PRACTICA 3 - VISIÓN POR COMPUTADOR

Identificación de íconos empleando momentos de Zernike y clasificación de imágenes usando Patrones Binarios Locales (LBP)

Realizado por: Ariel Vazquez John Farez

Objetivo:

Reforzar los conocimientos adquiridos en clase sobre la aplicación de las técnicas de momentos invariantes de HU y los Patrones Binarios Locales (LBP) para tareas de identificación y clasificación de imágenes, respectivamente

> Desarrollar un programa que permita generar un corpus de test y uno de entrenamiento para clasificar imágenes usando los momentos invariantes de HU. Para ello, deberá realizar las siguientes tareas:
>
> Seleccionar 1.700 imágenes de forma aleatoria del corpus ShapeDataset. De estas imágenes deberá usar 1.000 para obtener los momentos de HU para entrenamiento y 700 para test (ningún grupo puede tener las mismas imágenes).

Variables donde se almacenaran las imageens para train y test.

```
const int ntraining=1000;//cantidad de imágenes para el entrenamiento
const int ntest=700;//cantidad de imágenes para la validación
```

Método para generar el conjunto de datos.

```
void listarArchivos(){
    int numero=0;
    std::string elem;
    DIR *direccion;
    struct dirent *elementos;
    if(lista.size() != 0)
        lista.clear();
    if(direccion=opendir(dir.c_str())){
        while(elementos=readdir(direccion)){
            if(std::strlen(elementos->d_name) > 10){
                 lista.push_back(elementos->d_name);
                numero++;
            }
        }
    }
    closedir(direccion);
    std::cout<<"Cantidad archivos encontrados: "<<numero<<"\n";

    std::cout<<"Elementos de la lista"<<"\n";
    std::cout <<lista.size() <<endl;
}

std::vector<String> generarConjunto(int cantidad){
    std::vector<String> conjunto;
```

```
    for(int i=0;i<cantidad;i++){
        int posicion=generarNumeros((int)lista.size());
        conjunto.push_back(lista[posicion]);
    }
    return conjunto;
}
```

Debe calcular los momentos invariantes de HU solo de la forma (Shape) como se muestra en la Ilustración 2:

```
void calcularMomentosTraining(std::vector<String> conjunto){
        // Read image as grayscale image
        for(int i=0;i<conjunto.size();i++){
            String filename = dir+conjunto[i];
            Mat image =imread(filename);
            Mat im;
            cvtColor(image,im,COLOR_BGR2GRAY);

            Mat canny_output;

            threshold(im,im,thresh,thresh*2,3);
            Canny( im, canny_output, thresh, thresh*2, 3 );
            Moments mu = moments(canny_output);

            double huMoments[7];
            HuMoments(mu, huMoments);
            for(int j = 0; j < 7; j++)
                huMoments[j] = -1 * copysign(1.0, huMoments[j]) *
log10(abs(huMoments[j])); //log transforma para tener valores no muy
pequeños
            for(int j=0;j<7;j++){
                training[i][j]=huMoments[j];
            }
            training[i][7]=idFigura(conjunto[i]);
        }
}

void calcularMomentosTest(std::vector<String> conjunto){
        // Read image as grayscale image
        for(int i=0;i<conjunto.size();i++){
            String filename = dir+conjunto[i];
            Mat image =imread(filename);
            Mat im;
            cvtColor(image,im,COLOR_BGR2GRAY);

            Mat canny_output;

            Canny( im, canny_output, thresh, thresh*2, 3 );
```

```
            Moments mu = moments(canny_output);

            double huMoments[7];
            HuMoments(mu, huMoments);
            for(int j = 0; j < 7; j++)
                huMoments[j] = -1 * copysign(1.0, huMoments[j]) *
log10(abs(huMoments[j]));
            for(int j=0;j<7;j++){
                test[i][j]=huMoments[j];
            }
            test[i][7]=idFigura(conjunto[i]);
        }
}
```

Como se observa en la Ilustración 3, dada una de las imágenes originales, se procede a detectar únicamente el área de interés y de dicha área se calcularán los momentos invaritantes de HU. Debe tener presente que no se puede usar la función inRange() ya que los colores de las figuras y los fondos son generados de forma aleatoria y tienen un amplio rango.

```
RNG rng(12345);

void analizarImagen(){
        Mat src = imread(dir+lista[generarNumeros((int)lista.size())]);

        //mostrar imagen original
        imshow( "Imagen", src );

        //transformar a escala de grises
        cvtColor( src, src_gray, COLOR_BGR2GRAY );
        blur( src_gray, src_gray, Size(3,3) );

        //mostrar imagen clon
        imshow( "Clon", src_gray );

        //detección de bordes
        Mat canny_output;
        Canny( src_gray, canny_output, thresh, thresh*2, 3 );
        vector<vector<Point> > contours;
        findContours( canny_output, contours, RETR_TREE,
CHAIN_APPROX_SIMPLE );
        //graficar bordes
        Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
        Scalar color = Scalar( 255,255,255);
        for( size_t i = 0; i< contours.size(); i++ )
            drawContours( drawing, contours,1, color, 2 );
        //mostrar gráfica con bordes
        imshow( "Shape", drawing );
```

```
        //calcular hu moments
        Moments m=moments(contours[0]);
        double huMoments[7];
        HuMoments(m, huMoments);
        std::cout<<"Hu moments: \n";
        for(int j = 0; j < 7; j++)
            huMoments[j] = -1 * copysign(1.0, huMoments[j]) *
log10(abs(huMoments[j])); //log transforma para tener valores no muy
pequeños
        for(int i=0;i<7;i++){
            std::cout<<huMoments[i]<<"     ";
            }
        std::cout<<"\n";
        waitKey();
}
```

Para determinar la categoría de la imágen debe usar la distancia Euclídea (o cualquier métrica similar)

y con ello realizar las comparaciones a ver si las categorías coinciden. Si desea puede emplear técnicas de clustering usando librerías de C++ (a fin de reducir operaciones de cómputo y tiempos)

```
double calcularDistancia(Mat im1, Mat im2){
    return matchShapes(im1, im2, CONTOURS_MATCH_I2, 0);
}
```

5. Debe generar las siguientes métricas de análisis:

```
bool compararImagenes(){
    String filename1 =
conjuntoEntrenamiento[generarNumeros((int)conjuntoEntrenamiento.size())];
    Mat im1 =imread(dir+filename1);
    String filename2 =
conjuntoValidacion[generarNumeros((int)conjuntoValidacion.size())];
    Mat im2 =imread(dir+filename2);

    if( im1.empty() || im2.empty()){
        std::cout<<"\n Error al procesar un archivo \n";
        return false;
    }

    cvtColor(im1,im1,COLOR_BGR2GRAY);
    threshold(im1,im1,thresh,thresh*2,3);

    cvtColor(im2,im2,COLOR_BGR2GRAY);
    threshold(im2,im2,thresh,thresh*2,3);
```

```
    //detección de bordes
    Mat canny_output;
    Canny( im1, canny_output, thresh, thresh*2, 3 );
    vector<vector<Point> > contours;
    findContours( canny_output, contours, RETR_TREE, CHAIN_APPROX_SIMPLE
);
    //graficar bordes
    Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 256), rng.uniform(0,256),
rng.uniform(0,256) );
        drawContours( drawing, contours, (int)i, color, 2 );
    }

    //detección de bordes
    Mat canny_output_;
    Canny( im2, canny_output_, thresh, thresh*2, 3 );
    vector<vector<Point> > contours_;
    findContours( canny_output_, contours_, RETR_TREE,
CHAIN_APPROX_SIMPLE );
    //graficar bordes
    Mat drawing_ = Mat::zeros( canny_output_.size(), CV_8UC3 );
    for( size_t i = 0; i< contours_.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 256), rng.uniform(0,256),
rng.uniform(0,256) );
        drawContours( drawing_, contours_, (int)i, color, 2 );
    }

    imshow( "Im1", drawing );
    imshow( "Im2", drawing_);
    waitKey();

    double d = calcularDistancia(im1,im2);
    std::cout<<"Distancia : "<<d<<"\n";
    float comparaDistancia, comparaValid;
    if(d>0.09){
        std::cout<<"Son figuras distintas"<<"\n";
        comparaDistancia = false;
    }else{
        std::cout<<"Son figuras similares"<<"\n";
        comparaDistancia = true;
    }
    if(idFigura(filename1) == idFigura(filename2))
        comparaValid=true;
    else
        comparaValid=false;
    return (comparaDistancia == comparaValid);
```

```
}
```

1. Programar un método que permita convertir una imagen de un espacio de color en el espacio CIELab.

```
void convertirCielab(){
    Mat src = imread(dir+lista[generarNumeros((int)lista.size())]);
    //mostrar imagen original
    imshow( "Imagen", src );
    //transformar a escala de grises
    Mat src_cielab;
    cvtColor( src, src_cielab, 44);//cv::COLOR_BGR2Lab = 44,
    //mostrar imagen clon
    imshow( "Cielab", src_cielab );
    waitKey();
}
```

2. Programar un método que dada una imagen o región de interés permita calcular el descriptor LBP. Con ello, deberá almacenar el histograma en un archivo o base de datos.

```
void calculoLBP(){
    String filename1 = dir+lista[generarNumeros((int)lista.size())];
    Mat im1 =imread(filename1);
    cvtColor(im1,im1,COLOR_BGR2GRAY);
    threshold(im1,im1,thresh,thresh*2,3);
    Mat lbp=cv::Mat::zeros(im1.rows-2,im1.cols-2, CV_8UC1);
    for(int i=1;i<im1.rows-1;i++){
        for(int j=1;j<im1.cols-1;j++){
            unsigned char center = im1.at<unsigned char>(i,j);
            unsigned char code = 0;
            code |= (im1.at<unsigned char>(i-1,j-1) > center) << 7;
            code |= (im1.at<unsigned char>(i-1,j) > center) << 6;
            code |= (im1.at<unsigned char>(i-1,j+1) > center) << 5;
            code |= (im1.at<unsigned char>(i,j+1) > center) << 4;
            code |= (im1.at<unsigned char>(i+1,j+1) > center) << 3;
            code |= (im1.at<unsigned char>(i+1,j) > center) << 2;
            code |= (im1.at<unsigned char>(i+1,j-1) > center) << 1;
            code |= (im1.at<unsigned char>(i,j-1) > center) << 0;
            lbp.at<unsigned char>(i-1,j-1) = code;
        }
    }
    for(int i=0;i<lbp.cols;i++){
```

```
        for(int j=0;j<lbp.cols;j++){
            std::cout<<lbp.at<unsigned char>(i,j)<<" ";
        }
        std::cout<<"\n";
    }
}
```

```
void calcularHistograma(){
    Mat src = imread(dir+lista[generarNumeros((int)lista.size())]);
    vector<Mat> bgr_planes;
    split( src, bgr_planes );
    int histSize = 256;
    float range[] = { 0, 256 }; //the upper boundary is exclusive
    const float* histRange[] = { range };
    bool uniform = true, accumulate = false;
    Mat b_hist, g_hist, r_hist;
    calcHist( &bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize,
histRange, uniform, accumulate );
    calcHist( &bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize,
histRange, uniform, accumulate );
    calcHist( &bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize,
histRange, uniform, accumulate );
    int hist_w = 512, hist_h = 400;
    int bin_w = cvRound( (double) hist_w/histSize );
    Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );
    normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat()
);
    normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat()
);
    normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat()
);
    for( int i = 1; i < histSize; i++ )
    {
        line( histImage, Point( bin_w*(i-1), hist_h -
cvRound(b_hist.at<float>(i-1)) ),
              Point( bin_w*(i), hist_h - cvRound(b_hist.at<float>(i)) ),
              Scalar( 255, 0, 0), 2, 8, 0  );
        line( histImage, Point( bin_w*(i-1), hist_h -
cvRound(g_hist.at<float>(i-1)) ),
              Point( bin_w*(i), hist_h - cvRound(g_hist.at<float>(i)) ),
              Scalar( 0, 255, 0), 2, 8, 0  );
        line( histImage, Point( bin_w*(i-1), hist_h -
cvRound(r_hist.at<float>(i-1)) ),
              Point( bin_w*(i), hist_h - cvRound(r_hist.at<float>(i)) ),
              Scalar( 0, 0, 255), 2, 8, 0  );
    }
    imshow("imagen de origen", src );
    imshow("Histograma", histImage );
```

```
    waitKey();

}
```

3. Deberá calcular el descriptor LBP para al menos 10 imágenes distintas de dos tipos de textura: clase 1 y clase 2.

```
void calculoLBP(){
    int n=10;
    for(int i=0;i<n;i++){
        String filename1 = dir+lista[generarNumeros((int)lista.size())];
        Mat im1 =imread(filename1);
        cvtColor(im1,im1,COLOR_BGR2GRAY);
        threshold(im1,im1,thresh,thresh*2,3);
        Mat lbp=cv::Mat::zeros(im1.rows-2,im1.cols-2, CV_8UC1);
        for(int i=1;i<im1.rows-1;i++){
            for(int j=1;j<im1.cols-1;j++){
                unsigned char center = im1.at<unsigned char>(i,j);
                unsigned char code = 0;
                code |= (im1.at<unsigned char>(i-1,j-1) > center) << 7;
                code |= (im1.at<unsigned char>(i-1,j) > center) << 6;
                code |= (im1.at<unsigned char>(i-1,j+1) > center) << 5;
                code |= (im1.at<unsigned char>(i,j+1) > center) << 4;
                code |= (im1.at<unsigned char>(i+1,j+1) > center) << 3;
                code |= (im1.at<unsigned char>(i+1,j) > center) << 2;
                code |= (im1.at<unsigned char>(i+1,j-1) > center) << 1;
                code |= (im1.at<unsigned char>(i,j-1) > center) << 0;
                lbp.at<unsigned char>(i-1,j-1) = code;
            }
        }
        for(int i=0;i<lbp.cols;i++){
            for(int j=0;j<lbp.cols;j++){
                std::cout<<lbp.at<unsigned char>(i,j)<<" ";
            }
            std::cout<<"\n";
        }
    }
}
```

4. Dado un nuevo grupo de imágenes deberá calcular la precisión de su clasificador basado en el descriptor LBP para identificar a qué clase pertenece la imagen (clase 1 o clase 2).

```
int idFigura(String fichero){
    int posi=(int)fichero.find("_");
    string nfigura=fichero.substr(0,posi);
    if(nfigura.compare("Circle")==0)
        return 0;
```

```
    else
        if (nfigura.compare("Heptagon")==0)
            return 7;
        else
            if (nfigura.compare("Hexagon")==0)
                return 6;
            else
                if (nfigura.compare("Nonagon")==0)
                    return 9;
                else
                    if (nfigura.compare("Octagon")==0)
                        return 8;
                    else
                        if (nfigura.compare("Pentagon")==0)
                            return 5;
                        else
                            if (nfigura.compare("Square")==0)
                                return 4;
                            else
                                if (nfigura.compare("Star")==0)
                                    return 2;
                                else
                                    if (nfigura.compare("Triangle")==0)
                                        return 3;
    return 0;
}
```

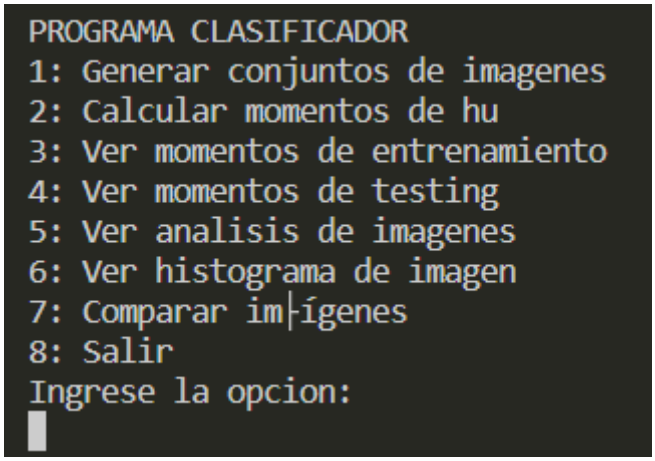
```
ing nombreFigura(int idFigura){
    switch (idFigura)
    {
    case 0:
        return "Circle";
        break;
    case 2:
        return "Star";
        break;
    case 3:
        return "Triangle";
        break;
    case 4:
        return "Square";
        break;
    case 5:
        return "Pentagon";
        break;
    case 6:
        return "Hexagon";
        break;
```

```
    case 7:
        return "Heptagon";
        break;
    case 8:
        return "Octagon";
        break;
    case 9:
        return "Nonagon";
        break;
    default:
        return "Unknown";
        break;
    }
}
```

A continuación, se presentarán los resultados obtenidos en base a los datos seleccionados con anterioridad.

Menú para el desarrollo de la practica considerando que es de forma secuencial es decir primero se generara los datos posteriormente se calculan los momentos y de la misma manera continuar con las otras opciones.

Menú.



Opción 1:

```
PROGRAMA CLASIFICADOR
1: Generar conjuntos de imagenes
2: Calcular momentos de hu
3: Ver momentos de entrenamiento
4: Ver momentos de testing
5: Ver analisis de imagenes
6: Ver histograma de imagen
7: Comparar imágenes
8: Salir
Ingrese la opcion:
1
Cantidad archivos encontrados: 34535
Conjunto de imagenes generado
```

Opción 2.

```
Ingrese la opcion:
2
[ INFO:0@411.689] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\parallel\registry_parall
el.impl.hpp (96) cv::parallel::ParallelBackendRegistry::ParallelBackendRegistry core(parallel): Enabled backends(3
, sorted by priority): ONETBB(1000); TBB(990); OPENMP(980)
[ INFO:0@411.691] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl
.hpp (67) cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\opencv\build\x64\vc15\bin\opencv_core_parallel_
onetbb460_64d.dll => FAILED
[ INFO:0@411.694] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl
.hpp (67) cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_onetbb460_64d.dll => FAILED
[ INFO:0@411.694] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl
.hpp (67) cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\opencv\build\x64\vc15\bin\opencv_core_parallel_
tbb460_64d.dll => FAILED
[ INFO:0@411.697] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl
.hpp (67) cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb460_64d.dll => FAILED
[ INFO:0@411.698] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl.hpp (67
) cv::plugin::impl::DynamicLib::libraryLoad load C:\opencv\opencv\build\x64\vc15\bin\opencv_core_parallel_openmp460_64d.dl
l => FAILED
[ INFO:0@411.701] global c:\build\master_winpack-build-win64-vc15\opencv\modules\core\src\utils\plugin_loader.impl.hpp (67
) cv::plugin::impl::DynamicLib::libraryLoad load opencv_core_parallel_openmp460_64d.dll => FAILED
CALCULADO CORRECTAMENTE
```

Opción 3:

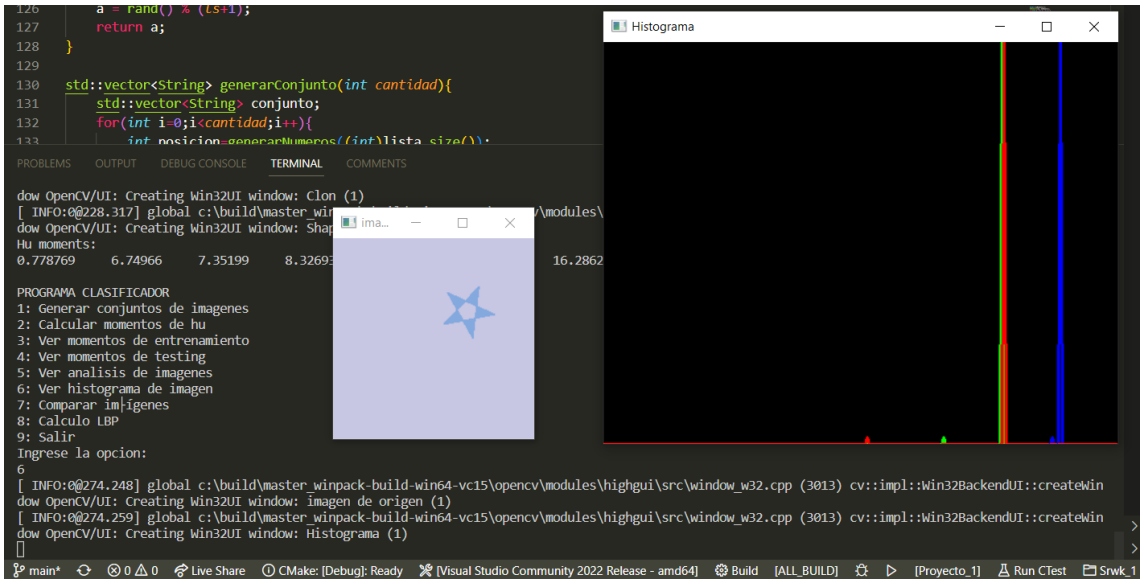
```
Ingrese la opcion:
3
TRAINING: Momentos de hu e identificador
h1      h2      h3      h4      h5      h6      h7      id
2.02    6.95    8.99    9.77    -19.30  13.26   19.31   Circle
1.63    7.72    9.02    9.70    19.11   -13.61  -19.39  Octagon
1.79    7.25    8.27    7.80    15.97   -12.15  16.00   Heptagon
2.27    7.37    9.74    11.04   -21.77  -14.79  21.48   Square
1.58    7.85    9.36    10.28   -20.51  14.29   20.14   Octagon
1.79    7.42    9.77    9.70    -20.59  -13.85  19.43   Octagon
2.12    7.38    9.11    11.27   21.50   15.03   21.85   Hexagon
2.05    7.20    8.66    9.16    18.25   -12.79  -18.19  Star
2.45    7.37    9.43    10.34   20.40   -14.12  -20.35  Star
1.48    7.74    8.82    9.74    19.03   -14.62  -19.56  Square
1.41    6.01    7.03    6.62    13.45   -9.62   14.49   Heptagon
1.92    6.63    8.00    8.75    17.32   -12.08  -17.24  Star
1.53    8.21    9.12    9.98    -19.55  -14.09  19.96   Square
2.10    7.97    10.74   10.39   -21.08  14.37   -21.12  Octagon
1.69    5.64    9.76    10.28   20.59   -13.24  20.37   Hexagon
1.63    8.46    9.63    8.89    -18.33  -13.14  -18.29  Circle
1.51    9.40    8.60    9.18    18.07   14.24   18.96   Circle
1.58    7.83    7.99    7.06    -14.58  11.19   -15.91  Nonagon
1.38    6.20    6.93    6.61    -13.68  10.01   13.44   Heptagon
1.46    6.42    7.13    6.89    -14.93  10.94   13.90   Heptagon
1.75    7.21    5.59    7.94    15.32   -11.78  14.72   Triangle
1.65    7.73    8.19    7.50    -15.63  -11.41  -15.40  Nonagon
1.38    6.00    6.88    6.54    -13.26  9.54    13.94   Heptagon
1.72    8.04    9.08    9.65    -19.24  -14.06  19.11   Circle
1.39    8.05    8.63    8.71    17.78   -12.75  -17.42  Circle
1.64    7.46    8.13    7.40    15.21   13.27   -15.57  Nonagon
1.37    7.82    8.74    9.04    18.01   -13.36  18.18   Octagon
1.81    7.61    9.67    9.79    19.98   13.60   19.55   Octagon
1.62    6.44    8.16    7.35    15.14   10.56   15.47   Pentagon
1.49    5.59    7.52    6.89    -14.32  9.70    14.19   Pentagon
1.85    6.48    8.12    8.02    16.10   -11.29  16.76   Heptagon
1.61    5.60    7.99    7.17    -14.75  9.98    16.04   Pentagon
```

Opción 4:

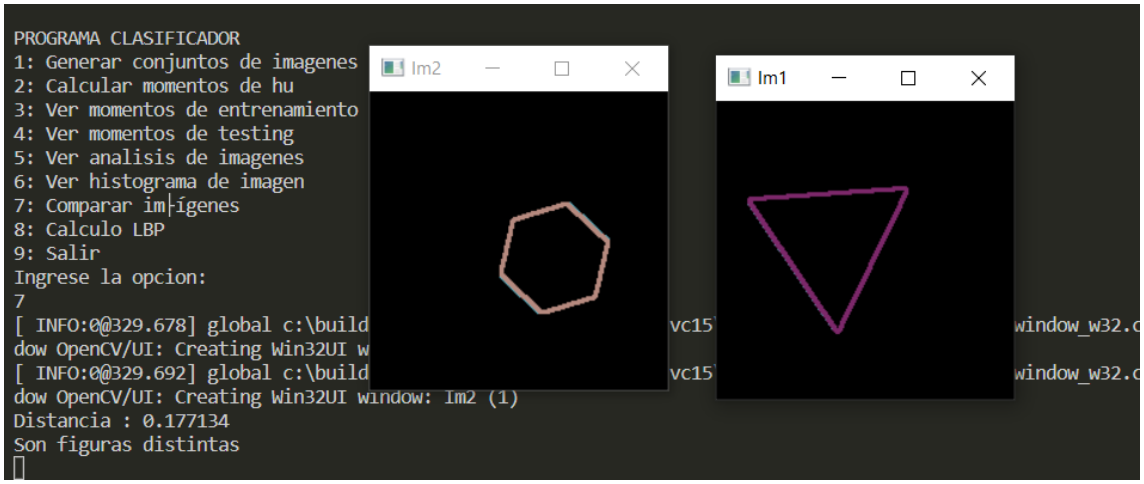
```
8. Salir
Ingrese la opcion:
4
TEST: Momentos de hu e identificador
h1      h2      h3      h4      h5      h6      h7      id
2.08    7.57    9.95    11.00   21.94   -15.16  -21.50  Square
1.38    7.71    8.50    10.13   -20.42  14.33   -19.45  Circle
1.63    9.78    9.29    10.24   -20.01  15.22   -20.84  Square
1.58    5.26    9.87    10.45   -22.66  -13.09  20.62   Hexagon
2.31    7.69    9.35    9.87    19.60   -13.90  -19.68  Star
1.55    5.07    9.73    11.82   22.61   14.53   -23.40  Hexagon
1.58    5.69    9.16    9.72    19.17   12.59   -19.87  Hexagon
1.79    7.15    8.14    7.83    -15.82  11.41   -16.43  Heptagon
1.60    6.56    7.73    7.24    14.85   -10.55  14.91   Heptagon
2.33    7.54    9.54    9.94    -19.69  -13.70  -20.31  Star
1.98    7.70    8.78    9.01    -17.96  -12.87  18.22   Star
2.54    10.13   10.46   10.69   -21.27  -15.78  -23.04  Star
2.08    6.95    8.47    9.21    20.60   -12.69  18.05   Star
1.52    6.51    8.24    7.08    14.76   10.35   -15.21  Pentagon
1.50    8.00    9.73    9.11    -18.91  13.22   -18.56  Circle
1.66    6.43    7.77    7.38    15.81   -10.81  14.96   Heptagon
1.65    6.80    7.81    7.41    -15.13  11.02   15.21   Heptagon
1.54    7.60    8.46    7.32    15.88   11.34   -15.21  Nonagon
1.42    7.20    7.57    6.73    14.86   -10.46  13.89   Nonagon
1.79    6.61    8.11    7.81    15.77   -11.13  16.52   Heptagon
2.64    7.75    10.51   10.78   -21.43  -14.66  22.87   Star
1.49    6.23    7.72    6.92    14.29   10.05   14.62   Pentagon
1.91    8.07    9.08    9.36    18.58   13.47   19.43   Octagon
2.12    7.05    8.74    9.26    -19.67  -12.82  -18.26  Star
1.56    7.90    8.46    9.32    -18.32  -13.41  -18.42  Octagon
1.59    9.63    9.14    10.00   20.88   -15.32  -19.58  Square
2.23    7.51    7.08    9.34    17.93   14.48   -17.60  Triangle
2.16    7.38    6.85    9.35    -19.90  -13.53  -17.44  Triangle
2.39    7.07    9.04    10.24   19.90   -14.15  20.49   Star
2.08    7.31    8.91    9.35    18.79   -13.14  -18.55  Star
1.42    6.16    6.91    6.66    -13.48  9.78    -13.83  Heptagon
2.07    7.78    8.92    8.89    -18.27  12.81   17.82   Nonagon
```

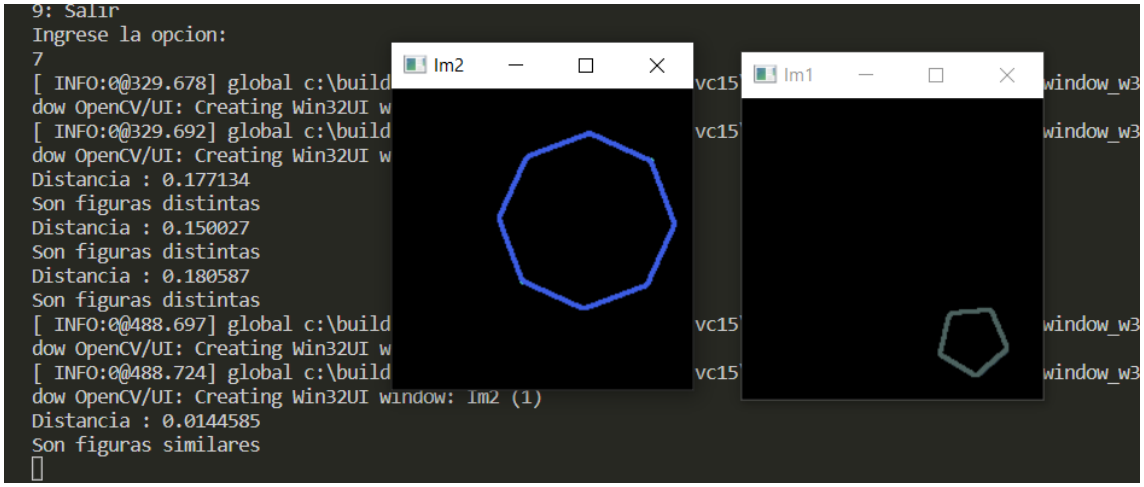
Opción 5:

Opción 6:



Opción 7:





Enlace del código en git.

https://github.com/Jhon14DEA/Practica-3-VC.git