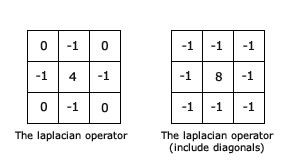
1-Edge Detection

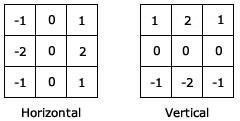
Three type of edge detection used in image processing

* 1-  the Laplacian edge detector uses only one kernel. It calculates second order derivatives in a single pass. Here's the kernel used for it



* //// Laplacian(src\_gray, dst, ddepth, kernel\_size, scale, delta, BORDER\_DEFAULT);
* //Cv2.Laplacian(
* // src //source image
* // , edge, //desination image of the same size and the same number of channels as src
* // MatType.CV\_8U, //desired depth of the destination image
* // 3, //aperture size used to compute the second-derivative filters.
* // 1, //optional scale factor for the computed Laplacian values. By default, no scalling is applied
* // 0, //Optional delta value that is added to the results prior to storing them in dst.
* // BorderTypes.Default //pixel extrapolation method
* // );
* Is more sensitive to noise in image Because we used Guessing filter before used it to remove noise in image

2- Sobel X or Y direction is a gradient based method. It works with first order derivatives. It calculates the first derivatives of the image separately for the X and Y axes.



// //order of the derivative x. when used 1 in x order

////order of the derivative y.when used 1 in y order

// Cv2.Sobel(src, edgex, MatType.CV\_8U, 1, 0, 3, 1, 0, BorderTypes.Default);

// Cv2.Sobel(src, edgey, MatType.CV\_8U, 0, 1, 3, 1, 0, BorderTypes.Default);

// Cv2.BitwiseOr(edgex, edgey, sob, null);

3- Canny is optimal filter used in edge detection Because ti is Mult-steps

**1-Noise Reduction**

**2-Finding Intensity Gradient of the Image**

**3-Non-maximum Suppression**

**4-Hysteresis Thresholding**

**// int ratio = 3;**

**// Mat cann = new Mat();**

**// //{ //Min //Max**

**// Cv2.Canny(src, cann, 100, 100 \* ratio);**

**// //});**

2- BLOB : Blob detector not only finds the boundaries, but also calculates the center and whether or not it matches certain shapes and sizes that you define connected component is a group of pixels that are connected to each other such that each pixel in the group is connected to (or) at least one neighboring pixel belonging to the same group

Preconfition image is gray scale and divide hyparparmetars of algorithem and ti is best if you do threshold image

////# Read image

//Mat srcImage = new Mat($"C:\\Users\\Abd-ElRahman\\OpenCvTest\\bl.jpg", ImreadModes.Color);

//Size s = new Size(3, 3);

//Cv2.ImShow("Source", srcImage);

//Cv2.WaitKey(1); // do events

//var binaryImage = new Mat(srcImage.Size(), MatType.CV\_8UC1);

//Cv2.CvtColor(srcImage, binaryImage, ColorConversionCodes.BGRA2GRAY);

//Cv2.Threshold(binaryImage, binaryImage, thresh: 120, maxval: 255, type: ThresholdTypes.Binary);

//var detectorParams = new SimpleBlobDetector.Params

//{

// //MinDistBetweenBlobs = 10, // 10 pixels between blobs

// //MinRepeatability = 1,

// //MinThreshold = 100,

// //MaxThreshold = 255,

// //ThresholdStep = 5,

// //FilterByArea = false,

// FilterByArea = true,

// MinArea =300, // 10 pixels squared

// MaxArea = 10000,

// FilterByCircularity = false,

// //FilterByCircularity = true,

// //MinCircularity = 0.001f,

// //FilterByConvexity = false,

// FilterByConvexity = true,

// MinConvexity = 0.001f,

// MaxConvexity = 1,

// FilterByInertia = false,

// //FilterByInertia = true,

// //MinInertiaRatio = 0.001f,

// //FilterByColor = false

// FilterByColor = true,

// BlobColor = 0 // to extract light blobs

//};

//var simpleBlobDetector = SimpleBlobDetector.Create(detectorParams);

//var keyPoints = simpleBlobDetector.Detect(binaryImage);

//Console.WriteLine("keyPoints: {0}", keyPoints.Length);

//foreach (var keyPoint in keyPoints)

//{

// Console.WriteLine("X: {0}, Y: {1}", keyPoint.Pt.X, keyPoint.Pt.Y);

//}

//var imageWithKeyPoints = new Mat();

//Cv2.DrawKeypoints(

// image: binaryImage,

// keypoints: keyPoints,

// outImage: imageWithKeyPoints,

// color: Scalar.FromRgb(255, 0, 0),

// flags: DrawMatchesFlags.DrawRichKeypoints);

//Cv2.ImShow("Key Points", imageWithKeyPoints);

//Cv2.WaitKey(1); // do events

//Cv2.WaitKey(0);

//Cv2.DestroyAllWindows();

//srcImage.Dispose();

//imageWithKeyPoints.Dispose();

2-Contours : Contours is in essence the outline of an object

where as blob detector is an algorithm on top of findContours and draw line in edge of image.And draw line in all edge detection of image

Precondition Must Image is gray Scale and threshold image

// Mat srcImage = new Mat($"C:\\Users\\Abd-ElRahman\\OpenCvTest\\bl.jpg", ImreadModes.Color);

// Mat src = new Mat($"C:\\Users\\Abd-ElRahman\\OpenCvTest\\bl.png", ImreadModes.Color);

// Size s = new Size(512, 512);

// //Size dim2 = new Size();

//Cv2.Resize(srcImage, srcImage, s, 0, 0, InterpolationFlags.LinearExact);

// var binaryImage = new Mat(srcImage.Size(), MatType.CV\_8UC1);

// Cv2.CvtColor(srcImage, binaryImage, ColorConversionCodes.BGRA2GRAY);

// //Cv2.Threshold(binaryImage, binaryImage, thresh: 20, maxval: 255, type: ThresholdTypes.Binary);

// // //Min //Max

// Cv2.Canny(binaryImage, binaryImage, 20, 255);

// //

// Cv2.FindContours(binaryImage, out Point[][] contours, out HierarchyIndex[] hierarchy, RetrievalModes.Tree, ContourApproximationModes.ApproxNone);

// Console.WriteLine("number on contours =" + Convert.ToString(contours));

// foreach (Point[] conter in contours)

// {

// Point[] outk= Cv2.ApproxPolyDP(conter, .01, true);

// Cv2.DrawContours(srcImage, outk, -1, Scalar.Green, 1);

// }

// using (new Window("Image", srcImage))

// using (new Window("thres", binaryImage))

// {

// Cv2.WaitKey(0);

// }

3- Implement Circle/Square/Rectangle in images detector :we used Contour to cal all contour and image and apple condition to detect rect , square , triangle ,circle

Mat gray = new Mat($"C:\\Users\\Abd-ElRahman\\OpenCvTest\\shaps.png", ImreadModes.Grayscale);

Mat src = new Mat($"C:\\Users\\Abd-ElRahman\\OpenCvTest\\shaps.png", ImreadModes.Color);

//Blurring to reduce high frequency noise to make our contour detection process more accurate.

Mat blurred = new Mat();

blurred = gray.GaussianBlur(new Size(5, 5), 0);

//Binarization of the image.

Mat threshold = new Mat();

threshold = blurred.Canny(60, 255);

//find contours

Point[][] contours;

HierarchyIndex[] hierarchyIndexes;

Cv2.FindContours(

threshold,

out contours,

out hierarchyIndexes,

RetrievalModes.Tree,

ContourApproximationModes.ApproxNone);

Cv2.DrawContours(src, contours, -1, Scalar.Green, 3);

//loop over the contours

foreach (var c in contours)

{

Moments m = Cv2.Moments(c);

Point pnt = new Point(m.M10 / m.M00, m.M01 / m.M00); //center point

Cv2.Circle(src, pnt, 5, Scalar.Red, -1);

string shape =Shap.GetShape(c);

Cv2.PutText(src,shape, pnt, HersheyFonts.HersheySimplex, 0.5, Scalar.Green, 2);

}

using (new Window("src", src))

using (new Window("threshold", threshold))

{

Cv2.WaitKey(0);

}

Function used to Getshap

public static string GetShape(Point[] c)

{

string shape = "unidentified";

double peri = Cv2.ArcLength(c, true);

Point[] approx = Cv2.ApproxPolyDP(c, 0.01 \* peri, true);

if (approx.Length == 3) //if the shape is a triangle, it will have 3 vertices

{

shape = "triangle";

}

else if (approx.Length == 4) //if the shape has 4 vertices, it is either a square or a rectangle

{

Rect rect;

rect = Cv2.BoundingRect(approx);

double ar = rect.Width / (double)rect.Height;

if (ar >= 0.95 && ar <= 1.05) shape = "square";

else shape = "rectangle";

}

else if (approx.Length == 5) //if the shape has 5 vertice, it is a pantagon

{

shape = "pentagon";

}

else //otherwise, shape is a circle

{

shape = "circle";

}

return shape;

}