**Visión por Computador**

17

**Trabajo 2**

**RECONOCIMIENTO 2D**

**Jorge Andrés - 679155**

**Javier Aranda - 679184**

**INTRODUCCIÓN**

El objetivo de este segundo trabajo de la asignatura es el desarrollo con “OpenCV” de un programa capaz de reconocer objetos planos y en 2D con una serie de muestras aprendidas previamente. El trabajo en sí consta de dos partes claramente diferenciadas. Una primera que trata el tema del análisis de los distintos objetos que se querrán reconocer, así como el aprendizaje de los parámetros que los caracterizan y facilitarán poder diferenciar unos de otros. La segunda parte es aplicar los datos aprendidos de los objetos de muestra para reconocer distintos objetos en una imagen compuesta.

El programa, además de ser capaz de reconocer los objetos que componen una imagen, tiene que poder decir si un cierto objeto no es de ninguna de las clases estudiadas o, en caso contrario, sí duda entre más de una clase, informará de cuáles son los tipos de objetos que cumplen los criterios. Para conseguirlo, se calcula la distancia de “mahalanobis” del objeto a una cierta clase, y se compara con el correspondiente valor de la chi-cuadrado, sabiendo de éste modo si puede ser el objeto de esa clase o no.

**UMBRALIZACIÓN**

El primer paso para realizar nuestro reconocimiento de imágenes es umbralizar cada una de las imágenes de muestra proporcionadas, consiguiendo de ese modo que la imagen quede dividida en dos partes: fondo y objeto (el fondo de la imagen se representará con el color negro y el objeto de blanco). Esta umbralización para pasar la imagen a binaria es un paso bastante complejo pero que ayuda en gran medida al reconocimiento. Las imágenes binarias son más rápidas de procesar al tener píxeles de tan sólo dos colores, pero conllevan el problema de elegir el método más adecuado para obtener un valor umbral que divida con la máxima precisión posible el fondo del objeto.

Para obtener la imagen umbralizada se estudiaron distintos métodos para saber cuál podría ser el más adecuado para nuestras muestras, entendiendo como adecuado el que aportara mejores resultados globalmente. En primer lugar, se estudió el método adaptativo. Este método lo que hace es fijar un valor umbral en distintas secciones de la imagen dependiendo de lo que haya en dicha región. Es muy utilizado en imágenes que no tienen iluminación uniforme y, por lo tanto, sería difícil encontrar un valor cercano al ideal como umbral.

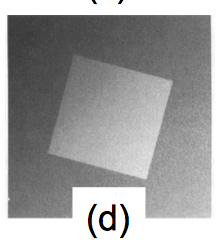
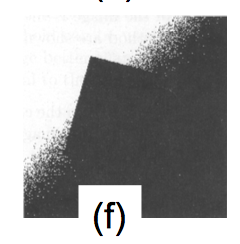


Ilustración 1.- Imagen sin iluminación uniforme y resultado al umbralizarla.

Para implementarlo se utilizó la función de “adaptiveThreshold” accesible en “OpenCV”. Esta función recibía como parámetros la matriz con la imagen original y una matriz en la que devolvería la imagen umbralizada. Además, tenía otros parámetros para realizar el umbralizado mediante la media o el método gaussiano. El método de la media realiza el promedio del área de los vecinos, mientras que el gaussiano es la suma ponderada de los valores de los vecinos donde el peso era una ventana gaussiana. Como último parámetro, había que indicarle cual sería el tamaño de dicha ventana de vecinos que se utilizaría para encontrar el valor umbral, algo que lógicamente no ayudaba a decidirse por este método ya que habría que ajustarlo dependiendo de la imagen.

El método adaptativo se descartó finalmente por la iluminación y las características de los objetos de muestra. La iluminación de la imagen era claramente uniforme existiendo un claro contraste entre el objeto y el fondo, por lo que era innecesario fijar más de un umbral. Además, se observó que con otros métodos el agujero de los objetos círculo y rueda desaparecían al pasarla a binaria, pero esto tampoco se podía arreglar con la umbralización adaptativa. Si se fijara un tamaño de ventana ideal para reconocer el agujero y umbralizarlo bien en el caso de los círculos, se producirían problemas al intentarlo con las ruedas, ya que en uno de los casos el agujero era más oscuro que el objeto y en el otro todo lo contrario.

Finalmente, nos decantamos por el método de “Otsu”. La característica principal y casi única que debía tener una imagen para ser umbralizada correctamente por el método de Otsu es que su histograma tuviera dos picos claramente diferenciados, uno que marcara el color del objeto, y otro el fondo. Por lo tanto, mostramos el histograma de cada muestra, y pudimos apreciar lo que ya se presuponía a simpe vista mirando las imágenes, que el histograma tenía en prácticamente todos los casos los dos picos indicados. Por lo tanto, se utilizó la función “threshold” de “OpenCV”, indicando que se quería utilizar este método de umbralizado, y se obtenía la matriz de la imagen umbralizada según dicho método.

|  |  |
| --- | --- |
| Ilustración .- Histograma de un círculo | Ilustración .- Histograma de un triángulo |

Con todo esto, se fueron recorriendo una a una las imágenes de muestras proporcionadas, umbralizándolas con el método de “Otsu”, y teniéndolas así preparadas para el siguiente paso del aprendizaje. Cabe mencionar que, uno de los aspectos comentados anteriormente sobre el tema de los agujeros de los objetos, aplicando el método de “Otsu” los toma como objetos, pero que no supone un problema importante ya que los objetos que tienen agujeros son muy distintos entre sí en lo que a tamaño se refiere, por lo que no deberían ocasionar excesivos problemas en la etapa de reconocimiento.

|  |  |
| --- | --- |
| Ilustración .- Imagen binaria de un círculo | Ilustración .- Imagen binaria de unarueda |

**BLOPS**

Partiendo de las imágenes umbralizadas del paso anterior, la siguiente acción a realizar era la obtención de los blops que corresponden a los distintos objetos de la imagen (en nuestro caso sólo uno). Este proceso se realizó de manera sencilla con la función “findContours”, pero hubo que estudiar detenidamente cada uno de sus parámetros para utilizarla correctamente. El método recibía como parámetros la imagen umbralizada, así como un vector, que a su vez contenía distintos vectores de puntos. Este vector es el que el método utilizaría para devolver los distintos contornos como secuencia de puntos.

Además, recibía un vector de 4 enteros en el que internamente guardaría la jerarquía entre los blops en el caso de usarla. Dicha jerarquía se obtenía indicando al método los siguientes parámetros:

* EXTERNAL: recupera sólo el contorno externo sin ninguna jerarquía.
* LIST: recupera todos los contornos sin jerarquía.
* CCOMP: recupera todos los contornos y establece una jerarquía de dos niveles.
* TREE: recuperar todos los contornos y construye la jerarquía total.

En nuestro caso, y ya que no se iban a realizar cosas excesivamente complicadas, se optó por utilizar “LIST” y recuperar todos los contornos, pero sin ninguna jerarquía. También, el método “findContours” aceptaba otro parámetro que indicaba como guardar los puntos de cada contorno reconocido, aceptando los siguiente:

* NONE: almacena absolutamente todos los contornos.
* SIMPLE: comprime horizontalmente, verticalmente y diagonales devolviendo sólo los puntos finales.
* TC\_89…: aplica una aproximación al algoritmo “Teh-Chin chain”.

En este caso, ya que sólo se entendían que hacían dos de las opciones, nos decantamos por almacenar todos los contornos mediante “NONE” para no complicarnos más de la cuenta. Se le podía indicar un cierto “offset” para desplazar los contornos, pero tampoco lo vimos necesario así que lo dejamos a 0. Con todos estos pasos, se obtuvieron los contornos que mostramos por pantalla para ver los resultados (uno de cada color).

|  |  |
| --- | --- |
| Ilustración .- Contorno de una rueda | Ilustración .- Contorno de un vagón |

Mostrando los contornos por pantalla, pudimos apreciar un posible problema que de otro modo no hubiéramos visto. En ciertos objetos algo más complejos, como podía ser el vagón, aparte del contorno principal del objeto, se generaban otros más pequeños que formaban parte del vagón, pero con la umbralización habían quedado separados. Por ello, y ya que se tenían que aprender los parámetros para un cierto contorno, se vio necesario aplicar un filtro para eliminarlos, pero que en este paso aún no se podía aplicar ya que hacía falta seguir un cierto criterio (área del contorno).

**PARÁMETROS DESCRIPTORES**

Partiendo de los contornos obtenidos en el paso anterior, se debía de calcular sobre ellos unos ciertos valores que permitieran caracterizar al objeto. Dichos valores debían ser invariantes a traslaciones, rotaciones… ya que, si un mismo objeto aparecía girado en distintas imágenes, no podía dar distintos resultados, sino que debía indicar que era en todos los casos el mismo objeto. Para caracterizar una clase de objeto, se utilizan los llamados descriptores y, en nuestro caso, serían el área, perímetro y los tres primeros “Hu moments”, todos ellos independientes de la localización del objeto dentro de la imagen.

Mediante la función “moments” de “OpenCV”, y pasándole como argumento cada uno de los contornos, se obtenían los distintos momentos. De ellos, el primer momento que devuelve el método es el área y, con la función “arcLength”, se obtiene el perímetro del contorno. Para sacar los “Hu moments” invariantes del objeto, se llama a la función “HuMoments” con los momentos devueltos por el método anterior, y se guardan los tres primeros que son los que se utilizarían. Para cada contorno, se aplica el mismo procedimiento, y se almacenan los valores obtenidos.

Además, debido a que para cada imagen se podía generar más de un contorno como ya vimos en el apartado anterior, se decidió aplicar un filtro de área para dejar tan sólo aquel que fuera más grande y correspondiera al objeto. Para ello, se observó el valor devuelto por cada objeto correspondiente al área, y se fijó un valor de 500 a partir del cual sería contorno de objeto y, por debajo, serían contornos que se ignorarían.

**APRENDIZAJE**

Para cada uno delos objetos que nuestro sistema debía reconocer, se debía aprender la media y varianza de los descriptores indicados en el apartado anterior. Dichos valores debían ser recordados al acabar el programa, por lo que era necesario dejarlos persistentes con algún fichero. Para conseguirlo, se utilizó “FileStorage”, fácil de utilizar para escribir datos, y aún más fácil de leer al devolver datos tan sólo indicándole el nombre de la variable a recuperar.

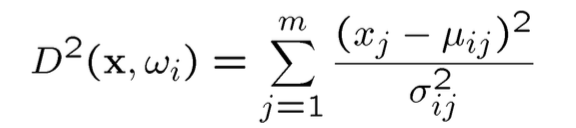
Para calcular la media y varianza se utilizó el cálculo a partir de vectores (“vec” en “OpenCV”). Se realizó un recorrido de los descriptores obtenidos para un objeto de la misma clase, y se iban sumando los resultados en un vector siguiendo operaciones sencillas de vectores y, finalmente, se dividían entre un escalar que era el número de muestras de dicho objeto. Para la varianza, se realizó un proceso similar, pero utilizando la correspondiente fórmula con la media calculada anteriormente (valor de denominador N-1 ya que tiende a la varianza real).

Finalmente, tras realizar todo el proceso, se escribían los valores de media y varianza para cada descriptor en un fichero de formato “yml”. Además, se incluían el número de objetos de cada clase, por si en el proceso de reconocimiento eran necesarios para algo en concreto. Así pues, con todo este procedimiento se conseguía aprender cada una de las clases de objetos que se debían reconocer posteriormente en las imágenes de prueba.

**RECONOCIMIENTO**

Para el proceso de reconocimiento, se siguieron los mismos pasos que para el aprendizaje. En primer lugar, se umbralizaba la imagen con el método de “Otsu”, ya que era el más apropiado como ya se comentó. Seguidamente, se obtuvieron los blops o contornos que configuraban la imagen y, en este caso, sí que eran más de uno los que se debían tener en cuenta, ya que cada imagen estaba formada por distintos objetos a reconocer.

Para cada contorno, se calcularon los descriptores que se habían aprendido, y se aplicó el mismo filtro de área para eliminar los de pequeño tamaño y que no representaban a ningún objeto. Teniendo ya sus valores, se fueron comparando con los guardados en el fichero en la etapa de aprendizaje. Dicha comparación se realizó mediante la distancia de “Mahalanobis” indicada por la siguiente fórmula:



A cada parámetro, se le resta la media para esa clase, y se eleva al cuadrado. El resultado se divide entre la varianza y se realiza el sumatorio para todos los descriptores, obteniendo finalmente un valor entero positivo.

Partiendo de la distancia de “Mahalanobis”, se puede aproximar a una distribución chi-cuadrado con tantos grados de confianza como parámetros, y un nivel de confianza que se decidió que fuera de 0,05. Observando la tabla de la chi-cuadrado, se pudo ver que este valor era en torno a 11,0705, por lo que, si el sumatorio daba un valor inferior para una cierta clase, eso indicaba que cumplía los requisitos para que ese blop fuera de esa clase. Debido a que un blop podía pasar el test para más de una clase, se aplicó el test a cada blop para todas las clases, y se guardaba si lo cumplía o no en una lista para mostrarlo al final.

Al acabar de pasar el test, se mostraba un mensaje por pantalla con la clase a la que correspondía el objeto (una, varias o desconocido). Se mostraba en una imagen a parte el blop separado del resto para diferenciar de manera clara el que se había analizado y, hasta que no se pulsara una tecla, no pasaba a repetir el mismo procedimiento con el siguiente blop.

Finalmente, en lo que a resultados se refiere, se pudo apreciar que en las imágenes a reconocer 1 y 3, se acertaban los resultados esperados, incluido un vagón que debía dar como desconocido al aparecer cortado en la imagen. Sin embargo, la imagen 2 acertaba el círculo y un objeto desconocido compuesto por un triángulo y una rueda que no estaban separados, pero fallaba con el vagón y el rectángulo, aunque para esas clases daba un valor muy cercano al umbral marcado por la chi-cuadrado pero ligeramente superior.

// ¿Por qué falla con vagón y rectángulo?

// IMAGEN DE DOS BLOPS RECONOCIDOS (uno el compuesto en la imagen 2).

|  |  |
| --- | --- |
|  |  |

**REGULARIZACIÓN**

**REFERENCIAS**

* Moodle de “Visión por computador”:
  + https://moodle2.unizar.es/add/course/view.php?id=17541
* Tutorial “opencv”:
  + http://docs.opencv.org/2.4/doc/tutorials/tutorials.html

**ANEXO**

Se incluyen los datos obtenidos de cada descriptor para cada objeto, además de su media y varianza.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **CÍRCULO** | | | | |
|  | **ÁREA** | **PERÍMETRO** | **INV 1** | **INV 2** | **INV 3** |
|  | 1852,0 | 161,196 | 0,159293 | 2,390E-05 | 4,9700E-07 |
|  | 1928,5 | 164,610 | 0,159600 | 1,170E-04 | 5,7100E-06 |
|  | 1884,0 | 163,196 | 0,159325 | 2,980E-05 | 1,8700E-06 |
|  | 1870,0 | 162,024 | 0,159264 | 1,529E-05 | 1,3700E-06 |
|  | 1914,0 | 163,539 | 0,159633 | 1,180E-04 | 3,9866E-07 |
| **MEDIA** | 1889,7 | 162,913 | 1,5942E-01 | 6,0798E-05 | 1,96913E-06 |
| **VARIANZA** | 984,45 | 1,772546 | 3,18035E-08 | 2,70601E-09 | 4,74911E-12 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **RECTÁNGULO** | | | | |
|  | **ÁREA** | **PERÍMETRO** | **INV 1** | **INV 2** | **INV 3** |
|  | 2606,5 | 207,463 | 0,172755 | 2,5395E-03 | 1,81E-06 |
|  | 2592,5 | 201,556 | 0,173599 | 2,8260E-03 | 2,99E-06 |
|  | 2605,5 | 212,468 | 0,172187 | 2,3300E-03 | 2,91E-07 |
|  | 2588,5 | 214,350 | 0,174021 | 2,9690E-03 | 8,93E-07 |
|  | 2576,0 | 205,941 | 0,172238 | 2,2840E-03 | 8,20E-07 |
| **MEDIA** | 2593,8 | 208,3556 | 1,7296E-01 | 2,5897E-03 | 1,3608E-06 |
| **VARIANZA** | 161,2 | 26,4265633 | 6,7372E-07 | 9,07807E-08 | 1,12796E-12 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **RUEDA** | | | | |
|  | **ÁREA** | **PERÍMETRO** | **INV 1** | **INV 2** | **INV 3** |
|  | 5601,5 | 386,375 | 0,159392 | 1,2678E-05 | 1,498E-07 |
|  | 5561,5 | 354,919 | 0,159335 | 1,4737E-05 | 3,990E-08 |
|  | 5512,5 | 340,777 | 0,159335 | 2,1307E-05 | 2,020E-08 |
|  | 5612,5 | 369,061 | 0,159380 | 2,0927E-05 | 3,075E-07 |
|  | 5650,5 | 387,899 | 0,159361 | 5,2725E-06 | 3,130E-07 |
| **MEDIA** | 5587,7 | 367,8062 | 1,5936E-01 | 1,4984E-05 | 1,6608E-07 |
| **VARIANZA** | 2772,7 | 411,6882612 | 6,683E-10 | 4,37488E-11 | 1,97631E-14 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **TRIÁNGULO** | | | | |
|  | **ÁREA** | **PERÍMETRO** | **INV 1** | **INV 2** | **INV 3** |
|  | 2047,0 | 209,823 | 0,191158 | 1,7627E-05 | 4,320E-03 |
|  | 2175,0 | 208,309 | 0,187124 | 2,2506E-05 | 3,631E-03 |
|  | 2114,0 | 210,652 | 0,190613 | 2,1338E-05 | 4,223E-03 |
|  | 2162,5 | 209,723 | 0,188177 | 5,6145E-05 | 3,784E-03 |
|  | 2071,5 | 210,894 | 0,190891 | 1,5103E-05 | 4,275E-03 |
| **MEDIA** | 2114 | 209,8802 | 1,8959E-01 | 2,6544E-05 | 4,0464E-03 |
| **VARIANZA** | 3092,125 | 1,0300297 | 3,31886E-06 | 2,82512E-10 | 9,97409E-08 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **VAGÓN** | | | | |
|  | **ÁREA** | **PERÍMETRO** | **INV 1** | **INV 2** | **INV 3** |
|  | 8023,5 | 494,624 | 0,201529 | 8,9126E-03 | 3,459E-06 |
|  | 7958,5 | 491,772 | 0,193454 | 6,5358E-03 | 1,618E-06 |
|  | 8005,0 | 492,701 | 0,200129 | 8,7637E-03 | 2,248E-06 |
|  | 7920,0 | 476,936 | 0,194394 | 6,8528E-03 | 3,307E-06 |
|  | 7998,0 | 473,127 | 0,199566 | 8,2296E-03 | 1,549E-06 |
| **MEDIA** | 7981 | 485,832 | 1,9781E-01 | 7,8589E-03 | 2,43622E-06 |
| **VARIANZA** | 1724,625 | 100,0804665 | 1,3234E-05 | 1,20734E-06 | 8,23726E-13 |