UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Contar objetos en una imagen mediante técnicas de Visión por Computador

Apellidos, nombre	Agustí i Melchor, Manuel (magusti@disca.upv.es)
Departamento	Departamento de Informática de Sistemas y Computadores (DISCA)
Centro	Escola Tècnica Superior d'Enginyeria Informàtica Universitat Politècnica de València

1 Resumen de las ideas clave

Este documento recoge, de forma práctica y basada en la experimentación, cómo un computador puede enumerar las diferentes instancias de un “objeto” presentes en la escena que le proporciona una cámara. Entendiendo, como objetos, las áreas que cumplen unas mismas características, dentro de la imagen que analiza.

Este proceso recibe el nombre de etiquetado de objetos, también conocido por su denominación anglosajona “labelling” o por “blob detection”. El **interés principal** de esta operación radica en que es posible para el computador encontrar un objeto en la escena y que interactúe con el usuario. Esto es, que responda a las variaciones de las propiedades del objeto que se ha establecido como elemento de interacción. Focalizando la atención del ordenador en un elemento del mundo real (externo al computador) y que tiene presencia y sentido físico para el usuario.

La **dificultad** que presenta esta interacción basada en objetos es que hacemos uso de propiedades muy “simples”, para minimizar el tiempo de búsqueda del objeto. Cuanto más simple es también es posible encontrar más de una aparición del objeto de interés, o bien otras posibles apariciones parciales del mismo que restan precisión al cálculo de la posición del verdadero objeto en la escena real.

Haremos uso de las operaciones que proporciona OpenCV para extraer las diferentes instancias del objeto de interés en una imagen y ver de aislarlas del resto para estudiarlas.

2 Objetivos

El presente documento está encaminado a ofrecer una perspectiva inicial de cómo abordar una situación concreta y habitual en aplicaciones de interacción persona-computador en la que se utiliza la imagen como fuente de entrada de datos. **No es objetivo** de este documento la instalación ni configuración del equipamiento o de las librerías de funciones de soporte para el desarrollo de estas aplicaciones. Abordaremos la situación sin desarrollar un interfaz complejo. Sólo estamos interesados en el problema y ver un par de soluciones.

A partir del estudio de los ejemplos que se abordan, **el lector será capaz de**:

- Enunciar las funciones de OpenCV que permiten detectar y caracterizar objetos.
- Reconocer cuándo se da esta situación en una aplicación que hace uso de imágenes por computador.
- Enunciar la solución que mejor se adapte a su caso particular.

3 Introducción

En muchas ocasiones, el computador analizando una imagen ha de contar cuántos objetos tiene que cumplen unos ciertos criterios. Para, a partir de esa información, tomar una determinada acción. Es una tarea habitual en el campo de la Visión por Computador (VxC) realizada en las primeras etapas de muchos algoritmos de análisis de imagen.

3.1 Planteamiento del problema a resolver

La fig. 1 muestra una situación que se plantea al emular para un computador de escritorio el uso de un mando a distancia como se hace en las consolas de videojuegos y en otros tipos de aplicaciones menos lúdicas, encaminadas a la ejercitación muscular para los mayores.

Para sustituir el ratón por un mando de televisor se puede desarrollar una aplicación de VxC que detecte la posición del mando. Hay que valorar la precisión de este dispositivo (sin limitarlo a un modelo concreto) como sustituto del ratón y valorar su uso. Para ello hay que cuantificar el tiempo de respuesta y la precisión que se consigue al determinar la posición del mando.

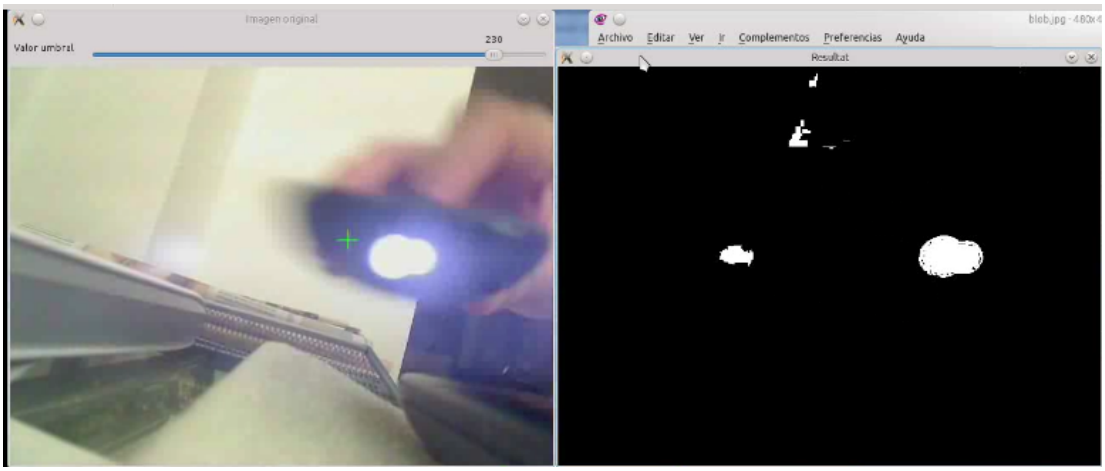


Figura 1: Ejemplo práctico de la problemática a resolver.

Como se quiere que el usuario pueda escoger cualquier mando que tenga a su disposición, la detección es compleja por lo variable de la forma del objeto. Así que utilizaremos la característica de emisión de luz infrarroja (IR) de estos dispositivos para minimizar el **tiempo de respuesta**: el que se emplea en detectar la presencia del mando en la escena que muestra la cámara. De esta forma la detección del punto de luz IR es la tarea a abordar. Para ello, buscaremos un área con un nivel de brillo cuyo valor el usuario de la aplicación podrá escoger con un control (de tipo barra de desplazamiento) situado en la parte superior de la ventan que muestra la imagen.

3.2 Justificación

La imagen de la izquierda de la fig. 1 muestra la imagen de vídeo que devuelve la cámara al enfocar a un mando con uno de los botones pulsados. Se observa un área muy brillante en la imagen (el *led* del mando encendido). También se observa una pequeña cruceta verde, que es la respuesta que da la aplicación a la detección de la **posición del mando** y que corresponde con el centro de gravedad de ese área brillante. Observe que debería estar en el centro del área de luz IR y está hacia la izquierda del mismo. **¿Por qué se ha producido este error y cómo se puede abordar la solución?**

La imagen de la derecha de la fig. 1 muestra que se han detectado otros "objetos" similares provocan el error. La existencia de otros elementos en la imagen influye en la detección del centro de gravedad del objeto … **¡y es que no se ha tenido en cuenta que pueda haber más de uno!** Cada una de esas áreas de puntos conectados que tienen cierta característica en común reciben el nombre de *blob,* cabe destacar la noción de forma irregular y tamaño variable que los caracterizan. **Hay que identificar los diferentes objetos,**

contarlos y caracterizarlos. Así será posible elaborar una estrategia que permita encontrar el que se está buscando en caso de que exista más de una opción.

3.3 Planteamiento de la solución

OpenCV [1] se ha convertido en una de las referencias imprescindibles que ofrece un API con el que desarrollar aplicaciones de VxC en C/C++, *Phyton* y Java (por citar algunas) y con opción de ser portada a diferentes plataformas de computadores incluyendo las más recientes destinadas a móviles y tabletas. OpenCV, ya desde su versión 2.4, ofrece una serie de opciones para la detección de objetos en las escenas, que se basan en buscar los *blobs* que cumplen ciertas propiedades en una escena.

En función del tipo de objeto a buscar hay que evaluar cuál de los detectores disponibles puede ser más efectivo. Nos vamos a adentrar en uno que nos permite ver la solución como un solo bloque y lo veremos desde dos perspectivas de diferente nivel de abstracción:

1. Solución global. Esta solución está basada en utilizar el *SimpleBlobDetector*. Ofrece una solución en bloque que es parametrizable.
2. Solución por pasos. Que permite detallar los pasos intermedios y que ofrece más control al desarrollador, a cambio de una mayor complejidad a la hora de implementarla. Esto es, se puede ajustar a nuestras necesidades, si las peculiaridades del caso permiten alguna simplificación o requieren otras comprobaciones diferentes de las que contempla *SimpleBlobDetector* con su solución "automatizada".

Veamos las dos soluciones por separado. Tras lo cual se compararán sus resultados y se propondrá al lector un par de ideas para poder determinar cuál elegir en un caso concreto.

4 Solución automática: *SimpleBlobDetector*

Este detector se encargará de generar una lista de objetos (*blobs)* encontrados a partir de una imagen, en grises y de un conjunto de parámetros que sirven para ajustar la búsqueda a nuestro contexto. El funcionamiento básico [1] de este algoritmo se basa en:

- Generar *N, véase ecuación 1,* imágenes binarias, a partir de hacer variar un valor umbral desde *minThreshold* hasta *maxThreshold*, con incrementos de *thresholdStep*. Para cada imagen se genera una lista con la posición y extensión de cada *blob* encontrado en cada imagen.

$$N = \frac{(maxThreshold - minThreshold)}{thresholdStep} \quad (1)$$

- Fusionar los *blobs* de todos los niveles cuyos centros de gravedad disten menos que el valor fijado en *minDistBetweenBlobs*. Los *blobs* fusionados se eliminan de la lista a devolver y se inserta el nuevo *blob* resultado de la agrupación, para el que se recalcula su centro y radio.

Como ejemplo de uso de esta solución se puede experimentar con la implementación de S. Mallick [2], que consta de dos pasos:

1. La detección de objetos. Se muestra en el listado 1 y se encarga de obtener la lista con la información de los *blobs* en la imagen. Se guardan en una estructura de tipo KeyPoint que guarda las coordenadas de posición, tamaño y otros parámetros de cada *blob*.
2. La gestión de los resultados. Esta parte depende de la aplicación a desarrollar, en este caso se limita a mostrar los resultados al usuario sobrepuestos a la imagen original para verificar su corrección.

```
Mat im;
SimpleBlobDetector detector;
std::vector<KeyPoint> keypoints;

im = imread( "blob.jpg", IMREAD_GRAYSCALE );
// ¿Parámetros?
detector.detect( im, keypoints);
```

Listado 1: Ejemplo de uso de SimpleBlobDetector de OpenCV.

```
SimpleBlobDetector::Params params;

params.minThreshold = 10;            // Cambiar umbrales
params.maxThreshold = 200;

params.filterByArea = true;          // Filtrar por área.
params.minArea = 1500;

params.filterByCircularity = true;   // Filtrar por circularidad
params.minCircularity = 0.1;

params.filterByConvexity = true;     // Filtrar por convexidad
params.minConvexity = 0.87;

params.filterByInertia = true;       // Filtrar por inercia
params.minInertiaRatio = 0.01;

SimpleBlobDetector::detector(params);
```

Listado 2: Inicialización de parámetros para SimpleBlobDetector.

Por defecto se utiliza un conjunto de parámetros que buscan objetos redondos de color negro, si es su caso no hace falta cambiar nada. Pero al algoritmo se le pueden especificar como parámetros otras propiedades a tener en cuenta para dejar o quitar de la lista los *blobs* encontrados hasta el momento en función del nivel de gris, tamaño, circularidad, ratio entre los ejes mayores y convexidad.

Así, antes de la ejecución, en la línea que dice “¿Parámetros?” se pueden asignar los valores que se considere, como se muestra en el listado 2.

4.1 Mostrar resultados: centros de gravedad

Para ello, en el ejemplo examinado, se utiliza (véase listado 3) la función *drawKeypoints* que permite dibujar círculos centrados en el centro de masas de cada *blob* y con un radio que corresponde con el tamaño del *blob* en cuestión.

```
Mat im_with_keypoints;

drawKeypoints( im, keypoints, im_with_keypoints,
               Scalar(0,0,255),
               DrawMatchesFlags::DRAW_RICH_KEYPOINTS );

imshow("keypoints", im_with_keypoints );
waitKey(0);
```

Listado 3: Mostrando resultados de SimpleBlobDetector.

La fig. 2 muestra un ejemplo de los resultados obtenidos, al colocar este código después de la detección de *blobs*,

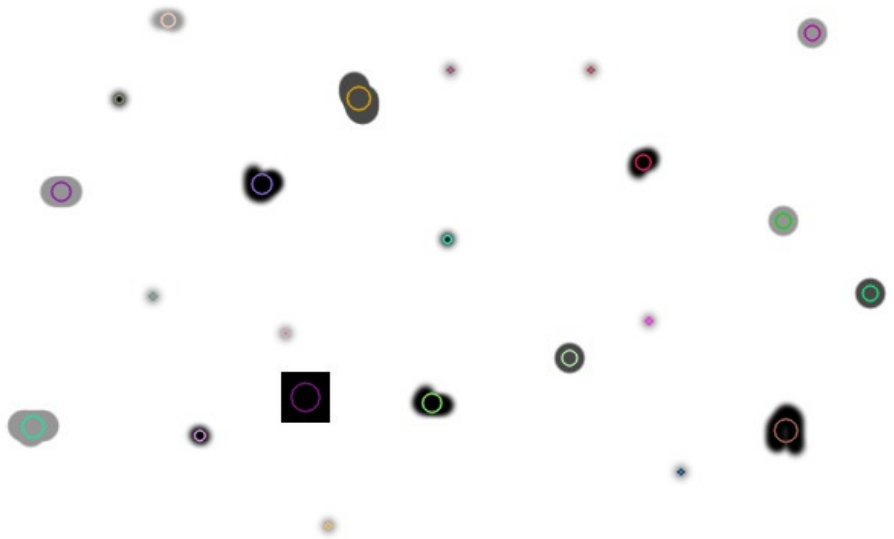


Figura 2: Resultados de SimpleBlobDetector.

5 Solución por pasos

Si las necesidades o las peculiaridades del caso permiten alguna simplificación o requieren otras comprobaciones diferentes de las de la solución anterior hay que implementar el algoritmo que analice la imagen para encontrar esos *blobs*.

Para ello se puede tomar como base el trabajo de D. Millán [3] que muestra, en operaciones básicas de *OpenCV*, una posible secuencia, véase listado 4, para abordar este problema y que subdivide en tres etapas:

- Preprocesado de la imagen. Es el paso de acondicionamiento de las características de la imagen de partida al algoritmo en cuestión. Está compuesto por un filtro (*Smooth*) para eliminar ruido global, una umbralización (*Threshold*) para obtener una versión binaria de la imagen de partida y un par de filtros morfológicos (Erode y Dilate) que quitan ruidos puntuales.
- Buscar los contornos. En la imagen binaria, se buscan con la función *cvFindContours* y se simplifican con *cvApproxPoly*, que aproxima una curva o polígono con otro de menos vértices.
- Obtenidas las posiciones de los objetos es momento de dimensionarlos y de enseñarle los resultados al usuario, p. ej., dibujando los bordes de los objetos encontrados.

```
#include <cv.h>
#include <highgui.h>
#include <cvaux.h>
#include <stdio.h>

const char *VENTANA="Detección de Blobs (objetos)";

int main( int argc, char** argv )
{
     IplImage *imagen, *imagen_color, *smooth, *threshold,
              *morfolg, *blobResult *img_contornos;
     CvSeq *contour, *contourLow;
     CvScalar avg, avgStd, color;
     CvRect rect;

     imagen=cvLoadImage(argv[1],0); //load image in gray level

     //Inicializaciones
     smooth  =  cvCreateImage(cvSize(imagen->width,  imagen->height),
                              IPL_DEPTH_8U, 1);
     threshold = cvCreateImage(cvSize(imagen->width, imagen->height),
                              IPL_DEPTH_8U, 1);
     morfolg  =  cvCreateImage(cvSize(imagen->width,  imagen->height),
                              IPL_DEPTH_8U, 1);

     contour = 0; contourLow = 0; color;
     CvMemStorage* storage = cvCreateMemStorage(0);
     cvNamedWindow( VENTANA, 0 );

     //Prepocesado
     cvSmooth(imagen, smooth, CV_GAUSSIAN, 3, 0, 0, 0);
     cvAvgSdv(smooth, &avg, &avgStd, NULL);
     cvThreshold(smooth,      threshold,      (int)avg.val[0]-7*(int)
(avgStd.val[0]/8), 255, CV_THRESH_BINARY_INV);
     cvErode(threshold, morfolg, NULL,1);
     cvDilate(morfolg, morfolg, NULL,1);

     //Buscar contornos
     img_contornos=cvCloneImage( morfolg );
     cvFindContours( img_contornos, storage, &contour,
                    sizeof(CvContour), CV_RETR_EXTERNAL,
                    CV_CHAIN_APPROX_SIMPLE, cvPoint(0, 0) );
     contourLow=cvApproxPoly(contour, sizeof(CvContour), storage,
                              CV_POLY_APPROX_DP,1,1);
     //
     // Mostrar resultados
     //
     ...
```

Listado 4: Ejemplo de solución descompuesta en pasos simples.

5.1 Mostrar resultados: centros de gravedad

El número o posición de los *blobs* encontrados **no es el único resultado que se obtiene** en este proceso, así que, para ver la versatilidad de esta aproximación, en la parte final, se puede incluir que sea posible ver los resultados intermedios y, también, el final. El código que muestra en el listado 5 pinta los *blobs* y permite alternar entre la vista del resultado final y la de los pasos intermedios con el uso de las teclas '1', '2', '3', '4' o '5'. La aplicación acabará al pulsar la letra 'q'.

```
...
//Mostrar resultados
for( ; contourLow != 0; contourLow = contourLow->h_next ) {
   color = CV_RGB( rand()&200, rand()&200, rand()&200 );
   cvDrawContours( imagen_color, contourLow, color, color,
                   -1, 0, 8, cvPoint(0,0) );
}
cvShowImage(VENTANA, imagen_color);
char c;
while ((c = waitKey(0)) != 'q') {
 if((char) c == '1')      cvShowImage(VENTANA, imagen);
 else if((char) c == '2') cvShowImage(VENTANA, smooth);
 else if((char) c == '3') cvShowImage(VENTANA, threshold);
 else if((char) c == '4') cvShowImage(VENTANA, morfolg);
 else if((char) c == '5') cvShowImage(VENTANA, imagen_color);
}
cvDestroyWindow(VENTANA);
return 0;
}
```

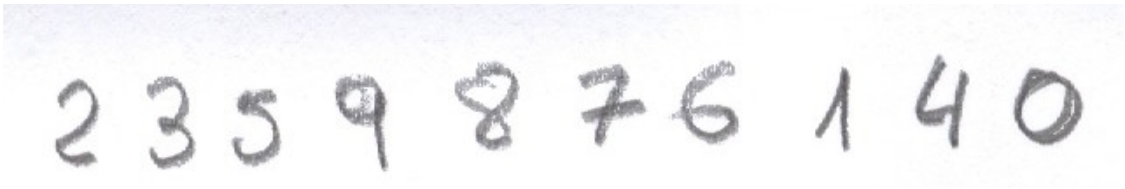
Listado 5: Ejemplo de salida de resultados para el usuario: contornos.

```
...
//Mostrar resultados
 CvRect rect;
 CvPoint pt1, pt2;

 //For each contour found
 for( ; contourLow != 0; contourLow = contourLow->h_next )
 {
  rect=cvBoundingRect(contourLow, NULL);
  pt1.x = rect.x;
  pt2.x = (rect.x+rect.width);
  pt1.y = rect.y;
  pt2.y = (rect.y+rect.height);
  cvRectangle(imagen_color, pt1,pt2, color, 1, 8, 0);
 }
 cvShowImage(VENTANA, imagen_color);
 ...
```

Listado 6: Ejemplo de salida de resultados para el usuario: caja contenedora.

Pero es que el resultado final tampoco es único, se pueden mostrar diferentes propiedades al respecto de los objetos encontrados. La fig. 3 muestra el resultado de aplicar esta solución a una imagen de dígitos manuscritos, fig. 3a. Obteniendo los contornos, fig. 3b, a partir del código que muestra el listado 5. Y también se pueden generar otras salidas al usuario: p. ej. se podría mostrar la caja contenedora (o *Bounding Box*), véase la fig. 3c obtenida con el listado 6.



a)

b)

c)

Figura 3: Resultados de la solución por pasos: imagen original (b) y resultados mostrando los contornos (b) y la caja contenedora (c).

A partir de la información de contornos se puede realizar la detección de objetos (que es la que nos interesaba aquí), pero también se puede realizar un análisis de formas y la clasificación o reconocimiento de formas. El cálculo de estadísticos de los "objetos" encontrados se puede hacer, a partir de este punto, como muestra el listado 7.

```
CvMoments moments;
CvHuMoments humoments;

//First calculate object moments
cvMoments(contourLow, &moments, 0);

//Now calculate hu moments
cvGetHuMoments(&moments, &humoments);
printf("%f\t%f\t%f\t%f\t%f\t%f\t%f\n",          humoments.hu1,
humoments.hu2,  humoments.hu3,  humoments.hu4,  humoments.hu5,
humoments.hu6, humoments.hu7);
```

Listado 7: Cálculo de estadísticos de los objetos.

6 Experimentos y comparativa

Es posible comprobar el funcionamiento de las dos aproximaciones utilizando la imagen de la fig. 4. En ella se han situado varios objectos agrupados en cada fila por sus diferentes valores de las propiedades que los describen: área, nivel de gris, circularidad, momento de inercia y convexidad.

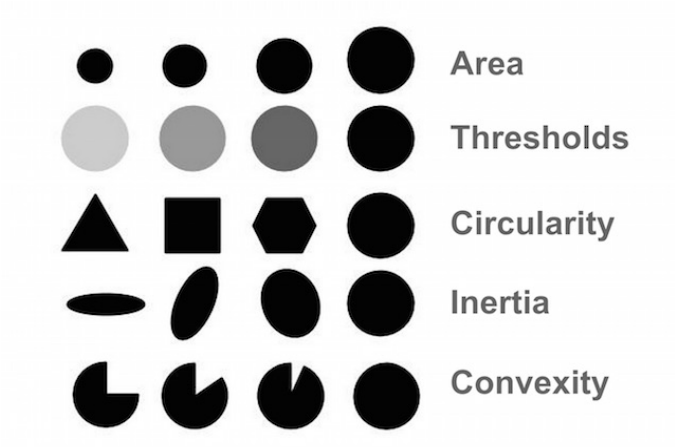
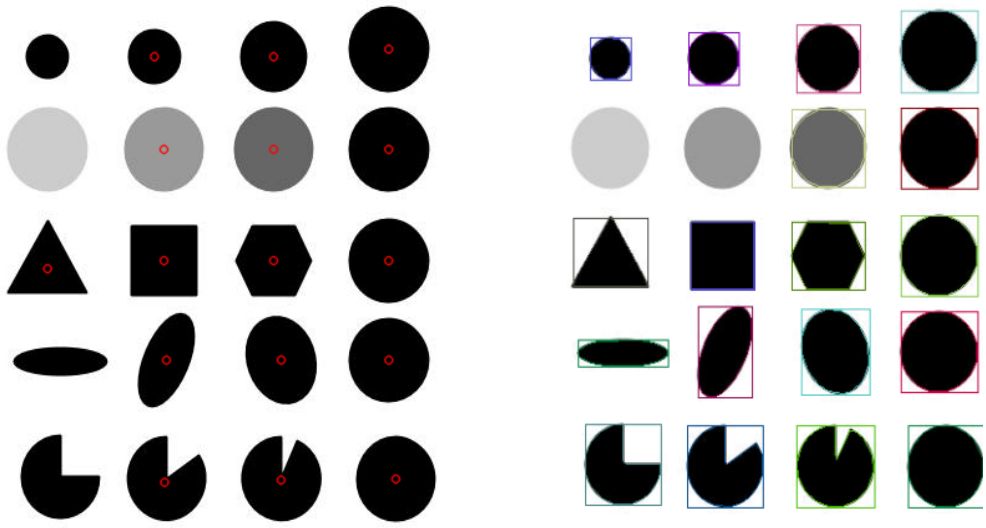


Figura 4: Imagen de test [2] para comprobar como se comporta el algoritmo frente a diferentes situaciones en la imagen.

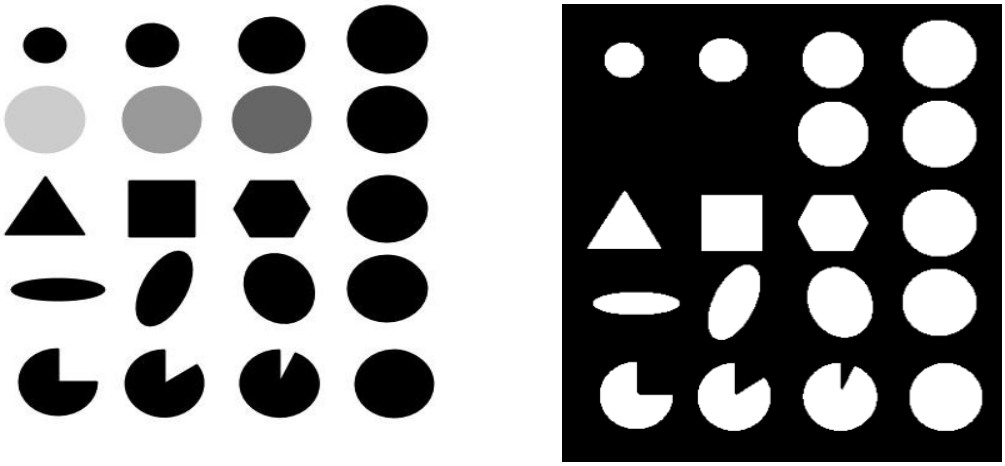
La soluciones exploradas obtienen, con los parámetros por defecto, los resultados mostrados en la fig. 5. ¿Por qué han fallado ambas en algunos objetos?



a) *b)*

Figura 5: Resultado de la solución global (a) y por pasos (b).

La fig. 6 lo muestra a partir del resultado intermedio de la umbralización de la solución por etapas. Sin un control más preciso de los valores de los parámetros, algunos objetos pueden no ser detectados al no acertar con el valor umbral.



a) *b)*

Figura 6: Imagen original (a) y paso intermedio (b) de la solución por pasos evidencia que se pueden perder objetos.

Ambas soluciones alcanza los mismos resultados si se utilizan los mismos parámetros. Dejo de la mano del lector interesado comprobar que llegan a detectar correctamente todos los objetos de la imagen de prueba.

7 Conclusión

La solución global de OpenCV hace un muy buen trabajo, pero se ha estudiado una implementación detallada para poderla aplicar en casos donde se necesite más libertad o donde se sepa que se puede prescindir de determinados pasos. La experimentación realizada ha dado pruebas de ello.

No todo acaba aquí. En caso de necesitar seguir a partir de aquí diferenciando entre esa lista de objetos similares, se debería pasar a profundizar en operaciones como las comentadas de cálculo de características, momentos, etc.

8 Bibliografía

[1] OpenCV. Disponible en <http://opencv.org/>. Consultada el 28 de mayo de 2016.
[2] Mallick, S. (2015). *Blob Detection Using OpenCV (Python, C++)*. Disponible en <http://www.learnopencv.com/opencv-c-vs-python-vs-matlab-for-computer-vision/>. Consultada el 8 de abril de 2016.
[3] Millan, D. (2012). *Segmentation and feature extraction. Contours and blob detection*. Disponible en <http://blog.damiles.com/2010/12/segmentation-and-feature-extraction-contours-and-blob-detection/>. Consultada el 8 de abril de 2016.