Isabel Manzaneque Núñez – 53902577F Imanzaneq3@alumno.uned.es

Visión Artificial

PEC 2 curso 2023-2024

Índice de contenido

[Segmentación con conocimiento del dominio 2](#_Toc153344794)

[1. Apartado a. 2](#_Toc153344795)

[2. Apartado b. 2](#_Toc153344796)

[3. Apartado c. 2](#_Toc153344797)

[4. Apartado d. 2](#_Toc153344798)

[5. Apartado e. 2](#_Toc153344799)

[6. Apartado f. 2](#_Toc153344800)

[Descriptores de puntos característicos 2](#_Toc153344801)

[**1.** Apartado a. 2](#_Toc153344802)

[2. Apartado b. 2](#_Toc153344803)

[3. Apartado c. 2](#_Toc153344804)

[4. Apartado d. 2](#_Toc153344805)

[5. Apartado e. 2](#_Toc153344806)

[Reconocimiento de objetos 2](#_Toc153344807)

[**1.** Apartado a. 2](#_Toc153344808)

[2. Apartado b. 2](#_Toc153344809)

[3. Apartado c. 2](#_Toc153344810)

[4. Apartado d. 2](#_Toc153344811)

[Aplicación de reconocimiento de objetos 3](#_Toc153344812)

[1. Apartado 4.1 3](#_Toc153344813)

[2. Apartado 4.2 3](#_Toc153344814)

# Segmentación con conocimiento del dominio

## Apartado a.

**cv2.HoughCircles** se utiliza para detectar círculos en imágenes. En la documentación de OpenCV podemos encontrar información sobre los parámetros que se utilizan para configurar la función:

* **image**: Imagen de entrada en escala de grises.
* **method**: Método de detección. Los métodos disponibles son HOUGH\_GRADIENT y HOUGH\_GRADIENT\_ALT.
* **dp**: Relación inversa de la resolución del acumulador. Por ejemplo, un valor de 1 significa que el acumulador tiene la misma resolución que la imagen de entrada, mientras que un valor de 2 significa que el acumulador tiene la mitad de ancho y altura.
* **minDist**: Distancia mínima entre los centros de los círculos detectados.
* **param1**: Umbral superior para el detector de bordes Canny.
* **param2**: Umbral para la detección de centros del círculo.
* **minRadius**: Radio mínimo del círculo a detectar.
* **maxRadius**: Radio máximo del círculo a detectar.

**cv2.HoughLines** se utiliza para detectar líneas en una imagen. En la documentación de OpenCV podemos encontrar también toda la información relativa a sus parámetros:

* **image**: Imagen de entrada, que debe ser una imagen binaria.
* **rho**: Resolución de la distancia del acumulador en píxeles.
* **theta**: Resolución del ángulo del acumulador en radianes.
* **threshold**: Umbral de votos mínimo para decidir si se trata de una línea. Como el número de votos depende del número de puntos de la línea, representa el largo mínimo de la línea a detectar.

**cv2.HoughLinesP** es una variante de cv2.HoughLines que utiliza la transformada de Hough probabilística. Cuenta con los parámetros de cv2.HoughLines y añade dos parámetros extra:

* **image**: Imagen de entrada, que debe ser una imagen binaria.
* **rho**: Resolución de la distancia del acumulador en píxeles.
* **theta**: Resolución del ángulo del acumulador en radianes.
* **threshold**: Umbral de votos mínimo para decidir si se trata de una línea. Como el número de votos depende del número de puntos de la línea, representa el largo mínimo de la línea a detectar.
* **minLineLength**: Longitud mínima de la línea. Las líneas más cortas que esto son rechazadas.
* **maxLineGap**: Máximo espacio permitido entre segmentos de línea para tratarlos como una sola línea.

## Apartado b.

La transformada de Hough es una técnica que se puede utilizar para detectar cualquier tipo de forma como elipses, polígonos y curvas arbitrarias.

La condición es que la forma que se desea detectar debe ser definible matemáticamente mediante una ecuación paramétrica. Por tanto, si tenemos conocimiento de la ecuación paramétrica del contorno que se está buscando, la transformada de Hough podrá detectar la forma incluso si presentan algo de discontinuidad o distorsión).

## Apartado c. X

Para resolver el apartado c se han realizado las siguientes funciones:

**drawHoughLines()**

Esta función aplica la transformada de Hough a la detección de la dirección del texto en la figura textoMolinos.png. Me he basado en el código del tutorial recomendado en el enunciado, el cual he adaptado a mis necesidades. La elección de los parámetros inicial ha sido la siguiente:

* **rho**: 1
* **theta**: pi/180
* **threshold**: 228

A screenshot of a computer screen

Description automatically generated

Con estos parámetros, se encuentra un total de 76 líneas. Como se había indicado antes, el threshold representa el largo mínimo de la línea a detectar. Parece que las líneas que no se han detectado tienen como característica el ser más cortas que las que sí se han detectado. Sin embargo, reducir el threshold a un valor inferior a 228 hace que se generen líneas ruidosas que no se corresponden con el texto.

Si cambiamos la resolución angular a pi/200, se detectarán un total de 92 líneas incluyendo la primera (que estaba siendo ignorada anteriormente). Esto puede deberse a que exista una ligera variación angular entre esta línea y el resto, por lo que una precisión angular más fina sería suficiente para detectarla. Hay que tener en cuenta que una resolución más fina puede también resultar en más falsos positivos, detectando líneas debido al ruido o variaciones menores en la imagen.

A screenshot of a computer screen

Description automatically generated

El valor de theta devuelto por cv2.HoughLines() es la inclinación de la línea respecto a la horizontal en radianes. Se puede utilizar para calcular la media de la inclinación en grados y así realizar una transformación afín para que las líneas del texto queden en horizontal. Esta inclinación media se le va a pasar a una función que rotará la imagen.

**rotate(meanAngle)**

Esta función recibe un ángulo “meanAngle” y rota la imagen esos grados en torno a su centro

A screenshot of a computer

Description automatically generated

**segment(rotatedImg)**

Esta función va a segmentar el texto de la imagen rotada. Comienza realizando una umbralización binaria para transformar la imagen rotada a una imagen en blanco y negro.

Se realizara una detección de bordes y contornos en la imagen, lo cual también detectará los bordes de la bounding box, lo cual queremos evitar. Tras realizar un filtrado de los contornos detectados, para eliminar aquellos mayores a un cierto umbral, el resultado de la segmentación es el siguiente:

## Apartado d.

RANSAC (Random Sample Consensus) es un algoritmo iterativo utilizado para estimar los parámetros de un modelo matemático a partir de un conjunto de datos que contiene valores atípicos. Es útil en el contexto de regresión lineal o ajuste de modelos en situaciones donde los datos pueden contener valores ruidosos y atípicos. Los parámetros a tener en cuenta en este método son:

* **Datos**: un conjunto de datos observados
* **Modelo**: un modelo que puede ser ajustado a los puntos de datos. Esto podría ser una regresión lineal, un ajuste de curva, o cualquier otro modelo matemático apropiado.
* **Valores de ajuste mínimos (n)** : el número mínimo de valores requeridos para ajustar el modelo
* **Número de iteraciones (k)** : el número máximo de iteraciones permitidas. Un número mayor de iteraciones aumenta la probabilidad de encontrar un conjunto de datos libre de valores atípicos.
* **Umbral (t)** : valor umbral para determinar cuándo un dato se ajusta a un modelo. Los puntos que caen dentro de esta distancia del modelo estimado se consideran parte del consenso, mientras que los que están fuera se consideran valores atípicos.
* **Valores cercanos requeridos (d)** : el número de valores de datos cercanos requeridos para afirmar que un modelo se ajusta bien a los datos

El método devolverá los parámetros del modelo que mejor se ajustan a los datos (o nulo si no se encuentra un buen modelo)

## Apartado e.

Para obtener la dirección de la línea más larga de la imagen, me he basado en el código del tutorial que se nos proporciona en enunciado. He realizado ciertos cambios en el código para adaptarlo al que he realizado algunos cambios para adaptarlo al caso que se está tratando:

El código del ejemplo trabaja con datos sintéticos, mientras que en este caso se debe trabajar con datos reales que existen en una imagen. Por esto, hay que iterar sobre la imagen original para contar extraer sus datos y las coordenadas de su localización.

Como los datos son reales, no se pueden controlar explícitamente características como el ruido y los outliers.

Además, el código del ejemplo compara regresión lineal y RANSAC. Como esta comparación no es necesaria en el ejercicio, se utiliza únicamente el algoritmo RANSAC y se elimina la regresión lineal.

Algo que me ha sorprendido ver en un principio es que al modelo RANSAC no se le pasa ningún parámetro en el ejemplo, lo cual me ha resultado extraño debido a que en el ejercicio anterior hablábamos de los parámetros que necesitaba el algoritmo. Tras una visita rápida a la documentación, he encontrado lo siguiente:

*class*sklearn.linear\_model.**RANSACRegressor**(*estimator=None*, *\**, *min\_samples=None*, *residual\_threshold=None*, *is\_data\_valid=None*, *is\_model\_valid=None*, *max\_trials=100*, *max\_skips=inf*, *stop\_n\_inliers=inf*, *stop\_score=inf*, *stop\_probability=0.99*, *loss='absolute\_error'*, *random\_state=None*)

Si no se especifica ningún parámetro, se usarán los parámetros por defecto que proporciona scikit learn. Estos parámetros por defecto parecen funcionar bien para nuestro caso, cuyo resultado se puede ver a continuación:

A chart with green and yellow dots

Description automatically generated

## Apartado f.

La presentación proporciona una comparación entre la Transformada de Hough y el algoritmo RANSAC para el ajuste de modelos en la presencia de ruido y datos atípicos.

Se detalla como la **Transformada de Hough** es un método clásico para la detección de formas que utiliza una estrategia de votación en el espacio de parámetros. Funciona bien para detectar líneas, pero puede ser difícil generalizar a dimensiones más altas y a formas más complejas.

Por su parte, **RANSAC** ofrece un enfoque robusto para el ajuste de modelos que puede manejar bien una gran cantidad de outliers. RANSAC selecciona aleatoriamente subconjuntos de puntos, ajusta modelos a estos subconjuntos y determina el mejor modelo basándose en la cantidad de inliers. Su flexibilidad hace que se pueda aplicar a una gran variedad de problemas, pero también tiene inconvenientes, como la gran cantidad de parámetros a tener en cuenta.

Ambos métodos tienen sus ventajas y desventajas. La elección entre ellos depende de las características específicas del problema y del conjunto de datos con el que se trabaja: la Transformada de Hough es más adecuada para detectar formas simples en entornos de baja dimensión, mientras que RANSAC es más flexible y puede manejar mejor los datos con un alto porcentaje de outliers.

# Descriptores de puntos característicos

## Apartado a.

Para transformar los formularios rellenos y visualizarlos en la misma escala y orientación que el patrón, he realizado las siguientes funciones:

**orbMatcher(img1, img2)**

Función inspirada en el tutorial “Feature matching”, recomendado en el enunciado. Esta función recibe 2 imágenes como parámetros y utiliza las características ORB para encontrar la relación entre ambas.

Al terminar, devuelve la imagen resultado, los puntos característicos de ambas imágenes y la relación entre ellos.

**affineTransform(img1, img2, matches, kp1, kp2)**

Función que recibe como parámetro las imágenes a comparar, sus puntos característicos y la relación entre ellos. Con esta información, realiza una transformación afín sobre la imagen2 para que tenga la misma orientación y escala que la imagen1.

Podemos obtener la matriz de transformación que pasa de la escala y orientación de la segunda imagen a las de la primera gracias a la función getAffineTransform de OpenCV. Esta función toma 3 puntos de referencia de ambas imágenes y mapea los puntos de la primera a los de la segunda.

Al terminar, devuelve la imagen resultado.

A continuación, podemos ver los efectos de utilizar ambas funciones sobre los pares formulario.png - formularioRelleno-original.png y formulario.png - formularioRelleno-rotado.png:

**formulario.png - formularioRelleno-original.png**

En la imagen resultado que devuelve la función orbMatcher, se pueden ver las 20 primeras coincidencias encontradas. Algunas de estas son correctas, aunque también hay coincidencias incorrectas (4 si no me equivoco, por ejemplo, las que salen del 1 en “Test 1” de la imagen formulario.png).

A screenshot of a computer

Description automatically generated

Ya se ha establecido que para realizar la transformación afín necesitamos tres puntos de cada imagen. Estos puntos se pueden obtener de las coincidencias encontradas, pero hay algo que debemos tener en cuenta: si la coincidencia no es correcta, la transformación tampoco lo será. Es más, aunque la coincidencia sea correcta, las transformaciones afines pueden variar enormemente dependiendo de los puntos escogidos como referencia. Es por este motivo que decidí implementar un bucle que recorriera el array de coincidencias y probara todas las transformaciones posibles para intervalos de 3 coincidencias. Esto no va a estar en el código que voy a entregar para el ejercicio ya que se mostraban más de 170 transformaciones candidatas, pero adjunto aquí la implementación:

A computer screen shot of a program

Description automatically generated

Tras revisar todas las transformaciones, llegué a la conclusión de que las mejores eran las obtenidas con las coincidencias 15:18, 23:26, 24:27, 59:62, 66:69, 71:74, todas ellas prácticamente iguales. La imagen que he escogido corresponde a la transformación generada con las coincidencias 66-69:

Screenshot of a screenshot of a test

Description automatically generated

No conseguí encontrar ninguna transformación en la que no se perdiera la pequeña sección superior derecha. Esto me parece razonable ya que, al rotar la imagen en sentido horario, el encuadre de la imagen con su bounding box va a verse afectado.

A continuación, adjunto un ejemplo (por curiosidad) de una transformación realizada tomando como puntos de referencia una coincidencia incorrecta (5:8):

A screenshot of a computer

Description automatically generated

**formulario.png - formularioRelleno-rotado.png**

Estos son los resultados obtenidos de la función orbMatcher para el segundo par de imágenes.

Para las 20 coincidencias que se muestran en la imagen, parece que la función ha sido algo más certera en este caso. Parece que todas las coincidencias han sido correctas (si no me engaña la vista). Esto puede ser debido a que las orientaciones y resoluciones de las imágenes son más similares en este caso.

**A screenshot of a test

Description automatically generated**

Al igual que en el anterior caso, he recorrido todos los posibles candidatos de transformación afín. Las transformaciones basadas en coincidencias correctas comparten la característica de que la escala ha sido reducida, quedando la bounding box más grande que la imagen. A continuación, el candidato 73:76:

**Screenshot of a computer screen with a picture of a fish

Description automatically generated**

Podemos observar el posible motivo de esto al comparar ambas imágenes en el visualizador de Windows:

A paper with a picture of a fish

Description automatically generated with medium confidence

El formulario relleno parece tener un tamaño de letra ligeramente mayor al del formulario original. Al mismo tiempo, si nos fijamos en la distancia de los bordes de las imágenes a las letras, da la impresión de que el formulario relleno está recortado dejando un margen blanco más estrecho.

Por estos motivos, una transformación afín para igualar la orientación y la escala del formulario relleno al original reduciría la escala del formulario relleno y esto haría que su bounding box fuera más grande que la imagen.

## Apartado b.

Para realizar este apartado he seguido el mismo procedimiento que en el apartado a, realizando una función siftMatcher y pasándole el resultado a affineTransform, definida antes.

**siftMatcher(img1, img2)**

Función inspirada en el tutorial “Feature matching”, recomendado en el enunciado. Esta función recibe 2 imágenes como parámetros y utiliza las características SIFT para encontrar la relación entre ambas.

Al terminar, devuelve la imagen resultado, los puntos característicos de ambas imágenes y la relación entre ellos.

A continuación, podemos ver los efectos de utilizar ambas funciones sobre los pares formulario.png - formularioRelleno-original.png y formulario.png - formularioRelleno-rotado.png:

**formulario.png - formularioRelleno-original.png**

La imagen a continuación muestra las 20 primeras coincidencias obtenidas al aplicar la función siftMatcher sobre las imágenes. En este caso (y si no me engaña la vista) parece que hay 3 coincidencias incorrectas. Esto es una coincidencia incorrecta menos que lo que se obtuvo utilizando ORB, pero no se puede decir que la mejoría haya sido muy grande.

**A screenshot of a computer

Description automatically generated**

En cuanto a la transformación afín, se ha seguido la misma estrategia de recorrer todas las transformaciones utilizando un bucle. El candidato 105:108 fue una de las mejores transformaciones encontradas y es prácticamente igual al que se seleccionó utilizando ORB.

**A screenshot of a computer

Description automatically generated**

Parece que en este par de imágenes la diferencia en los resultados obtenidos no ha sido muy grande utilizando las ORB y SIFT.

**formulario.png - formularioRelleno-rotado.png**

Al igual que en el caso anterior, mostramos las 20 primeras coincidencias. Parece que todas ellas son correctas, al igual que lo fueron con ORB para este par de imágenes.

**A screenshot of a test

Description automatically generated**

A la hora de realizar la transformación, el candidato 241:244 es el que más se parece al formulario sin rellenar en términos de orientación y escala. Al igual que en el caso con ORB, se ha tenido que reducir ligeramente la escala del formulario relleno, lo que hace que la imagen sea más pequeña que su bounding box.

**A screenshot of a test

Description automatically generated**

Como conclusión, en términos de resultados ambas técnicas han estado bastante ajustadas. Parece que SIFT es algo más precisa cuando se trata de imágenes rotadas y escaladas, aunque ORB no ha quedado muy atrás.

Si esta robustez de SIFT se hace más evidente al explorar más coincidencias, sería una buena idea utilizar SIFT en casos en los que se requiera una mayor precisión o las imágenes estén rotadas y escaladas. Como SIFT es computacionalmente más costoso que ORB, podría utilizarse ORB en aquellos casos en los que las imágenes tengan orientaciones y escalas similares o en casos en los que no se requiera tanta precisión.

## Apartado c.

## Apartado d.

## Apartado e.

# Reconocimiento de objetos

## Apartado a.

## Apartado b.

## Apartado c.

## Apartado d.

# Aplicación de reconocimiento de objetos

## Apartado 4.1

## Apartado 4.2