Reconocimiento de Iconos

Robótica y Percepción Computacional

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

UNIVERSIDAD
POLITÉCNICA
DE MADRID

Grado en Ingeniería Informática
Universidad Politécnica de Madrid
3 de mayo de 2019

Índice

1. Introducción

En este informe se detalla el proceso de creación de una base de datos de imágenes de iconos y los resultados de la evaluación del rendimiento de 2 clasificadores sobre esta base de datos mediante una prueba estática y otra dinámica.

El problema a resolver es el reconocimiento de varios iconos de forma automática mediante el uso de algoritmos clasificadores que estiman las clases de los datos a partir de varios atributos.

La base de datos consiste en 100 imágenes binarizadas por cada icono. La binarización de las imágenes se obtiene utilizando el módulo de segmentación previamente desarrollado en la práctica.

A partir de estas imágenes, se obtienen los momentos de Hu [1], que permiten la descripción de objetos y son invariantes ante cambios de traslación, rotación y escala.

Los clasificadores que se han evaluado son el *k-NN* (k vecinos más próximos) con $k = 1$ y el clasificador basado en la distancia de Mahalanobis.

Para la comparación de los clasificadores se ha realizado un test estático sobre la base de datos empleando la técnica de validación cruzada *leave-one-out*.

Por último, se ha realizado un test dinámico con un vídeo de prueba grabado con el robot en el que aparecen varios iconos y el clasificador deberá identificar correctamente cada uno.

2. Base de datos

La base de datos constiste en 100 imágenes por cada uno de los 4 iconos, siendo 400 imágenes en total. La Figura 1 muestra los 4 tipos de iconos que formarán parte de la base de datos.

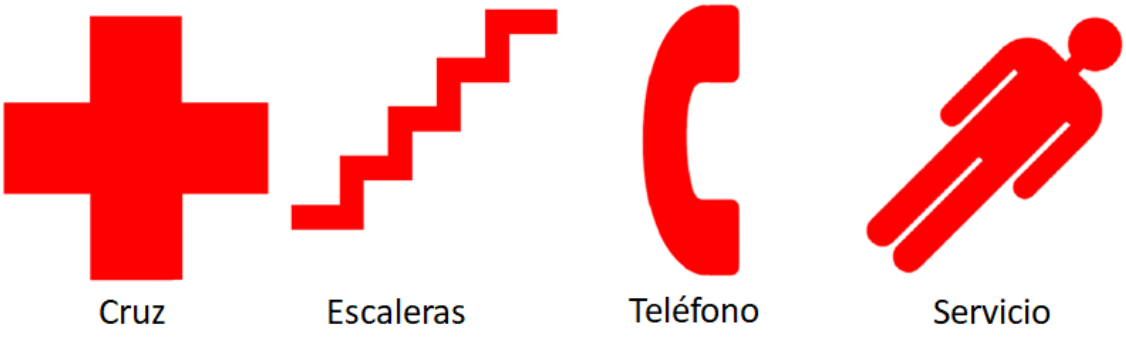


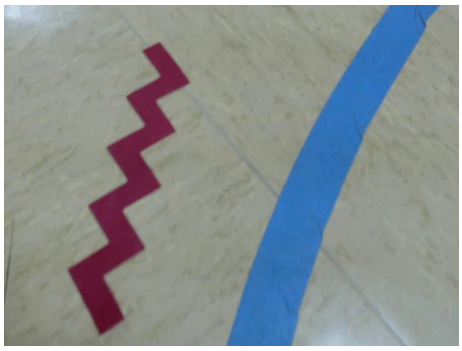
Figura 1: Iconos de la base de datos. Debajo de cada icono se muestra la clase asignada.

2.1. Obtención de imágenes

Para la obtención de las imágenes se ha utilizado la cámara del robot para grabar varios vídeos de cada icono en los que se va cambaindo la perspectiva de la imagen y la distancia de la cámara al icono para obtener la mayor variedad posible de casos.

En la figura 2 se muestran varios *frames* de los vídeos tomados.

(a) Cruz



(b) Escaleras

(c) Servicio

(d) Teléfono

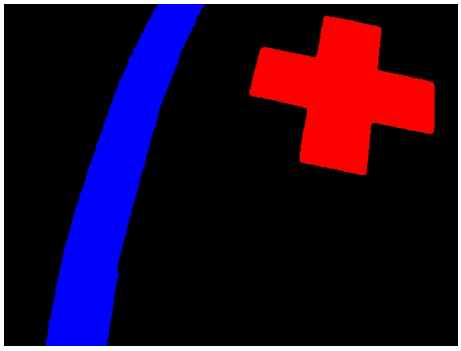
Figura 2: Ejemplos de *frames* de los vídeos tomados.

Una vez grabados los vídeos, se han seleccionado los *frames* de mayor interés para formar parte de la base de datos.

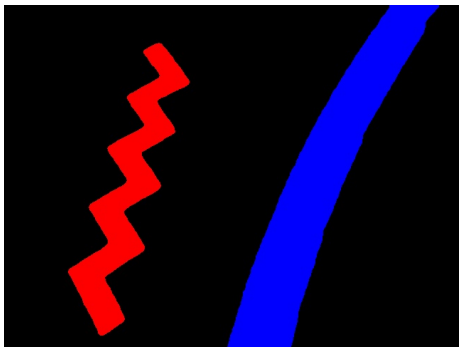
2.2. Binarización de imágenes

La binarización de las imágenes se ha realizado utilizando el módulo de segmentación desarrollado anteriormente en la práctica, que emplea el clasificador *QDA* de la librería *Scikit-learn*.

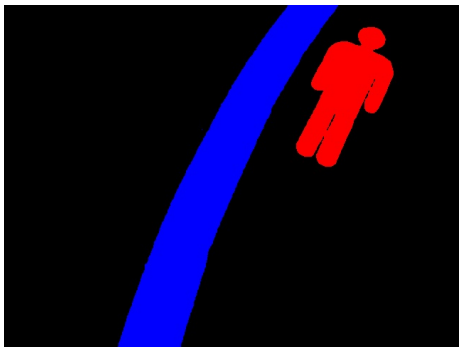
Primero, se segmenta la imagen y se separan fondo, marcas y línea. En la figura 3 se muestra un ejemplo de esta segmentación.



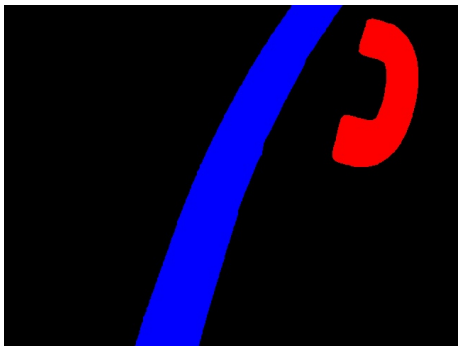
(a) Cruz



(b) Escaleras



(c) Servicio



(d) Teléfono

Figura 3: Segmentación de los *frames*.

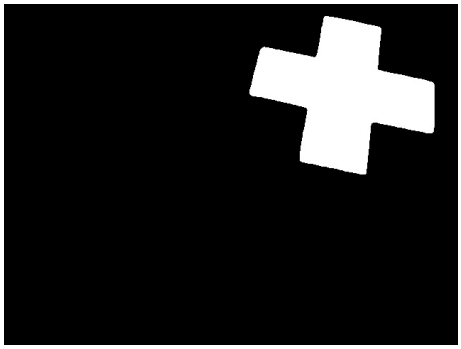
Una vez segmentadas las imágenes, se seleccionan los píxeles segmentados como "marcas" (en rojo) y se hallan los cierres convexos con *OpenCV*. Como la segmentación puede no ser perfecta, es posible que se clasifiquen erróneamente píxeles del fondo o la línea, por lo que se seleccionan los píxeles del cierre convexo con mayor área.

La función que binariza una imagen es:

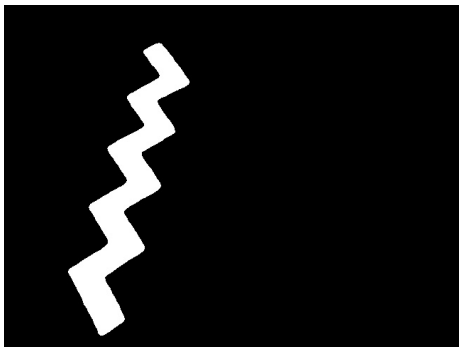
```
# Binariza una imagen pintando de blanco los pixeles del icono y en negro
# todo lo demas
```

```
def binarize ( labels_seg ):
    # Creo una imagen en negro
    img = np.zeros (( labels_seg .shape[0], labels_seg .shape[1]) )
    # Hallo los pixeles del icono
    mark = ( labels_seg ==2).astype(np.uint8 )*255
    # Hallo los cierres convexos
    contList , hier = cv2.findContours(mark,cv2.RETR_LIST,cv2.
        CHAIN_APPROX_NONE)
    if len( contList ) > 0:
            # Elijo el cierre con mayor area
            cont = contList .index( max(contList, key=lambda x : cv2.
                contourArea(x[0]) ) )
            # Pinto el cierre elegido
            cv2.drawContours(img, contList , cont, (255,255,255) ,cv2.cv.
                CV_FILLED)
            return img, contList [cont]
    else :
            return img, None
```

En la figura 4 se muestran las imágenes binarizadas.



(a) Cruz



(b) Escaleras

(c) Servicio

(d) Teléfono

Figura 4: Binarización de las imágenes segmentadas.

Las imágenes binarizadas serán las que finalmente formen la base de datos.

2.3. Variables discriminantes

Las variables discriminantes que se usarán para el proceso de clasificación son los momentos de Hu de las imágenes binarizadas. Esto permite describir cada icono de una forma invariante a transformaciones de traslación, rotación y escala.

Para cada imagen en la base de datos se calculan sus momentos de Hu y se forma una matriz de datos que usarán los clasificadores.

El cálculo de los momentos de Hu para una imagen se muestra a continuación:

```
# Convierte una imagen binarizada a momentos de Hu
def get_hu(img):
    moments = cv2.moments(img, True)
    return cv2.HuMoments(moments).T
```

3. Clasificadores

3.1. 1-NN

Este clasificador es la variante de $k = 1$ del clasificador *k-NN*, que compara cada dato con las muestras de la base de datos y elige la clase de aquella más cercana.

Se ha utilizado la implementación de la biblioteca *Scikit-learn*, con los siguientes parámetros:

- Número de vecinos (*neighbors*): 1.
- Algoritmo de búsqueda (*algorithm*): fuerza bruta.
- Distancia (*metric*): euclídea.

De modo que calcula el vecino más próximo usando la distancia euclídea, comparando cada dato de entrada con todos los datos de entrenamiento.

3.2. Distancia de Mahalanobis

EL clasificador basado en la distancia de Mahalanobis se ha implementado parcialmente, dejando el cálculo de la distancia a la biblioteca *Scipy*.

En la fase de entrenamiento se calculan las medias y las matrices de covarianza de las clases.

```
def fit(self,x,y):
    self.labels = np.unique(y)
    self.means = np.zeros((len(self.labels), x.shape[1]))
    self.invcov = np.empty((x.shape[1], x.shape[1], len(self.labels)))
    for i in xrange(len(self.labels)):
        self.means[i] = np.mean(x[y == self.labels[i]], axis=0)
        self.invcov[:,:,i] = np.linalg.inv(np.cov(x[y==self.labels[i]],
            rowvar=False))
    return self
```

Para clasificar los datos, se calula la distancia de Mahalanobis con cada clase y se selecciona aquella que sea menor.

```
def predict(self,x):
    distances = np.empty((x.shape[0], self.labels.shape[0]))
```

```
        for i in xrange(x.shape[0]):
            for j in xrange(len(self.labels)):
                distances[i,j] = dst.mahalanobis(x[i], self.means[j], self.invcov
                    [:,:, j])
        return self.labels[np.argmin(distances, axis=1)]
```

4. Evaluación del rendimiento

4.1. Prueba estática

La prueba estática consiste en evaluar el rendimiento de los clasificadores en la propia base de datos. Para ello, se ha utilizado el procedimiento de validación cruzada *leave-one-out*, que consiste en quitar un elemento de la base de datos que servirá de test y entrenar los clasificadores con los demás, repitiendo este proceso para cada dato de la base de datos.

A continuación se muestra el programa que realiza este test:

```
# 1-NN
errors = 0.0
for tr, te in loo:
    knn = sklearn.neighbors.KNeighborsClassifier(1, algorithm='brute', metric
        ='euclidean')
    knn = knn.fit(data[tr], labels[tr])
    ypred = knn.predict(data[te])
    errors += ypred != labels[te]
print("k-NN:")
print("\t-- Porcentaje de acierto: " + str((1-(errors/len(labels)))*100) +
    "%")

# Mahalanobis
errors = 0.0
for tr, te in loo:
    maha = mahalanobis.classifMahalanobis()
    maha = maha.fit(data[tr], labels[tr])
    ypred = maha.predict(data[te])
    errors += ypred != labels[te]
print("Mahalanobis:")
print("\t-- Porcentaje de acierto: " + str((1-(errors/len(labels)))*100) +
    "%")
```

Los resultados obtenidos son:

- 1-NN: porcentaje de acierto del $97{,}5\,\%$.
- Distancia de Mahalanobis: porcentaje de acierto del $99{,}25\,\%$.

En la sección 5 se realiza una valoración de estos resultados.

4.2. Prueba dinámica

La prueba dinámica consiste en la evaluación del rendimiento del clasificador con un vídeo en el que aparezcan varios iconos que deberá clasificar correctamente. Se ha utilizado el clasificador que mejor rendimiento obtuvo en la prueba estática, el de la distancia de Mahalanobis.

El resultado de esta prueba puede verse en este enlace: `https://drive.google.com/open?id=13cwOnYXRLw3tUppr5jqAIkpNhq9OvkYe`.

5. Conclusiones

Los resultados de la prueba estática son demasiado optimistas y muestran que puede haber una falta de variedad en la base de datos. Si el rendimiento del clasificador en la integración del prototipo final es bajo, se deberá modificar la base de datos con otro tipo de imágenes que permitan abarcar más casos.

6. Referencias

[1] M.-K. Hu, "Visual pattern recognition by moment invariants", *IEEE Transactions on Information Theory*, vol. 8, n.º 2, págs. 179-187, 1962, ISSN: 0018-9448.