SEGMENTACIÓN DE IMÁGENES

BASADO EN ETIQUETACIÓN DE PÍXELES

GONZALO LUZARDO M.

gonchalox@gmail.com

gluzardo@espol.edu.ec

Introducción

El siguiente articulo trata de la segmentación de imágenes para identificar objetos en la una escena basado en la etiquetación de pixeles. Aquí se propone un algoritmo de etiquetado que lo denominaremos “Merge Labeling”, el cual etiqueta aquellos pixeles activos próximos como pertenecientes al mismo objeto. La entrada al algoritmo es la imagen binarizada que contiene los pixeles activos correspondientes a los objetos a identificar, y la salida es una matriz de etiquetas para cada uno de los píxeles de la imagen, en donde cada etiqueta identifica el objeto al cual dicho pixel pertenece.

Este algoritmo es denominado “Merge Labeling”, debido a que en el proceso del algoritmo, este junta objetos etiquetados por separado cuando estos están unidos por al menos un pixel. Una ventaja de este algoritmo es su rapidez y complejidad lineal, pero la desventaja obvia es que objetos que sabemos son distintos, al estar juntos los toma como uno solo.

Si sabemos que los objetos a ser analizados no están juntos, por ejemplo sobre una banda transportadora, este algoritmo, debido a su simplicidad puede ser de mucha ayuda.

Los algoritmos y experimentos fueron realizados utilizando el lenguaje de programación Python, todo el código fuente está disponible tanto en Google Code, así como junto a este artículo, y puede ser utilizado y modificado siempre y cuando se haga referencia al autor.

Algoritmo de etiquetación

El algoritmo de etiquetación recibe como entrada la imagen binarizada que contiene los pixeles activos correspondiente a los objetos a identificar en la escena, y cuya salida es una matriz de etiquetas para cada uno de los píxeles de la imagen, en donde cada etiqueta identifica el objeto al cual dicho pixel pertenece.

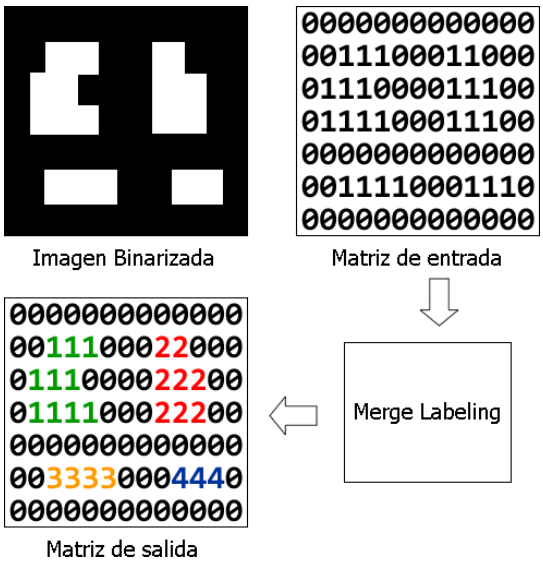


Fig. 1. Proceso de etiquetación, entradas y salidas del algoritmo

Podemos identificar en la imagen de entrada dos posibles valores para los píxeles: PIXEL_ACTIVO y PIXEL_NO_ACTIVO, y en la salida podemos identificar valores de PIXEL_NO_ACTIVO y las etiquetas para cada uno de píxeles, véase la Fig. 1.

En la Fig. 1 podemos observar un diagrama que representa el proceso de etiquetación, la entrada es la imagen binarizada donde el valor de 1 representa un PIXEL_ACTIVO y el valor de 0 un PIXEL_NO_ACTIVO. La salida es una matriz de etiquetas que identifica al objeto al cual pertenece cada pixel.

El algoritmo de etiquetado "Merge Labeling" es el siguiente:

```
1. Por cada píxel P de la matriz de entrada, por cada fila y columna
2.   Si P tiene etiqueta
3.     Si todos los vecinos etiquetados tienen la misma etiqueta
4.        Se marcan los vecinos no etiquetados con dicha etiqueta
5.     Si no.
6.        Se obtiene la lista de vecinos presentes en la vecindad
7.        Se procede a hacer el cambio de etiquetas (merge)
8.   Si P no tiene etiqueta
9.     Si todos sus vecinos no están etiquetados o no tiene vecinos
10.       Se marca el pixel con la etiqueta siguiente
11.       Se marcan todos los vecinos no etiquetados con la misma etiqueta
12.       Incrementamos el valor de la etiqueta
13.    Y Si todos los vecinos etiquetados son diferentes
14.       Se obtiene la lista ordenada de vecinos presentes en la vecindad
15.       Se procede a hacer el cambio de etiquetas (merge)
16.    Si no.
17.       Se marca el pixel con la etiqueta de sus vecinos
18.       Se marcan todos los vecinos no etiquetados con la misma etiqueta
```

En la línea 7 y 15 es donde se realiza el merge o fusión de objetos etiquetados; como el algoritmo recorre la matriz de entrada de manera horizontal, si nota que dos objetos que están unidos y que hasta el momento han sido etiquetados como objetos diferentes, se les otorga a ambos la etiqueta del primero de ellos que siempre será la menor de las dos.

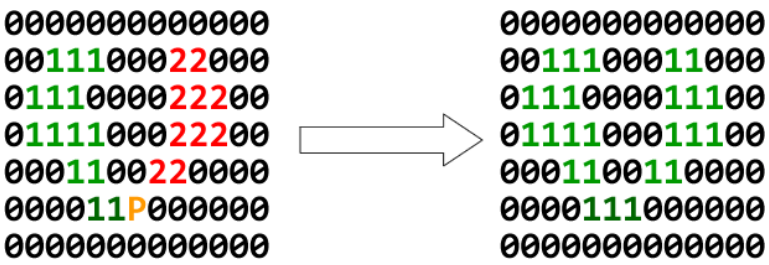


Fig. 2. Proceso de fusión de objetos etiquetados, cambio de etiquetas 1 por 2 hasta llegar a P

En la Fig. 2 podemos observar la fusión de objetos etiquetados. Notamos que al llegar a P el algoritmo nota que los dos objetos etiquetados como 1 y 2 realmente son uno mismo, de esta forma procede a

cambiar todas las etiquetas marcadas como 2 a su nueva etiqueta 1 desde el inicio de la matriz hasta llegar a P.

Simple Python Image Processing Library

Para la realización de los experimentos se creó una librería en Python denominada "Simple Python Image Processing Library" cuya versión más actual puede ser descargada desde el repositorio de Google Code a través de un cliente `svn` de la siguiente manera:

```
svn checkout http://simplepymage.googlecode.com/svn/trunk/ simplepymage-read-only
```

El repositorio contiene los siguientes ficheros:

Fichero	Descripción
`Blob.py`	*Clase que contiene un blob o segmento de imagen identificado*
`Clustering.py`	*Algoritmos de clustering y segmentación de imágenes, en este fichero está implementado el algoritmo de etiquetado*
`ErrorControl.py`	*Clase para el control de errores a través de excepciones*
`ImageProcessing.py`	*Clase que contiene algunos algoritmos de procesamiento de imagen*
`Imagex.py`	*Clase para manejo de imágenes*

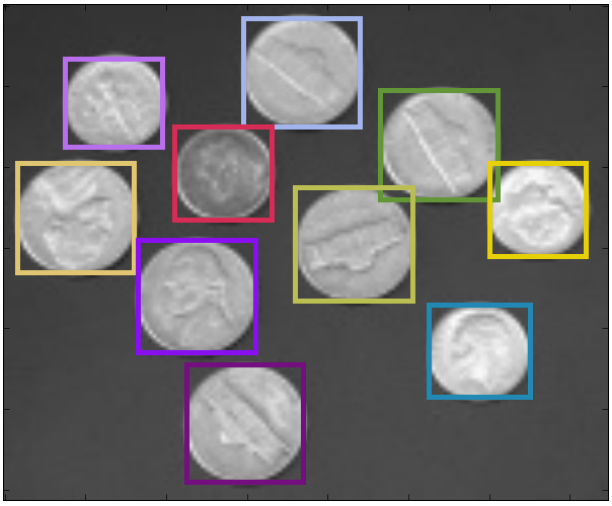
Experimentos y resultados obtenidos

Se realizaron tres experimentos, el primero de ellos analizamos la imagen `coins.jpg` que muestra un conjunto de monedas relativamente separadas entre sí, el código utilizado es el siguiente:

```
from Imagex import *
from ImageProcessing import *
from Clustering import *
from Blob import *
from pylab import figure

#Cargamos el fichero con la imagen
img=Imagex()
img.loadImage("coins.jpg")
#Obtenemos una imagen en escala de grises
binaryImage = rgb2gray(img)
#Binarizamos la imagen
binaryze(binaryImage,100)
#Utilizando los datos de la imagen procedemos ha hacer el cluster
#Esta vez utilizaremos labeling para la clusterizacion
#Obtenemos la lista de blobs encontrados en la imagen
labeled, blobs = labelClustering(binaryImage)
#Dibujamos los blobs
fig = figure()
#Dibujamos la imagen original
h = img.imgShow()
ax = h.axes
```

```
for b in range(0,len(blobs)):
       x = blobs[b]._xMin
       y = blobs[b]._yMin
       w = blobs[b]._width
       h = blobs[b]._height
       c = blobs[b]._color
       r = Rectangle((x,y),w, h, fill=False, edgecolor=c,linewidth=3)
       ax.add_patch(r)

#Mostramos los rectangulos
fig.show()
```

El resultado fue el siguiente:



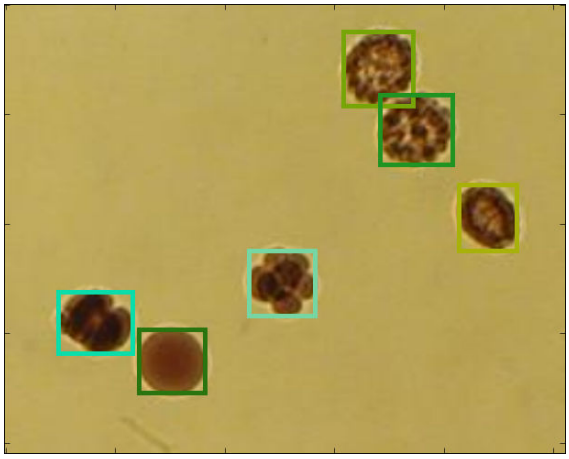
Como podemos observar el algoritmo ha segmentado de manera correcta la imagen de entrada, ya que ha logrado identificar de manera individual cada una de las monedas presentes en la imagen.

En el siguiente experimento analizamos la imagen `embryos.jpg` la cual es una imagen de unos embriones vistos bajo un microscopio, el código utilizado es el siguiente:

```
from Imagex import *
from ImageProcessing import *
from Clustering import *
from Blob import *
from pylab import figure

#Cargamos el fichero con la imagen
img=Imagex()
img.loadImage("embryos.jpg")
#Obtenemos una imagen en escala de grises y la invertimos
binaryImage = rgb2gray(img)
invertImage(binaryImage)
#Binarizamos la imagen
binaryze(binaryImage,130)
```

```
#Utilizando los datos de la imagen procedemos ha hacer el cluster
#Esta vez utilizaremos labeling para la clusterizacion
#Obtenemos la lista de blobs encontrados en la imagen
labeled, blobs = labelClustering(binaryImage)
#Dibujamos los blobs
fig = figure()
#Dibujamos la imagen original
h = img.imgShow()
ax = h.axes

for b in range(0,len(blobs)):
       x = blobs[b]._xMin
       y = blobs[b]._yMin
       w = blobs[b]._width
       h = blobs[b]._height
       c = blobs[b]._color
       r = Rectangle((x,y),w, h, fill=False, edgecolor=c,linewidth=3)
       ax.add_patch(r)

#Mostramos los rectangulos
fig.show()
```

El resultado fue el siguiente:



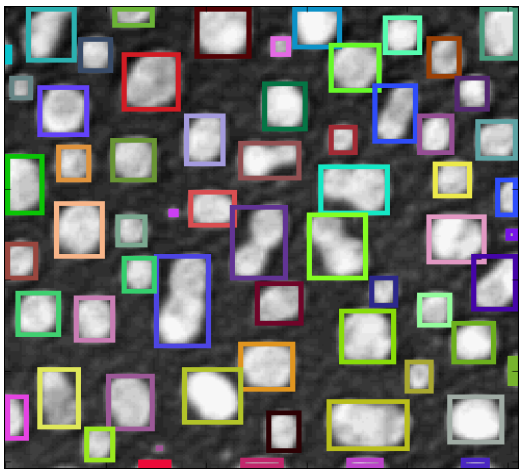
Como podemos observar el algoritmo ha segmentado de manera correcta la imagen de entrada, ya que ha logrado identificar de manera individual cada uno de los embriones presentes en la imagen.

En el siguiente experimento analizamos la imagen `blobs.gif` la cual es una imagen de unas células vistas bajo un microscopio, el código utilizado es el siguiente:

```
from Imagex import *
from ImageProcessing import *
from Clustering import *
from Blob import *
from pylab import figure
```

```
#Cargamos el fichero con la imagen
img=Imagex()
img.loadImage("blobs.gif")
#Obtenemos una imagen en escala de grises y la invertimos
binaryImage = Imagex()
binaryImage = img.copy()
#invertImage(binaryImage)
#Binarizamos la imagen
binaryze(binaryImage,130)
#Utilizando los datos de la imagen procedemos ha hacer el cluster
#Esta vez utilizaremos labeling para la clusterizacion
#Obtenemos la lista de blobs encontrados en la imagen
labeled, blobs = labelClustering(binaryImage)
#Dibujamos los blobs
fig = figure()
#Dibujamos la imagen original
h = img.imgShow()
ax = h.axes

for b in range(0,len(blobs)):
      x = blobs[b]._xMin
      y = blobs[b]._yMin
      w = blobs[b]._width
      h = blobs[b]._height
      c = blobs[b]._color
      r = Rectangle((x,y),w, h, fill=False, edgecolor=c,linewidth=3)
      ax.add_patch(r)

#Mostramos los rectangulos
fig.show()
```

El resultado fue el siguiente:



Como podemos observar, que a diferencia de los experimentos anteriores, el algoritmo no ha segmentado de manera correcta la imagen de entrada, ya que existen células que al estar muy juntas o solapadas entre sí, han sido identificadas como una sola cuando realmente son dos.

Conclusiones

Si analizamos los resultados obtenidos podemos mencionar que el algoritmo de segmentación basado en etiquetación de píxeles propuesto resulta muy efectivo en condiciones en las cuales sabemos que los objetos a segmentar no están juntos en la escena.

El algoritmo propuesto se basa en el hecho de que los objetos a identificar en la imagen no están juntos tal y como sucede en los dos primeros experimentos que hemos realizado. En el tercer experimento podemos observar claramente la deficiencia de nuestro algoritmo.