nos ofrece una valoración de lo "adecuada" que es para este tipo de aplicaciones de RA: lo hace con una característica que llama "Augmentable" y que valora de menos a más (entre 0 y 5 estrellas), como muestra la Figura 1a. En este proceso, la imagen escogida también se muestra, como aparece en la Figura 1b, en niveles de gris (obsérvese que la imagen original es en color) y con una serie de cruces en color amarillo. Esas señales identifican puntos característicos de una imagen. Estos son el germen de las estrategias actuales que se usan para caracterizar una imagen de forma automática, esto es, liberando al desarrollador de la aplicación de definir qué serie de parámetros geométricos o estadísticos son los adecuados para reconocer un patrón rígido en una escena.



a) b)

Figura 1: Ejemplo de Vuforia valorando una imagen para ser utilizada de referencia en una aplicación de RA. Imagen obtenida de [1].

Así, es habitual encontrarnos con estas estrategias para aplicaciones de RA a partir de marcas impresas (en papel y en productos) para campañas publicitarias, en juegos y en la generación de escenas que incluyan la visualización de modelos tridimensionales de objetos (en etapas de diseño) que se quiere considerar su resultado si entran a formar parte de la escena, o modelos de objetos reales que se quiere observar en 3D desde diferentes perspectivas.

En este trabajo vamos a enunciar cómo es posible obtener una de estas descripciones de imágenes mediante puntos característicos utilizando la biblioteca de funciones *OpenCV*. Veremos la secuencia de pasos necesaria para obtener esta representación y propondremos un interfaz visual para experimentar con los parámetros de estas técnicas y obtener un resultado en el que veamos qué puntos se escogen por parte de diferentes algoritmos disponibles.

2 Objetivos

El objetivo principal de este trabajo es explorar de forma práctica y visual el concepto de puntos característicos como definitorios del contenido de una imagen.

Una vez que el lector haya experimentado con el ejemplo de código proporcionado para entender el concepto y que existen diferentes alternativas para calcularlo, será capaz de:

- Aplicar una estrategia de cálculo de puntos característicos para el reconocimiento de la presencia de un cierto patrón en una escena utilizando *OpenCV*.
- Valorar de forma cualitativa el resultado de algunos de los algoritmos que ofrece *OpenCV* para el cómputo de puntos característicos.

En lo que sigue, se asume que se está trabajando en una plataforma *GNU/Linux* donde ya existe una versión de *OpenCV* instalada, pero la elección de las herramientas sugeridas permite estar trabajando en cualquier otra plataforma. Cuento con que el lector conoce esta librería a un nivel básico para entender los ejemplos de código.

Si está interesado en el código completo para explorar las ideas expuestas, no tiene más que enviarme un correo electrónico y se lo haré llegar. No quiero llenar las páginas con los detalles de la implementación, así que verá en algunos casos el uso de "..." para indicar que se obvia código en los listados que se incluyen en este trabajo.

3 Introducción

En el contexto de desarrollo de aplicaciones de RA se puede recurrir al uso de los sensores de posicionamiento que suelen llevar los dispositivos móviles actuales. La variabilidad en la precisión que ofrecen estos sensores, su alto consumo de la batería por su uso continuo, el tiempo de respuesta de la aplicación que ha de esperar a que estabilice la salida de estos sensores y la falta de ellos en otros dispositivos (como los equipos de sobremesa y un buen número de sistemas embebidos o empotrados) han dado lugar a que se utilice la imagen que proporciona una cámara digital para establecer esta posición del usuario en la escena.

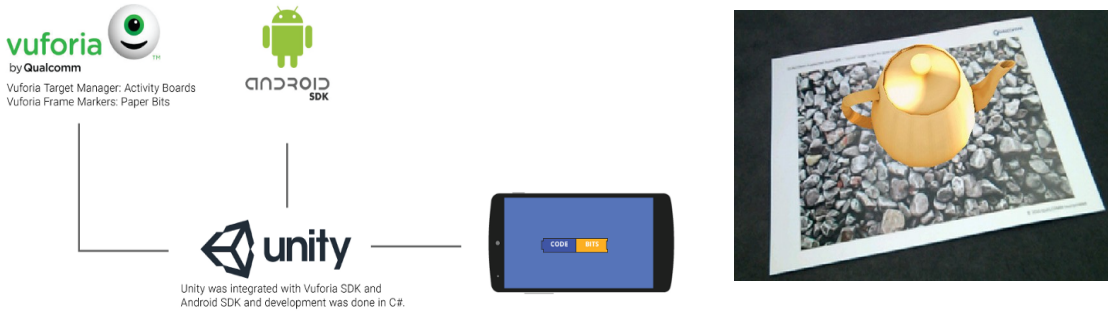
El uso de una imagen de referencia en la escena permite obtener la posición relativa de la cámara en la misma y, también, puede extraerse información codificada en la referencia. Con lo que puede ser una imagen de un conjunto predefinido (como es caso habitual de las marcas fiduciales en blanco y negro) o codificada (como es el caso en que usan códigos 1D o 2D). Incluso pueden ser imágenes escogidas en tiempo de ejecución y cuyo contenido tenga un significado para el usuario: puramente de carácter publicitario (si se reconoce la imagen de referencia, lo que enfatiza el uso de productos de una marca comercial) o sirve para asociarla con la acción que va realizar la aplicación al detectarla. Para ello es necesario disponer de una estrategia de identificación de un patrón visual en la escena.

¿Cómo se puede llevar a cabo esta tarea? Hay opciones como *ARKit*[1] o *ARCore*[2] que han aparecido con fuerza en el mercado de los dispositivos móviles que ofreciendo acceso especializado y optimizado al hardware asociado a cada uno de los fabricantes de estas interfaces de desarrollo para RA. De momento, solo son de uso en estos dispositivos, así que buscamos otras opciones más portables.

La única opción totalmente portable entre plataformas para detectar patrones visuales y extraer la información de la escena de posición de los mismos pasa por un estándar como ofrece OpenCV. Así que sobre esta plataforma de desarrollo vamos a ver los ejemplos. Describiremos en primer lugar cómo lo hace el caso que ha motivado este trabajo (Figura 1) para pasar a exponer formas de implementarlo con OpenCV.

3.1 Vuforia: *Image Targets*

Vuforia no es la única opción, pero es una de las que ha conseguido popularidad en este campo por su difusión y resultados [2] en plataformas móviles, siendo su objetivo la búsqueda de una imagen de referencia. En el contexto de aplicaciones de RA, es necesario además, desarrollar cuidando el interfaz de usuario, tanto en lo que respecta a sus detalles multimedia como a la interacción. Así que es habitual ver ejemplos de desarrollo en los que *Vuforia* se integra con otras herramientas de autor como es el caso de *Unity 3D* [3]. Como muestra la Figura 2a, es usual encontrar a estas dos componentes en colaboración en aplicaciones sobre dispositivos móviles y obtener así, Figura 2b, la posición donde ubicar un objeto sintético (la tetera) sobre una marca (la imagen de las piedras) y actualizar la del objeto virtual con la del físico en tiempo de real.



a) b)

Figura 2: RA con Unity 3D y Vuforia: Componentes de una aplicación de (a) y ejemplo de salida (b). Imágenes de [2] y [3]

Vuforia denomina *Image Targets* a las imágenes que su motor de reconocimiento ha visto previamente y "sabe" detectar. Para ello, elabora una descripción de la imagen basada en "características que se encuentran naturalmente en las imágenes" [3]. Esta descripción es la que buscará posteriormente en las imágenes a analizar. La descripción, como se veía en la Figura 1, es mostrada al usuario de forma gráfica, acompañada de una valoración de lo adecuada que es la imagen propuesta y de una visualización en la que aparecen unos puntos resaltados con unas pequeñas cruces.

1 Véase el sitio web de <https://developer.apple.com/arkit/ >.

2 Véase el sitio web de <https://developers.google.com/ar/> .

3.2 Extracción de puntos característicos con OpenCV

OpenCV pone a disposición del desarrollador algunos algoritmos para detección de objetos como esquinas líneas, círculos, rectángulos, *blobs* o caras (entre otros). Cuando no existe una definición cerrada de cómo es un objeto, se puede utilizar la detección de "puntos" que definan la forma de un cierto objeto y la presencia y en cierta disposición espacial relativa de estos. Esto permitirá lanzar hipótesis de la existencia de esos objejos en una imagen. Los puntos característicos o de interés (denominados *sobresalient* points, *feature points* o *keypoints*, en la terminología anglosajona) son la denominación que reciben esos puntos, o pequeñas áreas de una imagen, que son, con una alta probabilidad, parte de algún objeto en una escena.

Podemos encontrar ejemplos prácticos de uso [4] y [5] en que se entremezclan el uso de algún algoritmo de extracción de puntos característicos, con la correspondencia de estos en una nueva imagen. EL SDK de *OpenCV* ofrece varios algoritmos[3] (no los vamos a ver todos, tranquilos) para la detección de puntos característicos [7] como *AGAST*, *GFTTDetector*, *FastFEatureDetector*, *ORB*, *AKAZE*, *BRISK*, *KAZE* o *MSER*. También existen otros[4] que OpenCV los divide en dos grupos un tanto especiales: experimentales (*BoostDesc*, *BRIEF*, *DAISY*, *FREAK*, *LATCH*, *LUCID*, *MSDDetector*, *PCTSignaturesSQFD*, *StarDetector* o *VGG*) y extra (compuesto por *SIFT*, *SURF* y una versión de *SURF* optimizada para *CUDA*, sobre los que existen patentes y por lo tanto posibles restricciones de uso).

En este trabajo nos centramos en explorar la forma de incluir en una aplicación propia, un algoritmo de detección de puntos característicos, y aunque existen implementaciones del proceso completo, como p. ej. [6], queremos revisar la posibles formas de obtenerlos, sin entrar a establecer una comparativa, puesto que no es posible darla de forma absoluta.

4 Desarrollo

Los trabajos de [4] y [5], así como el ejemplo de la distribución de *OpenCV* [6], permiten comprobar cómo se realizar el proceso de extracción de puntos característicos y cómo se emplea en la etapa posterior de detección de la imagen de referencia. Como dice en el propio código: "no se puede decir cual es el mejor así, pero se pueden observar los resultados". Eso es lo que queremos ver y poder explorar los resultados, aunque nos conformaremos con ver la primera parte: qué son los puntos característicos.

Básicamente, la idea es ver cuáles son las opciones que *OpenCV* nos proporciona para la detección de marcas complejas como pueden ser las imágenes que el usuario puede utilizar para activar la respuesta de una aplicación de realidad aumentada y que se define de forma gráfica como muestra la Figura 3.

En la figura se esquematiza que, a partir de la imagen y de la elección de un algoritmo de detección de puntos característicos se obtendrán un conjunto de puntos característicos (*keypoints*), que queremos dibujar sobre la imagen para poder apreciar qué y cuántos detalles de la misma es capaz de extraer el proceso. El usuario deberá poder escoger el método de detección de puntos característicos y el número de estos que se muestran, partiendo del máximo que se detecta con los parámetros por defecto de cada algoritmo.

---

3 Veasé *OpenCV: 2D Features Framework*. Disponible en <https://docs.opencv.org/3.2.0/da/d9b/group__features2d.html>.

4 Tiene toda la información sobre ellos en la documentación de *OpenCV* disponible en <https://docs.opencv.org/3.2.0/d1/db4/group__xfeatures2d.html>.

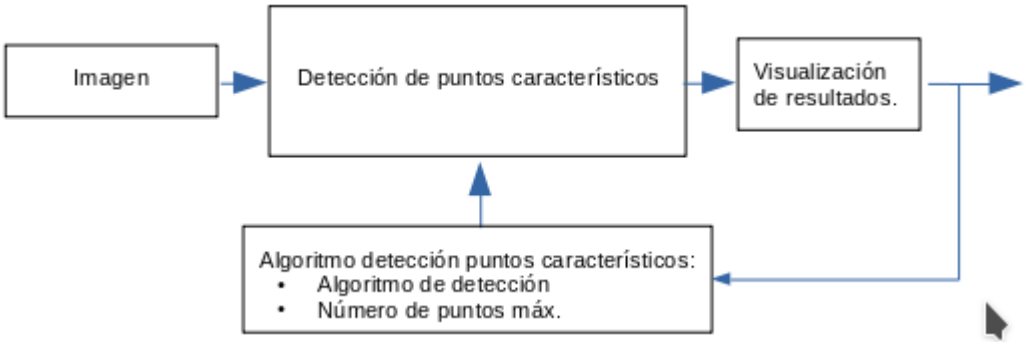


Figura 3: Esquema de la aplicación a desarollar.

Esto puede no parecer justo para alguna de las técnicas, pero puesto que depende de las imágenes y de profundizar en cada algoritmo, se ha preferido dejar fuera del interfaz todo el conjunto de parámetros de cada método y quedarnos solo con este número de características detectadas. Lo que permite poder visualizar más o menos de ellas y así observar qué es lo más “apreciable” para el computador. No se puede evaluar estos métodos sin poner en función de los parámetros de cada uno el resultado obtenido. Habitualmente lo que se hace es comprobar si esos puntos detectados siguen “viéndose” en la imagen modificada por algún grado de ruido, cambios en la iluminación o el contraste y versiones con algún tipo de deformación rígida o elástica aplicada

Como que OpenCV ofrece un método nativo (*drawKeypoints*) que dibuja un círculo sobre la posición de cada punto y, puesto que se solaparán, es más difícil de interpretar, haremos uso de funciones más complejas (como *drawMarker*) para afinar en la visualización del resultado.

4.1 La interfaz y el código

La Figura 4 muestra una captura de la interfaz desarrollado para cumplir con el esquema de funcionamiento de la Figura 3. Las dos barras de desplazamiento de la ventana de la izquierda permitirán variar el número de puntos característicos que se muestran en la ventana de la derecha. El segundo control determinar el índice del algoritmo de detección empleado, según la estructura definida en el código.

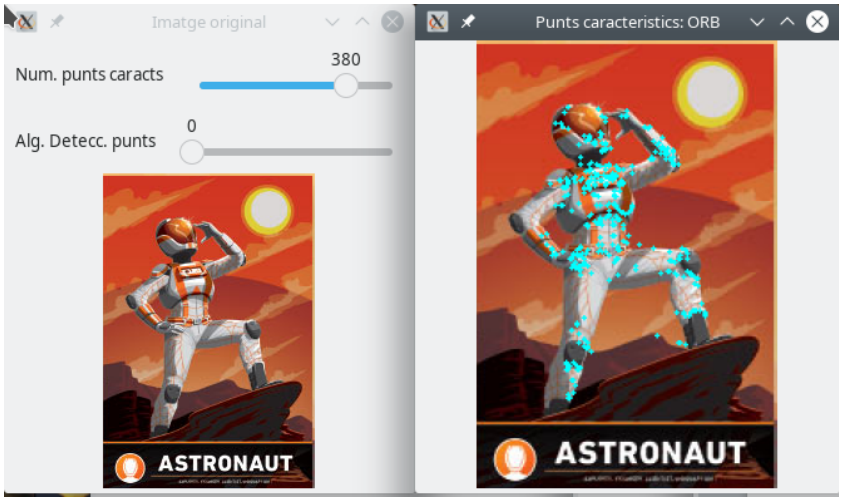


Figura 4: Captura de pantalla de la aplicación en ejecución: utilizando el algoritmo 0 (ORB) y obteniendo 380 puntos en su detección, que están dibujados en la imagen de la derecha.

Veamos algunos elementos importantes del código para establecer la secuencia de operaciones (que resaltaremos en negrita en los listados siguientes) y

facilitar al lector la comprensión del código completo, si se anima a verlo. Las declaraciones inciales se pueden ver en el Listado 1, donde se define el orden empleado con *tipoIdAlgDeteccio y las variables que llevarán la cuenta del método de detección de puntos característicos (algDetecPuntsCaract) y el conjunto de estos (keypoints)*.

```
#include <opencv2/features2d.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/opencv.hpp>
#include <vector>
#include <iostream>
using namespace std;
using namespace cv;


const int MAX_FEATURES = 500;
#define FPPAL "Imatge original"
#define FPC "Punts caracteristics: ORB"
#define ESC 27
#define NMAX_ALG 5


typedef enum{ ALG_ORB=0, ALG_AKAZE, ALG_KAZE, ALG_MSER, ALG_BRISK }
tipoIdAlgDeteccio;
String strIdAlgDeteccio[ NMAX_ALG ] = { "ORB", "AKAZE", "KAZE", "MSER",
"BRISK" };
...
int main(int argc, char **argv) {
  int ample, alt; nPuntsCaract = 0, idAlgDeteccio = 0;
  bool salir = false, haCanviatElNumeroDePunts = false,
       haCanviatElAlgDeteccio = false;
  Mat img, imgGris, imgPuntsCaracteristics;
  Ptr<Feature2D> algDetecPuntsCaract;
  std::vector<KeyPoint> keypoints;
...
```

Listado 1: Código de la implementacón: definiciones inicales.

El programa principal es el encargado de, Listado 2, cargar una imagen de disco (*img*) que se ha de convertir a escala de grises (*imgGris*) sobre la que se aplicará el proceso. Para ello, se inicializa el método a utilizar (inicialmente es el algoritom ORB por lo que se iniciazliza *algDetecPuntsCaract* a *ORB::create*).

Este algoritmo se ejecuta con el método *detec*, que es el que detecta los puntos y se obtiene un número (*keypoints.size*) que depende del algoritmo y del contenido de la imagen.

Ahora solo queda pintar los puntos obtenidos sobre la imagen para poder ver cuáles han sido escogidos.

```
…
  if ( argc == 1 ) {
   printf("Falta un argument: fitxer image (PNG|JPG)\n"); exit( 1 );}
   string refFilename( argv[1] );
   cout << "Reading reference image : " << refFilename << endl;
   img = imread(refFilename);
   if (img.empty()){
    printf("Falla al llegir %s\n", refFilename.c_str() ); exit( 2 ); }

  ample = img.cols;   alt = img.rows;
  namedWindow( FPPAL, CV_WINDOW_NORMAL | CV_GUI_EXPANDED);
  namedWindow( FPC, CV_WINDOW_NORMAL );
  createTrackbar( "Num. punts caracts", FPPAL, &nPuntsCaract,
           MAX_FEATURES, on_trackbarNPunts, &haCanviatElNumeroDePunts );
  createTrackbar( "Alg. Detecc. punts", FPPAL, &idAlgDeteccio,
          NMAX_ALG-1, on_trackbarALG, (void *)&haCanviatElAlgDeteccio );
  moveWindow(FPPAL, 0, 0); imshow(FPPAL, img);
  moveWindow(FPC, MAX_FEATURES*3/4+10, 0);
  resizeWindow(FPC, ample, alt);

  cvtColor(img, imgGris, CV_BGR2GRAY);
  algDetecPuntsCaract = ORB::create( );
  algDetecPuntsCaract->detect(imgGris, keypoints );

  nPuntsCaract = keypoints.size();
  createTrackbar( "Num. punts caracts", FPPAL, &nPuntsCaract,
                 keypoints.size(), on_trackbarNPunts,
                 (void *)&haCanviatElNumeroDePunts );
  pintarMarques(img, &imgPuntsCaracteristics, keypoints,
                keypoints.size());
  imshow(FPC, imgPuntsCaracteristics);
...
```

Listado 2: Secuencia básica de detección de puntos característicos y visualización de los mismos.

El resto del programa principal se ha obviado en este texto, porque es un bucle en el que la actulización de los parámetros del interfaz se lleva a las variables del código que los recogen y se vuelve a lanzar el proceso en secuencia:

1. Recuperando la imagen inicial.
2. Si el usuario ha cambio el método de detcción, se Inicializa el nuevo método, se aplica y se recogen el nuevo conjunto de puntos característicos.
3. Si el usuario ha cambio el método de detcción o el número de puntos a mostrar, hay que volver a pintarlo en una copia de la imagen original.

4. Actulizar la información de los resultados mostrados.

El Listado 3 muestra la función que recorre la lista de puntos característicos y los va marcando en la imagen, obsérvese que de la lista de puntos se extraen sus coordenadas en la imagen (*keypoints.pt*) y que se limita el número de puntos a los que recibe esta función como parámetro (*nPunts*). Este es el valor que el usuario puede modificar con uno de los controles de desplazamiento ubicados en la ventana que muestra la imagen de partida.

```
...
void pintarMarques( Mat img, Mat *imgPuntsCaracteristics,
                    std::vector<KeyPoint> keypoints, int nPunts )
{
  img.copyTo( *imgPuntsCaracteristics );
  for( size_t i = 0; i < nPunts; i++ )  {
        drawMarker( *imgPuntsCaracteristics,
                       cv::Point( keypoints[i].pt.x,keypoints[i].pt.y),
                       Scalar(255, 255, 0), MARKER_CROSS, 3, 1, 8);
  }
} // Fi de pintarMarques( ...
```

Listado 3: Función que realiza el pintado de los puntos característicos en una copia de la imagen original.

4.2 Los resultados

Sobre un máximo de puntos característicos que se configura para los algoritmos probados, la Figura 5 muestra algunos de los resultados obtenidos en cuanto a número de puntos: *ORB* detecta 380, *AKAZE* 142 y *KAZE* 357. Otros esquemas como *MSER* y *BRISK* obtienen 238 y 459, respectivamente. ¿Observas, estimado lector, la similitud de los resultados con el que obtenía *Vuforia* en la Figura 1?

Figura 5: Resultados obtenidos con los algoritmos (de izquierda a derecha): ORB, AKAZE y KAZE.

Es interesante que el lector pruebe a ejecutar el ejemplo y visualizar un número menor de cada método para observar que hay bastante coincidencia en los primeros puntos, esto es que son detectados por la mayoria de métodos.

5 Conclusión

El objetivo principal de este trabajo ha sido explorar de forma práctica y visualmente el concepto de puntos característicos como definitorios del contenido de una imagen. Para ello se ha contextualizado el tipo de aplicaciones que pueden hacer uso de estas técnicas y qué herramientas hay para implementarlas.

Se ha aplicado una estrategia de cálculo de puntos característicos para el reconocimiento de la presencia de un cierto patrón en una escena utilizando *OpenCV*, por su portabilidad y pontencialidad. Se ha resaltado la secuencia de pasos necesaria para obtener esta representación y se ha propuesto un interfaz visual para experimentar con los parámetros de estas técnicas y obtener un resultado en el que sea posible ver qué puntos se escogen por parte de los diferentes algoritmos disponibles.

¡Ánimo, no te quedes con la lectura, hay mucha diversión por el camino experimentando con la aplicación! Si te interesa, envíame un correo y te hago llegar el código completo.

6 Bibliografía

[1] Sitio web de Vuforia. Disponible en <http://www.vuforia.com>.

[2] “Quick Code”. (2018). *Top Tutorials To Learn Vuforia To Develop AR Applications*. Disponible en <https://medium.com/quick-code/top-tutorials-to-learn-vuforia-to-develop-ar-applications-274eedc2b18f>.

[3] *Getting Started with Vuforia Engine in Unity* <https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html>.

[4] S. Akalanka.(2018). *A Comparison of SIFT, SURF and ORB*. Disponible en <https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea>.

[5] S. Mallick. (2018). *Image Alignment (Feature Based) using OpenCV (C++/Python)*. Disponible en <https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python>.

[6] *OpenCV*. Ejemplo ORB, AKAZE y BRISK. Disponible en <https://github.com/opencv/opencv/blob/master/samples/cpp/matchmethod_orb_akaze_brisk.cpp>.

[7] *OpenCV*, *Feature Detection and Description. 2D Features Framework*. Disponible en <https://docs.opencv.org/3.2.0/d5/d51/group__features2d__main.html >.