Análisis de imágenes - Memoria

Robótica y percepción computacional

Alejandro Cobo Cabornero, 150333
Facundo Navarro Olivera, 140213
Diego Sánchez Lizuain, 150072

Contenido

1. Algoritmo de segmentación de imágenes

1.1. Algoritmo de segmentación

Para la segmentación de imágenes inicialmente se habían considerado varios clasificadores, pero finalmente se ha utilizado el clasificador *Quadratic Discriminant Analysis (QDA)* que ofrece la librería *SKLearn*, debido a que se han obtenido mejores resultados con un tiempo de ejecución aceptable

En la fase de entrenamiento, para cada clase, el clasificador almacena la media y las matrices de covarianza.

En la fase de clasificación, asigna una frontera de decisión cuadrática a las clases y supone una distribución gaussiana de los datos de entrada, de modo que emplea el teorema de Bayes para calcular la probabilidad condicionada con las clases.

1.2. Código

Como se ha usado una librería externa, el código del algoritmo en el fichero *classif.*py es muy simple, simplemente se llama al método *fit* en la fase de entrenamiento y al método *predict* en la fase de clasificación.

Para adaptar la imagen (de 3 dimensiones) a los datos de entrada del clasificador (de 2 dimensiones) se ajustan los datos de la siguiente forma:

- Primero se normaliza la imagen para reducir el ruido.

```
imNp = np.rollaxis((np.rollaxis(imNp,2)+0.0)/np.sum(imNp,2),0,3)[:,:,:2]
```

- Luego se pasa un filtro gaussiano para reducir más el ruido.

```
imNp = cv2.GaussianBlur(imNp, (0,0), 1)
```

- Se reajustan las dimensiones:

```
im2D = np.reshape(imNp, (imNp.shape[0]*imNp.shape[1],imNp.shape[2]))
```

- Finalmente, se segmenta la imagen y se ajustan las dimensiones de la salida del clasificador para que tengan las mismas dimensiones que la imagen original.

```
labels_seg = np.reshape(seg.segmenta(im2D), (imDraw.shape[0], imDraw.shape[1]))
```

1.3. Tiempos de ejecución

El tiempo de entrenamiento suele ser de unos 100 ms.

El tiempo de segmentación de cada imagen está en torno a los 45 ms. Esto nos permite segmentar una de cada dos imágenes suponiendo una tasa 24 fotogramas por segundo.

2. Algoritmo de análisis de imágenes

2.1. Suposiciones/simplificaciones del problema

- Las líneas son infinitas (nunca se cortan en medio de la imagen).
- En una imagen no se pueden ver segmentos futuros o pasados del trazado.
- En la línea, los píxeles de la entrada serán los que tengan mayor coordenada Y en la imagen (los que estén más abajo). En caso de empate, será el conjunto de píxeles más cercanos al centro.
- En un cruce no habrá ninguna marca aparte de la flecha.
- No habrá más de dos contornos convexos que se correspondan con la línea.

2.2 Algoritmo de análisis de imágenes

Para identificar si existe un cruce o no, se hallan los contornos del fondo y, según el número de contornos. Si el número de contornos es mayor a 2, existe un cruce o una bifurcación.

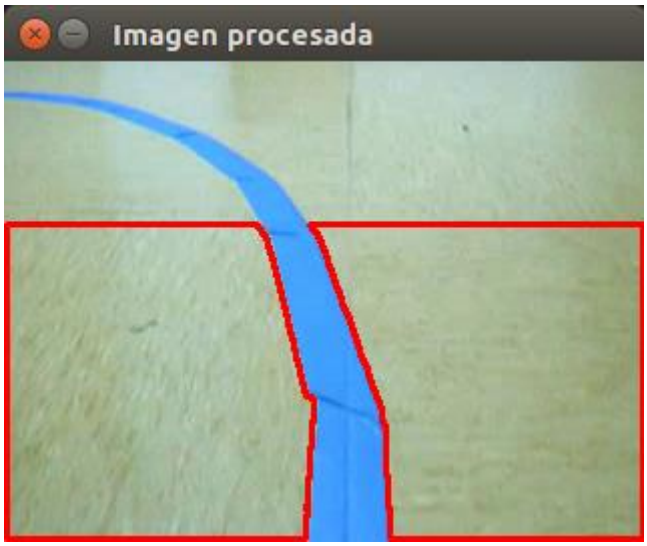

Ilustración 1: contornos del fondo en una línea sin cruces.

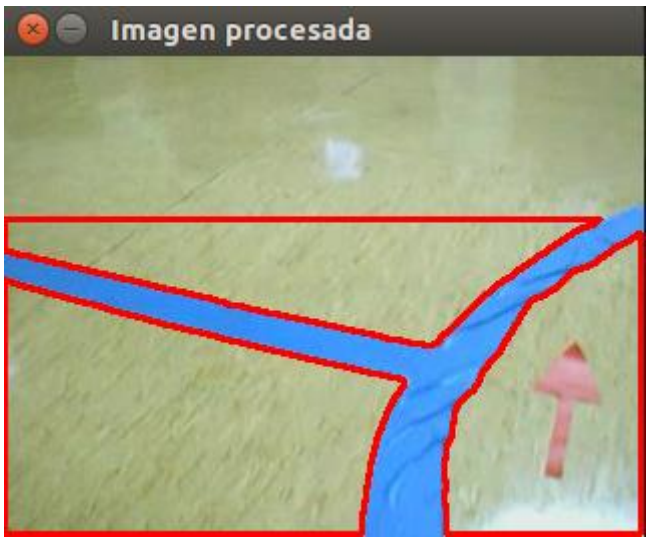

Ilustración 2: contornos del fondo en un cruce.

En caso de que exista un cruce, hay que procesar la dirección y orientación de la flecha. Para ello, se inscribe una elipse en la flecha y se utilizan dos puntos del eje mayor para definir una recta. Esa recta tocará los bordes de la imagen en dos puntos.

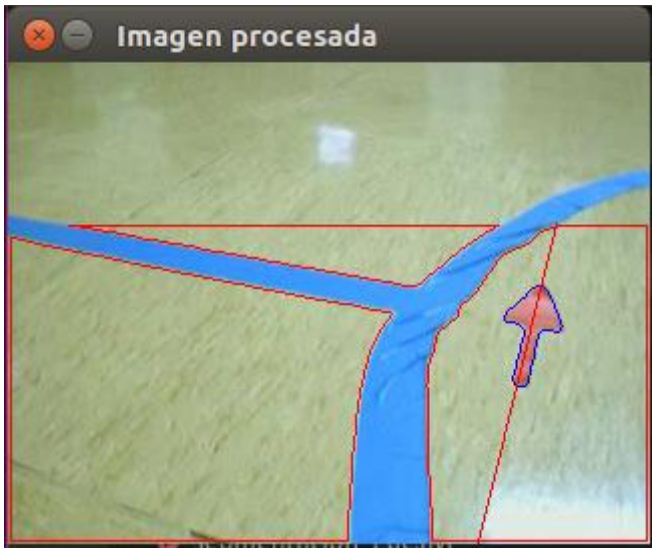

Ilustración 3: ejemplo de la recta que define la elipse inscrita en la flecha.

Para saber cuál de esos dos puntos es el punto de salida, se divide la flecha en dos partes usando el eje menor de la elipse anteriormente calculada y se determina la mitad con mayor área. Aquella mitad de la flecha con mayor área determinará el punto de salida.

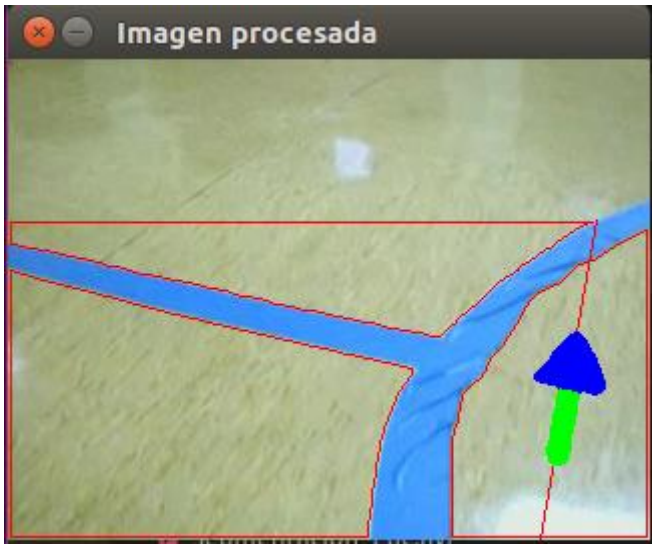

Ilustración 4: visualización de las dos mitades de la flecha. El área azul es mayor y, por tanto, se tomará como punto de salida el que está situado más arriba.

Finalmente, se determinan los extremos de la línea azul según los píxeles del contorno que están en los bordes de la imagen.

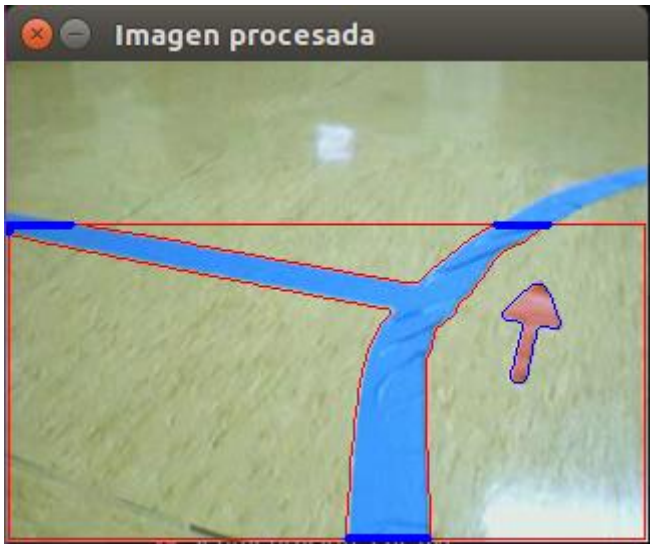

Ilustración 5: píxeles del contorno de la línea que están sobre los bordes de la imagen.

De entre estos conjuntos de píxeles, aquel con mayor coordenada Y (más abajo en la imagen) será el que represente la entrada. En caso de empate, se elegirá el que esté más cerca al centro del borde inferior de la imagen.

En una línea recta, la salida será el conjunto que no sea el de entrada y, en un cruce, el conjunto que represente la salida de la línea en la escena será el que esté más cerca del punto de salida calculado anteriormente al procesar la flecha.

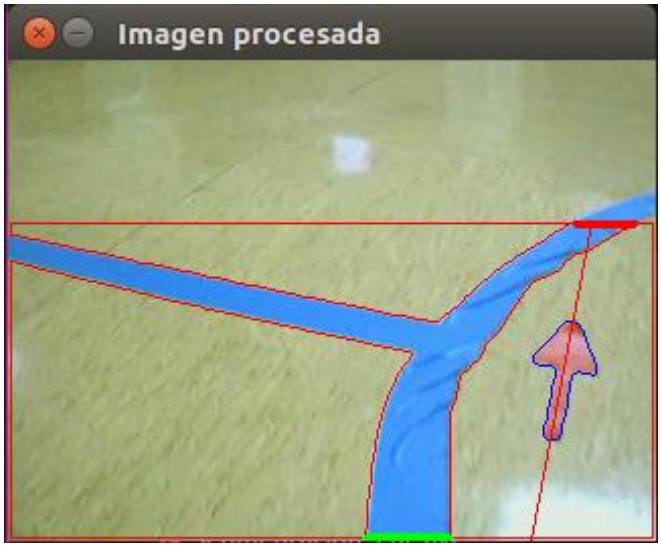

Ilustración 6: entrada (verde) y salida (rojo) de la escena.

Hay que considerar que habrá momentos en los que el robot esté en un cruce y aún no pueda ver la flecha porque no se ha acercado demasiado, por lo que no habrá conjunto de salida. En este caso, simplemente se le ordenará al robot que siga con la consigna que tenía anteriormente y, eventualmente, encontrará la flecha.

Estamos investigando una forma más robusta de resolver este problema intentando que compare las salidas con la última salida y seleccione aquella más cercana, pero

debemos trabajar los casos en los que no lo hace correctamente (como en un cruce en T).

2.3. Código

Se ha separado el código correspondiente al algoritmo de análisis de imágenes para una mayor claridad. El código se encuentra en el archivo *analisis.py* y consta de las siguientes funciones:

- *esCruce*: determina si el robot está en un cruce contando el número de contornos del fondo. Para mayor robustez, se cuentan también los píxeles pertenecientes a marcas y flechas como si fueran del fondo.
- *get_pSalida*: devuelve el punto de salida determinado por la flecha. Para ello, halla los puntos del borde de la imagen dentro de la recta definida por los puntos azules mostrados en la *Ilustración 7*. Una vez hecho eso, estima la orientación calculando el número de puntos a cada lado de la flecha usando el área signada del triángulo definido por los dos puntos en verde de la *Ilustración 7* con cada punto de la flecha. El área signada de los puntos de una mitad tendrá un signo y los de la otra mitad, el signo contrario.
    - Para mayor robustez y, dado que habrá momentos en los que el robot solo pueda ver una parte de la flecha y el cálculo del área sea incorrecto, se guarda el último punto de salida calculado y, si la diferencia entre el último y el actual es muy grande, se considera un error y se sigue usando el último.

Ilustración 7: puntos que definen la dirección de la flecha (azul) y los que definen un segmento que divide la flecha en dos (verde).

- *get_bordes*: devuelve los píxeles del contorno de la línea que están en los bordes de la imagen. La salida es una lista con cada borde representado por una lista de puntos. Recorre la lista de puntos de los contornos y añade los puntos consecutivos que estén en los bordes de la imagen a una lista, de modo que genera una lista por cada conjunto de puntos.
    - Existe un caso en el que un mismo borde que abarca la esquina superior izquierda de la imagen es considerado como dos, porque es

cuando termina la lista de contornos. En este caso, se juntan los dos bordes en uno teniendo en cuenta que hay que juntar los puntos del principio del contorno con los del final.

- *get_entrada*: dada una lista de bordes, devuelve el índice de aquel que representa la entrada. Será aquel con mayor coordenada Y o, en caso de empate, el que esté más cerca del punto medio del borde inferior de la imagen.
- *get_salida*: dada una lista de bordes, devuelve el índice de aquel que represente una salida. Si solo hay dos bordes, será el que no sea la entrada. Si hay más bordes, será el que esté más cerca del punto de salida indicado en sus parámetros de entrada.

En la librería *geometry.py* existen dos funciones auxiliares:

- *dist*: devuelve el cuadrado de la distancia euclídea entre dos puntos.
- *sarea*: devuelve el área signada del triángulo definido por 3 puntos. Si al seguir los puntos se sigue un sentido horario (como en la *Ilustración 8*), el área será negativa, y positiva en el caso contrario.
    - No obstante, como en la imagen las coordenadas Y están "invertidas" (a mayor Y, más abajo en la imagen), en este caso es al revés: el área signada será positiva si se sigue un sentido horario y negativa en otro caso.

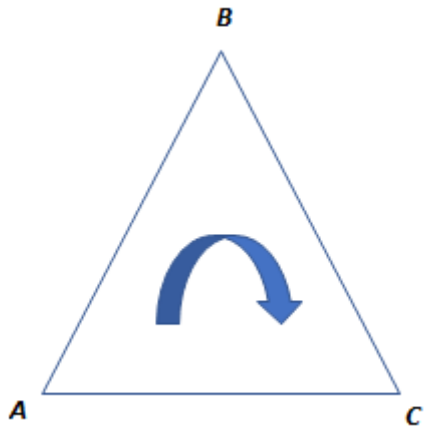

Ilustración 8:el triángulo ABC tendrá un área signada negativa.