Universidad Pontificia de Comillas

ICAI

LABORATORIO 3

Visión por Ordenador I

Lydia Ruiz Martínez, Pablo Tuñón Laguna

EXTRACCIÓN DE CARACTERÍSTICAS Y BOLSA DE PALABRAS VISUALES

Curso 2024-2025

Índice

1. Introducción

La visión por ordenador permite extraer y analizar características visuales a partir de imágenes digitales. El siguiente escrito se enfoca en la comprensión extracción de características así como en la construcción de una bolsa de palabras visuales, herramientas fundamentales para la clasificación de imágenes y la identificación de determinados patrones.

Se han estudiado múltiples técnicas de detección de características, estas pueden ser desde esquinas, líneas hasta diferentes puntos de interés. Para ello se han empleando métodos como los detectores de Harris y Shi-TOmasi en detección de esquinas, la transformada de Hough para líneas y detectores como *SIFT* y *KAZE* para la extracción de puntos de interés. Realizado el estudio de extracción, se han organizado dichas carcterísticas en una estructra de bolsa de palabras visuales. Esto permite la creación de un vocabulario el cual es útil para identisticas o tipos de imágenes nuevos con mayor rapidez.

El desarrolo del proyecto se ha realizado en Python mediante la librería OpenCV para la manipulación de imágenes y *helpers* con métodos para obtener los resultados deseados. Con el fin de facilitar al lector que quiere consultar los resultados explicados en el escrito con el programa desarrollado, todo el procesado de imágenes se almacena en el fichero *processed_data* (el cual se crea automáticamente) teniendo este a su vez tres subcarpetas en la que se almacenan las imágenes, hay una para guardar la detección de esquinas (*partA*), una para la de líneas (*partB*) y finalmente una tercera para la detección de puntos de interés (*partC*).

2. Detección de esquinas

A lo largo de la siguiente sección se va a explicar el desarrollo llevado a cabo para la detección de esquinas en imágenes digitales. Para ello se han empleado dos métodos diferentes, el detector de Harris y el detector de Shi-Tomasi. Ambos métodos se han aplicado a diferentes imágenes con el fin de comparar los resultados obtenidos. Sin embargo en un primer momento es necesario cargar las imágenes, para ello, se implementaron dos funciones de lectura que hacen uso de las librerías *opencv* e *imageio*. El método *imread()* en opencv carga las imágenes en formato BGR, mientras que el mismo método en imageio carga las imágenes en formato RGB. Esto produce una alteración en la organización de los colores de la imagen a tener en cuenta ya que algunos métodos de detección de esquinas pueden ser sensibles a los colores.

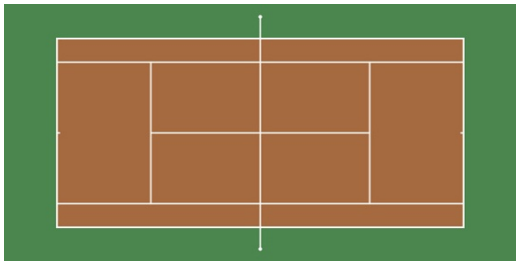

Figura 1: Carga de pista de tenis a través de *imageio* (RGB), autoría propia.

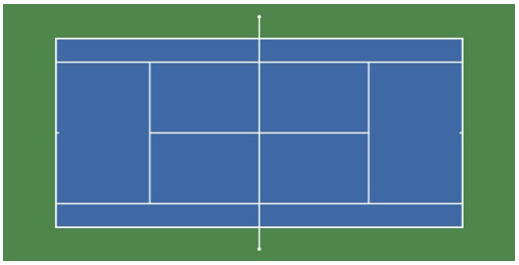

Figura 2: Carga de pista de tenis a través de *opencv* (BGR), autoría propia.

La detección de esquinas se realiza localizando en la imagen procesada cambios fuertes en el gradiente de la imágen, es decir, cambios en la intensidad de los píxeles. Es por ello que tanto para el método de Harris como para el de Shi-Tomasi se ha convertido la imagen a escala de grises. Para ello se ha utilizado el método *cv2.cvtColor()* (cargándola previamente con el método de *opencv*).

2.1. Método de Harris

El método de Harris estudia la variación de la intensidad comentada como una comparación de la intensidad de los píxeles vecinos en todas las direcciones, sumando posteriormente los cuadrados de las diferencias:

$$E(u,v) = \sum_{x,y\ \in W} [I(x+u, y+v) - I(x,y)]^2 \tag{2.1}$$

Dicha pequeña variación de los píxeles cercanos evaluados se puede entender como una derivada discreta, por lo que realizando una aproximación de Taylor de primer orden se puede obtener lo siguiente:

$$E(u,v) \approx \sum_{x,y\ \in W} [I_x u - I_y v]^2 = \sum_{x,y\ \in W} (u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2)$$
$$A = \sum_{x,y\ \in W} I_x^2; \quad B = \sum_{x,y\ \in W} I_xI_y; \quad C = \sum_{x,y\ \in W} I_y^2$$

Por todo lo anterior se puede obtener la matriz de Harris, presente en la expresión:

$$E(u,v) \approx [u\ v] \begin{bmatrix} A & B \\ B & C \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.2}$$

A partir de los distintos valores de la matriz de Harris se puede saber si el conjunto de píxeles evaluados corresponden a una esquina, un borde o una región plana:

- Si A = B = C = 0, entonces se trata de una región plana (no hay variación en ninguna dirección.
- Si A = B = 0 y C != 0, entonces se trata de un borde horizontal.
- Si A != 0 y B = C = 0, entonces se trata de un borde vertical.
- Si A != 0, B != 0 y C!= 0 , entonces se trata de una esquina (cambio significativo en todas las direcciones).

Explicado el fundamento matemático, la parte programística se ha llevado a cabo mediante la función *cv2.cornerHarris()* que recibe como parámetros la imagen en

escala grises, el tamaño de píxeles a evaluar alrededor del píxel central, el tamaño de la apertura del kernel de Sobel (ya que se utiliza para calcular los gradientes) y el valor de la constante de Harris, que en la función permite afinar la sensibilidad del detector. A continuación se muestra la extracción de esquinas mediante el método explicado:

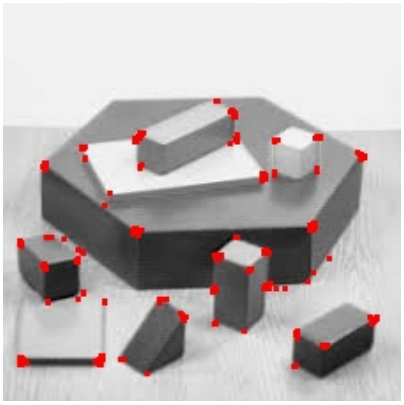

Figura 3: Extracción de esquinas mediante el método de Harris para geometry, autoría propia.

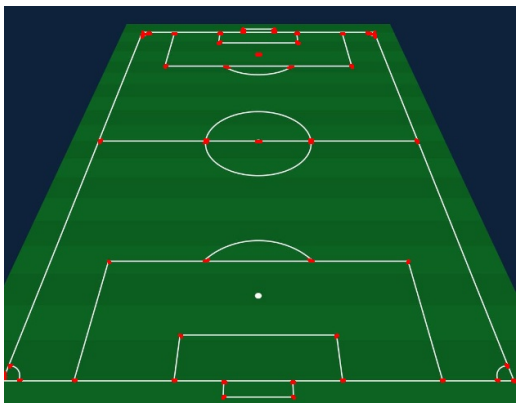

Figura 4: Extracción de esquinas mediante el método de Harris para football, autoría propia.

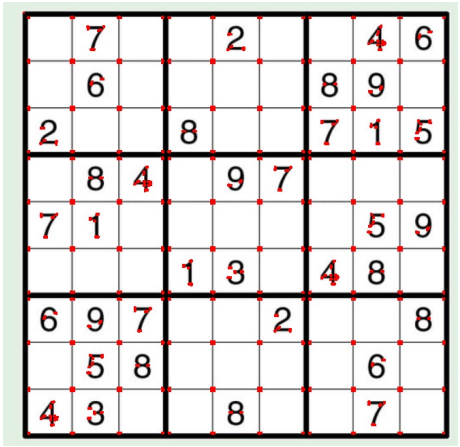

Figura 5: Extracción de esquinas mediante el método de Harris para sudoku, autoría propia.

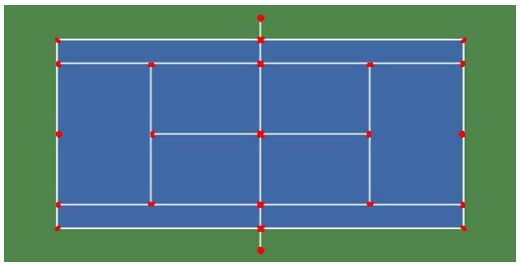

Figura 6: Extracción de esquinas mediante el método de Harris para tennis, autoría propia.

2.2. Método de Shi-Tomasi

El método de Shi-Tomasi presenta algunas mejoras sobre el método de Harris. Este método selecciona las esquinas en función del valor mínimo de los autovalores de la matriz de Harris, resultando en una detección de esquinas más robusta en condiciones variables de iluminación y textura:

- Si $\lambda_1 \approx 0$ y $\lambda_2 \approx 0$, entonces se trata de una región plana (no hay variación significativa en ninguna dirección).
- Si $\lambda_1 \gg 0$ y $\lambda_2 \approx 0$, entonces se trata de un borde horizontal (o veritcal si se intercambia la relación de los autovalores).
- Si $\lambda_1 \gg 0$ y $\lambda_2 \gg 0$, entonces se trata de una esquina (variación en ambas direcciones).

La función *cv2.goodFeaturesToTrack()* permite la detección de esquinas mediante el método de Shi-Tomasi. Esta función recibe como parámetros la imagen en escala de grises, el número de esquinas a detectar, la calidad mínima de la esquina (valor entre 0 y 1) y la distancia mínima entre las esquinas detectadas. A continuación se muestra la extracción de esquinas mediante el método de Shi-Tomasi realizando un una circunferencia de 4 píxeles en torno a ellos en distintos colores:

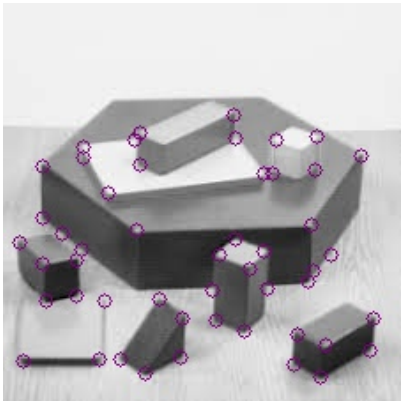

Figura 7: Extracción de esquinas mediante el método de Shi-Tomasi para sudoku, autoría propia.

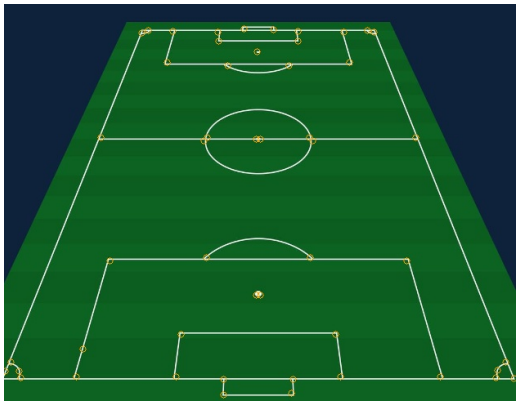

Figura 8: Extracción de esquinas mediante el método de Shi-Tomasi para football, autoría propia.

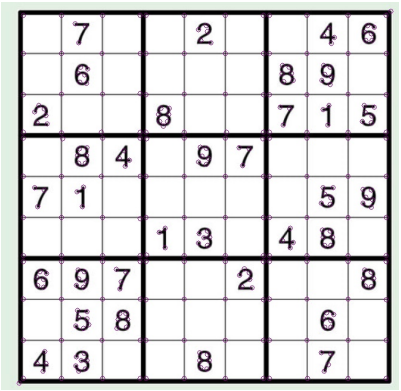

Figura 9: Extracción de esquinas mediante el método de Shi-Tomasi para sudoku, autoría propia.

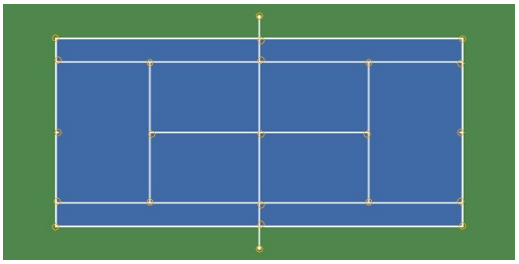

Figura 10: Extracción de esquinas mediante el método de Shi-Tomasi para tennis, autoría propia.

3. Detección de líneas

La detección de líneas permite la identificación de los segmentos que unen los bordes anteriormente detectados en la sección anterior. A lo largo de la siguiente sección se va a exponer el funcionamiento de la transformación de Hough como extractor de líneas. El fundamento matemático de la detección de líneas se basa en la ecuación de la recta en forma explícita:

$$y = mx + n \tag{3.3}$$

Se recorre cada conjunto de puntos de la imagen donde se han detectado bordes y se calcula la ecuación de la recta que los une. En el plano (x,y) hay infinitas rectas que pasan por un punto dado, sin embargo, en el espacio de parámetros (m,n) hay una única recta que pasa por dicho punto. Sería como invertir los parámetros de la recta, donde en lugar de tratar de obtener y a partir de x, se trata de obtener n a partir de m.

La transformación anterior es útil de la siguiente manera, una vez se tiene una recta en el plano (m,n) se calcula la recta del posterior punto perteneciente al borde en el plano (m,n). La recta que une los dos puntos del plano (x,y) (que no son más que puntos de un borde detectado) es el punto de intersección de las rectas en el plano (m,n). Una vez que se encuentra dicha recta se disponen en una máscara de ceros (del mismo tamaño que la imagen evaluada) unos que permiten identificar la recta, dicha máscara se actualiza con cada recta encontrada.

Sin embargo el método anteriormente explicado falla en la identificación de líneas verticales, debido a que la pendiente de la recta es infinita. Es por ello que se hace un razonamiento análogo pero en lugar de usar coordenadas cartesianas se usan coordenadas polares, donde la recta se define como:

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{3.4}$$

En este nuevo caso el espacio al que se realiza la transformada es el plano (ρ, θ), donde en lugar de tratar con intersección de rectas se realiza con intersección de sinusoidales.

Programísticamente el porcedimiento es similar aunque ciertas funciones de *opencv* permiten la abstracción de la transformación y la intersección iterativa. Se comienza detectando los bordes de la imagen en blanco y negro a través de la función *Canny* que destaca los contornos de los objetos presentes en la imagen. La función

usada requiere de un ajuste en los parámetros low_threshold y high_threshold, los cuales controlan la sensibilidad a los bordes y permiten reducir el ruido en la imagen. La función devuelve una imagen binaria, donde los píxeles que forman los bordes tienen valor blanco (255) y el fondo es negro (0). A continuación se puede observar la extracción de bordes realizada con Canny:

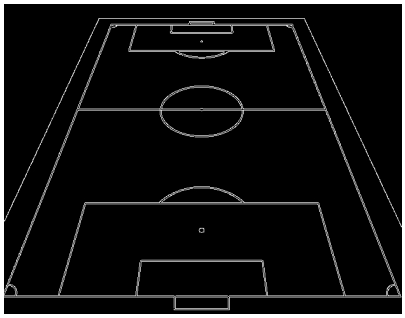

Figura 11: Extracción de bordes de football mediante Canny, autoría propia.

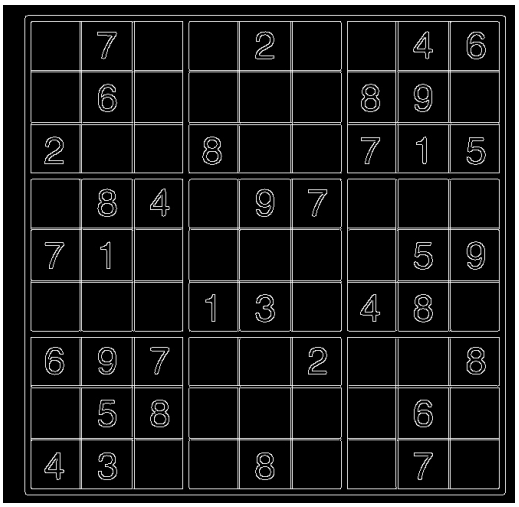

Figura 12: Extracción de bordes de sudoku mediante Canny, autoría propia.

Para identificar líneas en la imagen a partir de los bordes obtenidos, se utiliza la transformada de Hough mediante *HoughLinesP*, que detectará las líneas a partir de los bordes identificados. La función recibe los bordes, ρ que es un parámetro que permite identificar sinusoidales desde 0 hasta dicho valor (en este caso $\pi/180$), la longitud mínima de la línea y la máxima entre líneas así como un threshold. Este threshold permite que sólo se identifiquen las líneas que sean n veces coincidentes con el fin de reducir el posible ruido en la imagen. El resultado de la función es una lista que contiene los puntos iniciales y finales de cada recta detectada.

Tras obtener las líneas presentes en la imagen, se usa la función *line* que dibuja una recta dado una coordenada de inicio y de final así como un grosor y color para la recta. Cabe destacar que el proceso explicado anteriormente puede extraer tanto líneas rectas como curvas. Esto se comnsigue mediante el afinando del parámetro de mínima longitude de líneas así como el de máxima distancia entre ellas. Esto se debe al hecho de que dos píxeles alineados en una imagen forman una recta y dado que las imágenes a procesar están compuestas de píxeles y no de vectores (de momento imposibles en una imagen digital) son posibles de obtener.

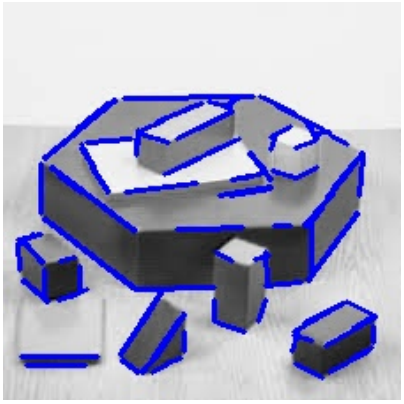

Figura 13: Extracción de líneas de geometry mediante Hough, autoría propia.

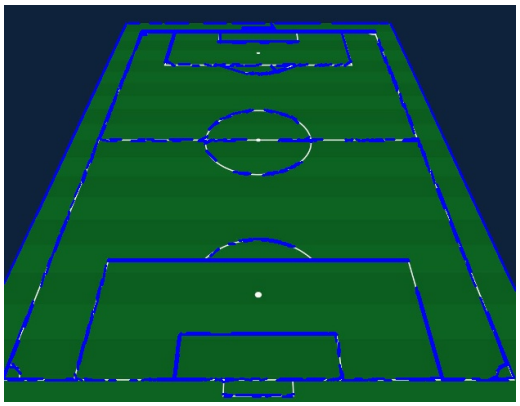

Figura 14: Extracción de líneas de football mediante Hough, autoría propia.

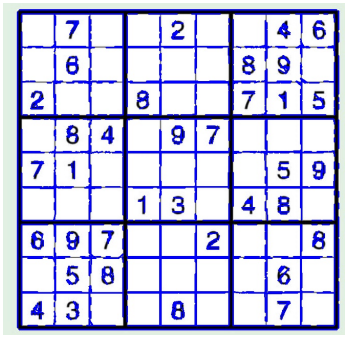

Figura 15: Extracción de líneas de sudoku mediante Hough, autoría propia.

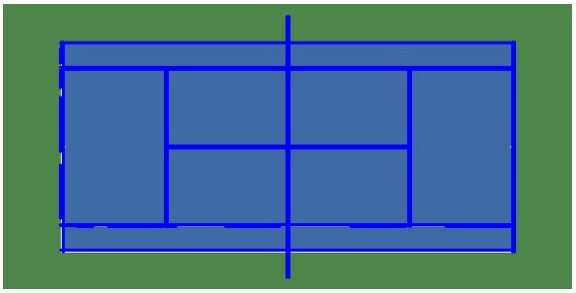

Figura 16: Extracción de líneas de tennis mediante Hough, autoría propia.

4. Estudio de puntos de interés

Un punto de interés es un píxel o conjunto de píxeles que destacan por su singularidad en la imagen. Estos puntos son útiles para la identificación de objetos en una imagen, ya que permiten la comparación de características entre distintas imágenes. A lo largo de la siguiente sección se va a explicar la extracción algo más manual de los puntos. Posteriormente se emplarán funciones de opencv para la extracción de puntos de interés de forma más eficiente y automática, con dichos puntos de interés se construirá una bolsa de palabras y se mostrará la eficiencia en la clasificación de la misma.

4.1. Extracción manual de puntos de interés

Para la extracción de puntos de interés se ha empleado la imagen geometry mientras que para una posterior verificación se ha usado una rotación de 90^{o} horarios de la misma. El proceso a replicar es el realizado por *SIFT* (Scale-Invariant Feature Transform), este comienza aplicando varios filtros gaussianos de distintas σ a la imagen. La utilidad de aplicar dichas máscaras es la de obtener distintas escalas de la imagen, ya que un punto de interés puede ser detectado en una escala y no en otra. Es decir, aplicando dichas máscaras se obtiene una imagen con distintos niveles de emborronado, el objetivo es que de cada uno de los niveles se pueda extraer una serie de puntos de interés. Esto es posible debido a la naturaleza del filtro gaussiano, que al difuminar las zonas, resalta los bordes relevantes (cambios de intensidad) de la imagen.

Las distintas escalas de emborronado se obtienen a través de *generateGaussianSigmas*, que recibe la σ inicial y el número de escalas a obtener, devolviendo una lista con las distintas σ a aplicar. Posteriormente se toma cada una de esos valores y se aplica un filtro gaussiano a la imagen original (que fue cargada y convertida a escala de grises) mediante la función *GaussianBlur*, dicha función requiere de una imagen y una σ para aplicar el filtro. Cabe destacar que se aplica una y otra vez la función sobre la imagen ya emborronada almacenándola en distintos estados, esto se hace para que las máscaras tengan un efecto de difuminado acumulativo.

Una vez que se poseen dichas imágenes, se diferencia cada imagen con la posterior, de esta manera, únicamente los puntos que permanecen después del emborronado quedan presentes en la imagen. Dichos puntos se almacenan en imágenes binarias, estas son producidas por la función *substract* the opencv. Dicha función requiere de las dos escalas, la actual y la posterior.

El siguiente paso consiste en la identificación de máximos o mínimos locales en

la imagen binaria, con este objetivo se ha desarrollado una función *isPixelAnExtremum*. Dicha función toma 3 cuadrículas de 3x3 píxeles, identifica el píxel central de la segunda cuadrícula y verifica que en valor absoluto es mayor a un threshold (proporcionado por el llamado de la función) y evalúa que dicho punto sea mayor, menor o igual a todos y cada uno de los píxeles de la cuadrícula central y las adyacentes. Si cumple con dichas condiciones se considera un punto de interés, ya que aquellos puntos que permanecen a lo largo de todas las escalas o que nunca aparecen (por efecto del emborronado) son puntos de interés.

Se deja el siguiente ejercicio de imaginación al lector:

- Imagine una toroide en un espacio tridimensional con un agujero no demasiado grande. Dicho agujero además es el origen de coordenadas.
- Ahora imagine que el toroide se emborrona gradualmente. Las fronteras se difuminan cada vez más, los bordes no son precisos, sin embargo puede seguir identificando el toroide. Tal vez ya no percibe su profundidad, pero es capaz de ver parcialmente su sombra y desde luego ve el agujero del centro.
- Al diferenciar las imágener emborronadas, los puntos que permanecen son los que definen el toroide, los que no, son los que se han difuminado.
- El origen de coordenadas o centro del toroide, al no haber variado nunca, está presente en todas las imágenes. Es por ello que cuando traza una cuadrícula de 3x3 píxeles alrededor de dicho punto y lo compara con el resto de imágenes diferenciadas, siempre será un máximo o un mínimo (según el emborronado y el color original).

Se espera que con el ejercicio anterior se haya podido comprender algo mejor el proceso de extracción enunciado con anterioridad. Se ha enunciado el proceso para extraer el punto de interés a partir de un conjunto de cuadrículas de 9 píxeles, sin embargo las imágenes tienen un tamaño considerablemente mayor. Para procesar los puntos de interés de la imagen se ha desarrollado la función *findScaleSpaceExtrema*. Esta función necesita las imágenes difuminadas, las imágenes diferenciadas, los distintos σ y un threshold para la identificación de los puntos de interés. La función recorre cada píxel del interior de la imagen (no toma los píxeles de los bordes para poder realizar la verificación de máximos y mínimos locales) y evalúa si dicho píxel es un punto de interés. En caso de serlo, se ajusta la localización del punto de interés en la imagen original con un ajuste cuadrático, se obtiene la orientación y finalmente se almacena para su posterior uso.

Finalmente se crean descriptores para cada punto de interés, estos son vectores que describen la orientación, localización y distribución de los píxeles alrededor del punto de interés. Dichos descriptores son útiles para la comparación e identificación

de puntos de interés y se obtienen a través de la función *generateDescriptors* a partir de los puntos de interés y de las escalas de emborronado.

Cabe destacar que todos los procesos y funciones anteriormente descritos fueron unidos en la función *computeKeypointsAndDescriptors* que sólo necesita la imagen original, el primer σ y el número de intervalos y devuelve los puntos de interés y sus descriptores. Para ejemplificar se usa la función *matchFeatures* que al recibir los descriptores de dos imágenes, compara los puntos de interés y devuelve una imagen con la unión de los mismos:

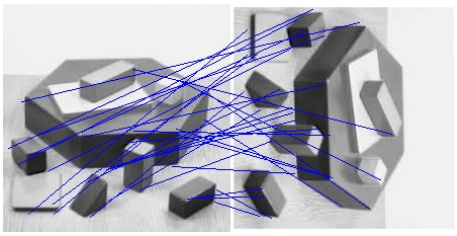

Figura 17: Verificación de puntos de interés y orientación manual, autoría propia.

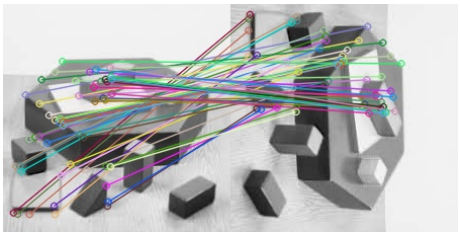

Figura 18: Verificación de puntos de interés y orientación con opencv, autoría propia.

Tal y como se puede observar (y como cabía esperar), la extracción manual de puntos de interés, aunque correcta, es un proceso lento y menos eficiente que el realizado por la implementación de *opencv* (la cual se explicará en la próxima sección). Se prefiere la implementación no manual no sólo por la eficiencia sino por la robustez y precisión de la misma obteniendo un mayor número de puntos de interés y descriptores.

4.2. Trabajo con bolsas de palabras visuales

Tal y como se ha enunciado a lo largo del escrito, una bolsa de palabras es un conjunto de puntos de descriptores que permiten la clasificación de imágenes y objetos a partir de comparaciones de características propias de cada imagen. Dado que en la anterior sección se identificó la eficiencia de la extracción de puntos de interés con *opencv* por encima de la extracción manual, se va a analizar dicha implementación para usarla en la creación de una bolsa de palabras.

El proceso comienza creando un extractor de descriptores, en la sección anterior se usó $SIFT_create$ que generará un extractor que sigue el algoritmo *SIFT* pero se pueden crear otros descriptores que se ajusten a las necesidades del problema.

Posteriormente se extraen los descriptores y puntos de interés de una imagen en escala de grises mediante el método *detectAndCompute* del extractor.

Explicado ya el proceso de extracción de *opencv* se ha desarrollado un fichero Python: *words_bag.py* que permite la creación de una bolsa de palabras visuales así como la clasificación de distintas imágenes todo ello con distintos parámetros y descriptores. Las pruebas se han realizado sobre las imagenes de un dataset con 2981 imágenes de entrenamiento y 1501 de test y se han recogido las 32 ejecuciones en el fichero anexo *results.csv*.

4.2.1. Creación de la bolsa de palabras visuales

Los pasos para la creación de la bolsa de palabras visuales son los siguientes:

- Creación del extractor de descriptores ya sea con $SIFT_create$ o con $KAZE_create$ (se quisieron evaluar más extractores pero los programas proporcionados de validación no estaban adaptados, cosa que se hará de cara al desarrollo del ÄirTrackPad¨)
- Extracción de los descriptores y puntos de interés de las imágenes en escala de grises de entrenamiento con el método *detectAndCompute* del extractor.
- Creación de un vocabulario con un tamaño determinado de vocabulario y un número de iteraciones, ya que se realiza un clustering a través de K-Means de los descriptores para obtener las palabras visuales.
- Guardado del vocabulario en un fichero $descriptor_parameters_vocabulary.pickle$ para su posterior uso.
- Cargado del vocabulario para el entrenamiento de un clasificador a través de la clase *Bow* y el método $load_vocabulary$.
- Entrenamiento de un clasificador de imágenes de la clase *ImageClassifier* con el vocabulario cargado y el método *train* (que recibe el training set). Este proceso devuelve el clasificador entrenado, este es guardado en una combinación del fichero $descriptor_parameters_labels.json$ y el fichero $descriptor_parameters_model.xml$ para su posterior uso.
- Inferencia en el conjunto de entrenamiento con el método *predict* del clasificador entrenado. Este método devuelve las etiquetas inferidas y la precisión de la clasificación.
- Inferencua en el conjunto de test con el método *predict* del clasificador entrenado. Este método devuelve las etiquetas inferidas y la precisión de la clasificación.

4.2.2. Muestreo de resultados

Se ha realizado un total de 32 pruebas con las siguientes modificaciones:

- Se ha variado el tamaño del vocabulario entre: 50, 100, 200 y 400.
- Se ha variado el número de iteraciones entre: 10, 20, 30 y 40.
- Se ha variado el extractor de descriptores entre *SIFT* y *KAZE*, cabe destacar que *SURF* y *AKAZE* (las versiones rápidas y optimizadas de los anteriores) no se pudieron probar debido a que fueron publicados y por tanto limitados al uso público. Del mismo modo queda pendiente adaptar los programas de entrenamiento y validación para probar *FAST*, *ORB* y *BRISK*.

Después de unas 15h de ejecución, se obtuvieron resultados concluyentes respecto a los parámetros a usar así como el extractor de descriptores óptimo:

- El extractor de puntos de interés *KAZE* es considerablemente más lento e ineficiente que *SURF* consiguiendo siempre menor precisión para los mismos parámetros, así mismo es más ineficiente temporalmente. Prácticamente en todo momento hay una diferencia de 0.09 en la precisión de la clasificación, en test y en train.
- El número de iteraciones óptimo en ambos extractores está entre 20 y 30, siendo 30 algo más común, sería interesante realizar ciertas pruebas con 25 y 35 iteraciones para ver si se obtiene una mejora en la precisión.
- Aparentemente cuanto mayor es el número de palabras en el vocabulario, mayor es la precisión en la clasificación, aumentar dicho valor aumenta casi exponencialmente el tiempo de creación de vocabulario. A pesar de ello, el tiempo de evaluación y entrenamiento aumenta de forma lineal (aparentemente) por lo que dado que el vocabulario sólo se genera una vez y se almacena, sería interesante aumentar todavía más el número de palabras en el vocabulario y observar si hay un aumento en la eficiencia de la clasificación, ya que el máximo valor se ha obtenido con 400 palabras en el vocabulario.

Por todo lo anterior se ha decidido que la mejor combinación de parámetros y extraxctores (a falta de realizar más pruebas es la siguiente):

- Tamaño del vocabulario: 400
- Número de iteraciones: 20
- Extractor de descriptores: *SIFT*

- Accuracy en train: 0.811
- Accuracy en test: 0.521
- Tiempo de ejecución de aproximado: 18 minutos

La mejor versión de *KAZE* ha sido:

- Tamaño del vocabulario: 400
- Número de iteraciones: 20
- Extractor de descriptores: *KAZE*
- Accuracy en train: 0.685
- Accuracy en test: 0.437
- Tiempo de ejecución de aproximado: 29 minutos

Del mismo modo se recomienda para futuras pruebas aumentar el conjunto de entrenamiento para tener una mayor precisión en el entrenamiento (tratando siempre de no realizar overfitting), realizar pruebas con otros extractores de descriptores, probar 25 y 35 iteraciones y aumentar el tamaño del vocabulario a 500 y 600 palabras para observar si hay una mejora en la precisión de la clasificación.