Tarea 6

Angel Manrique Pozos Flores; N.C M07211505
Instituto Tecnológico Nacional de México, Blvd. Industrial, Mesa de Otay, 22430 Tijuana, B.C., México.
(Dated: 16 de Febrero del 2016)

En el presente trabajo se implementa un algoritmo aplicando los operadores morfológicos de apertura y cerradura basados en los operadores de dilatacion y erosion.

I. INTRODUCCIÓN

El procesamiento de imágenes es de gran utilidad en la mayoría de las áreas de investigación ya que todo lo que captamos en el mundo la gran mayoría de los datos provienen de nuestros sentidos en especial la vista.

Por ello el estudio de la visión computacional es de gran importancia, el presente trabajo se hace un acercamiento a esta área, utilizando las librerías de visión open source OpenCV y el software libre CodeBlocks el cual se configuro para que pudiera ser capaz de reconocer las librerías de OpenCV.

II. OPERACIONES MORFOLÓGICAS, APERTURA Y CERRADURA

Las operaciones morfológicas de apertura y cerradura se basan en la combinación de las operaciones previamente estudiadas.

- Erosión
- Dilatación

II.1. Apertura

Se obtiene de erosionar una imagen seguida de aplicarle una dilatación.

$$dst = open(src, elmnt) = dilate(erode(src, elmnt)) \quad (1)$$

Se utiliza para remover pequeños objetos, asumiendo que estos objetos son mas brillantes respecto al fondo, a manera de ejemplo se muestra la Figura 1, en la imagen del lado izquierdo se muestra la original y en la del lado derecho el resultado de aplicar la apertura, donde se puede observar como los pequeños espacios que existen entre las esquinas tienden a desaparecer.

II.2. Cerradura

Se obtiene de aplicar una dilatación a una imagen, seguida de una erosión.

Figura 1: Imagen con apertura aplicada.

$$dst = close(src, elmnt) = erode(dilate(src, elmnt)) \quad (2)$$

Se utiliza comúnmente para remover pequeños huecos en la imagen (regiones oscuras).

Figura 2: Imagen con cerradura aplicada.

III. DESARROLLO

Se tienen las siguientes imágenes a las cuales se les desea aplicar el tratamiento de transformación morfológica.

Figura 3: Imagen a tratar con apertura.

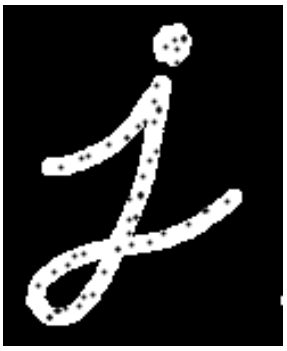
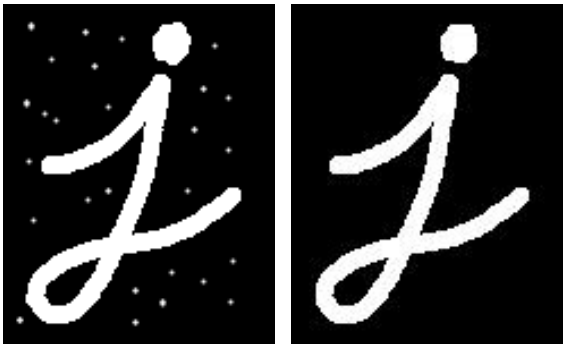


Figura 4: Imagen a tratar con cerradura.

Para ello se desarrollo el siguiente código en C++ utilizando las librerías de visión de OpenCV.

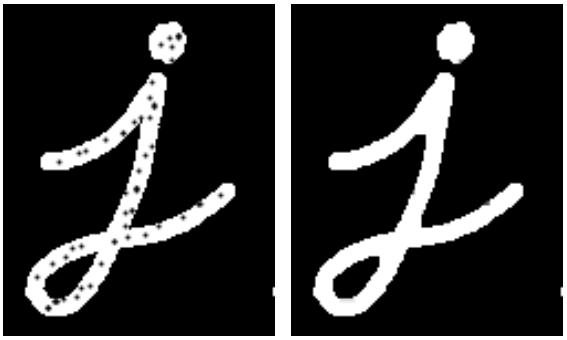
```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>

using namespace std;
using namespace cv;


int main()
{
    Mat img1 = imread("opening.png",
        CV_LOAD_IMAGE_GRAYSCALE);
    Mat img2 = imread("closing.png",
        CV_LOAD_IMAGE_GRAYSCALE);

    //Structural Element
        int morph_s1 = 2;
        Mat element =
            getStructuringElement(cv::
            MORPH_RECT,
                cv::Size(2 * morph_s1 + 1,
                    2 * morph_s1 + 1),
                cv::Point(morph_s1,
                    morph_s1));


    Mat dst1, dst2;

    // Opening
    morphologyEx( img1, dst1, MORPH_OPEN
        , element );
    imshow("Imagen Original 1", img1);
    imshow("Apertura aplicada a Imagen
        Original 1", dst1);

    // Closing
    morphologyEx( img2, dst2,
        MORPH_CLOSE, element );
    imshow("Imagen Original 2", img2);
    imshow("Cerradura aplicada a Imagen
        Original 2", dst2);

    waitKey(0);
    return 0;
}
```

Obteniendo como resultado las siguientes imágenes.



(a) Imagen original sin aplicar apertura. (b) Imagen con apertura aplicada.

Figura 5: Imágenes procesadas, donde se observa que el ruido (*puntos blancos*) a sido eliminado al aplicarle una apertura a la imagen.



(a) Imagen original sin aplicar cerradura. (b) Imagen con cerradura aplicada.

Figura 6: Imagenes procesadas, donde se observa que el ruido dentro de nuestro objeto (*puntos negros*) a sido eliminado al aplicarle una cerradura a la imagen.

III.1. Hit-miss

La transformación de hit-miss es una poderosa herramienta para el análisis de imágenes binarias, en esta sección se muestra como utilizar esta función morfológica, esta transformacion es de utilidad para poder localizar pixeles que correspondan con el kernel B_1 y no corresponden con el kernel B_2 donde esta operacion se define como:

$$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2) \quad (3)$$

Para ello se tiene la Figura 7 ya transformada a binario.

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	1	0	1	0	0
0	0	1	0	0	0	0	0
0	0	1	0	0	1	1	0
0	1	0	1	0	0	1	0
0	1	1	0	0	0	0	0

Figura 7: Imagen binaria a aplicar hit-miss transformacion.

Y los siguientes kernels mostrados en la Figura 8.

0	1	0
1	0	1
0	1	0

a

0	0	0
0	1	0
0	0	0

b

0	1	0
1	-1	1
0	1	0

c

Figura 8: Kernels utilizados, donde a) es el kernel de *Hit* y b) el kernel de *Miss* y c) el kernel combinado de ambos donde 1 = primer plano, -1 = fondo y 0 = no importa.

Para ello se desarrollo el siguiente codigo:

```
#include <stdlib.h>
#include <cv.hpp>
#include <cxcore.hpp>
#include <highgui.h>
#include <iostream>
#include <opencv2/imgproc/imgproc.hpp>


using namespace std;
using namespace cv;


void hitmiss(cv::Mat& src, cv::Mat& dst,
    cv::Mat& kernel)
{
  CV_Assert(src.type() == CV_8U && src.
      channels() == 1);

  cv::Mat k1 = (kernel == 1) / 255;
  cv::Mat k2 = (kernel == -1) / 255;

  cv::normalize(src, src, 0, 1, cv::
      NORM_MINMAX);

  cv::Mat e1, e2;
  cv::erode(src, e1, k1, cv::Point
      (-1,-1), 1, cv::BORDER_CONSTANT, cv
      ::Scalar(0));
  cv::erode(1-src, e2, k2, cv::Point
      (-1,-1), 1, cv::BORDER_CONSTANT, cv
      ::Scalar(0));
  dst = e1 & e2;
}

int main()
{
cv::Mat a = (cv::Mat_<uchar>(8,8) << 0,
    0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
    0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0,
    1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
    1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0,
    0, 0, 0 );

cv::Mat b = (cv::Mat_<char>(3,3) << 0,
    1, 0, 1,-1, 1, 0, 1, 0 ); //kernel

hitmiss(a, a, b);
std::cout << a << std::endl;

return 0;
wait

}
```

Obteniendo la respuesta esperada de hit-miss donde localiza los valores de los dos puntos donde existe un 0 central rodeado en norte, sur, este y oeste por valores = 1, como se observa en la Figura 9.

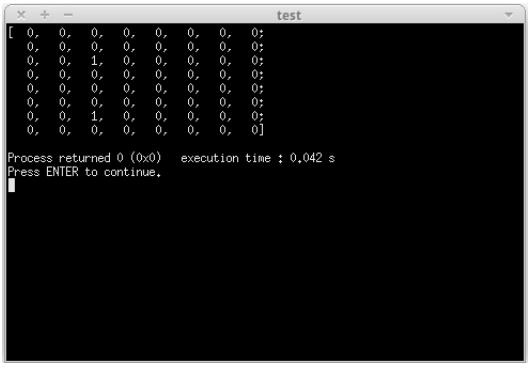


Figura 9: Salida obtenida de hit-miss.

IV. BIBLIOGRAFÍA

- OpenCV, Morphological transformations. (*http://tinyurl.com/jyqwxfo*)
- OpenCV, More morphological transformations. (*http://tinyurl.com/z88oemn*)