Tradución al Castellano del Tutorial de OpenCV.js en Internet

Enlace al tutorial original en Inglés:

https://docs.opencv.org/4.x/d5/d10/tutorial_js_root.html

Realizado por:

Vicente Soriano Ocheda

Ingeniero Industrial por la ETSIT de la UPV

Dr Ingeniero de Telecomunicación por la ESIT de la UPV

A mi mujer y a mi hija con todo el amor...

Índice

Prólogo

Escribiré yo mismo el prólogo de este libro para poder subirlo ya a algún dominio y que se pueda descargar.

Decir que he disfrutado traduciendo las páginas del Tutorial Oficial de Opencv 4.6.0 porque al mismo tiempo he podido aprender y ver las maravillas que se pueden hacer con los ordenadores. La programación mediante una librería para procesamiento de imágenes, concretamente OpenCV, ha resultado finalmente fantaśtica en los tres lenguajes que me he permitdo trabajar. Primemro con C++, rápido pero aconsejado para usuarios de este lenguaje, luego con Python, fantástico como lenguaje para ivestigación pero también para los sque son sólo aficionados a la programación y finalmente con JavaScript, de la que solo diré que es la niña de mis ojos.

Espero que los hablantes de lengua castellana encuentren en esta traducción una forma de meterse en el tema y darle vida al gusanillo que todos tenemos dentro.

Saludos a todo el mundo

Que tengan todos un feliz día

Vicente Soriano

1.- Introducción a OpenCV.js

¡Aprende a usar OpenCV.js dentro de tus páginas web!

1.1.- Introducción a OpenCV.js y Tutoriales

Introducción a OpenCV.js y Tutoriales

1.1.1.- OpenCV

OpenCV fue creado en Intel en 1999 por Gary Bradski. El primer lanzamiento salió en el año 2000. Vadim Pisarevsky se unió a Gary Bradski para administrar el equipo OpenCV de software ruso de Intel. En 2005, se utilizó OpenCV en Stanley; el vehículo que ganó el DARPA Grand Challenge 2005. Posteriormente, su desarrollo activo continuó bajo el apoyo de Willow Garage, con Gary Bradski y Vadim Pisarevsky al frente del proyecto. OpenCV ahora es compatible con una multitud de algoritmos relacionados con la visión artificial y el aprendizaje automático y se está expandiendo día a día.

OpenCV es compatible con una amplia variedad de lenguajes de programación, como C, C++, Python , Java y JavaScript y, está disponible en diferentes plataformas, incluidas Windows, Linux, OS X, Android e iOS. Las interfaces para operaciones de GPU de alta velocidad basadas en CUDA y OpenCL también están en desarrollo activo. OpenCV.js lleva OpenCV a la plataforma web abierta y lo pone a disposición del programador de JavaScript.

1.1.2.- OpenCV.js: OpenCV para el programador JavaScript

La web es la plataforma informática abierta más omnipresente. Con los estándares HTML5 implementados en cada navegador, las aplicaciones web pueden reproducir videos en línea con etiquetas de video HTML5, capturar videos de cámaras web a través de la API WebRTC y acceder a cada píxel de un cuadro de video a través de la API de canvas. Con la abundancia de contenido multimedia disponible, los desarrolladores web necesitan una amplia gama de algoritmos de procesamiento de imágenes y visión en JavaScript para crear aplicaciones innovadoras. Este requisito es aún más esencial para las aplicaciones emergentes en la web, como Web Virtual Reality (WebVR) y Augmented Reality (WebAR). Todos estos casos de uso exigen implementaciones eficientes de núcleos de visión de computación intensiva en la web.

Emscripten es un compilador de LLVM a JavaScript. Toma el código de bits LLVM, que se puede generar a partir de C/C++ usando clang, y lo compila en asm.js o WebAssembly que puede ejecutarse directamente dentro de los navegadores web. Asm.js es un subconjunto de JavaScript de bajo nivel altamente optimizable. Asm.js permite la compilación y optimización anticipadas en el motor de JavaScript que proporciona una velocidad de ejecución cercana a la nativa. WebAssembly es un nuevo formato binario portátil, eficiente en tamaño y tiempo de carga, adecuado para la compilación en la web. WebAssembly tiene como objetivo ejecutarse a velocidad nativa. WebAssembly está siendo diseñado actualmente como un estándar abierto por W3C.

OpenCV.js es un enlace de JavaScript para un subconjunto seleccionado de funciones de OpenCV para la plataforma web. Permite que las aplicaciones web emergentes con procesamiento

multimedia se beneficien de la amplia variedad de funciones de visión disponibles en OpenCV. OpenCV.js aprovecha Emscripten para compilar funciones de OpenCV en objetivos asm.js o WebAssembly, y proporciona una API de JavaScript para que la aplicación web acceda a ellos. Las futuras versiones de la biblioteca aprovecharán las API de aceleración que están disponibles en la web, como SIMD y la ejecución de subprocesos múltiples.

OpenCV.js se creó inicialmente en Parallel Architectures and Systems Group en la Universidad de California Irvine (UCI) como un proyecto de investigación financiado por Intel Corporation. OpenCV.js se mejoró aún más y se integró en el proyecto OpenCV como parte del programa Google Summer of Code 2017.

1.1.3.- Tutoriales de OpenCV.js

OpenCV presenta un nuevo conjunto de tutoriales que lo guiarán a través de varias funciones disponibles en OpenCV.js. Esta guía se centra principalmente en la versión OpenCV 4.x.

El propósito de los tutoriales de OpenCV.js es:

1. Ayudar con la adaptabilidad de OpenCV en el desarrollo web
2. Ayudar a la comunidad web, los desarrolladores y los investigadores de visión por computadora a acceder de forma interactiva a una variedad de ejemplos de OpenCV basados en la web para mejorar la comprensión de los algoritmos de visión específicos.

Debido a que OpenCV.js puede ejecutarse directamente dentro del navegador, las páginas web del tutoriales de OpenCV.js son intuitivas e interactivas. Por ejemplo, el uso de la API de WebRTC y cambiar el código JavaScript en el editor de la página, permite a los desarrolladores cambiar los parámetros de las funciones de CV y realizar la codificación de CV en vivo en las páginas web para ver los resultados en tiempo real. Mas adelante veremos como podemos descargar las páginas del tutorial de OpenCV.js en nuestra computadora y ejecutar los localmente.

Se recomienda tener conocimientos previos de JavaScript y desarrollo de aplicaciones web para comprender esta guía.

1.1.4.- Colaboradores

A continuación se muestra la lista de colaboradores de tutoriales de OpenCV.js.

- Sajjad Taheri (Architect of the initial version and GSoC mentor, University of California, Irvine)
- Congxiang Pan (GSoC student, Shanghai Jiao Tong University)
- Gang Song (GSoC student, Shanghai Jiao Tong University)
- Wenyao Gan (Student intern, Shanghai Jiao Tong University)
- Mohammad Reza Haghighat (Project initiator & sponsor, Intel Corporation)
- Ningxin Hu (Students' supervisor, Intel Corporation)

1.2.- Usando OpenCV.js

Comenzando con OpenCV.js

1.2.1.- Descargar la librería opencv.js

En este tutorial, aprenderá cómo incluir y comenzar a usar opencv.js dentro de una página web. Puede obtener una copia de opencv.js en su última versión 4.6.0 (agosto 2022) en el siguiente enlace:

opencvjs-4.6.0

Cuando se habra la pagina web con el código de opencv.js seleccione guardar como y guarde con el nombre opencv.js.

1.2.2.- Crear una página web que carge una imagen

Antes de empezar con OpenCV.js crearemos una página sencilla que carge una imagen en una etiqueta img mediente un campo **input** de tipo **file**. El código se muestra a continuación:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cargar una imagen</title>
</head>
<body>
<h2>Cargar una imagen</h2>
<div>
  <div class="inputoutput">
    <img id="imageSrc" alt="No Image" />
    <div class="caption">
      imageSrc <input type="file" id="fileInput" name="file"/>
    </div>
  </div>
</div>
<script type="text/javascript">
  // creamos el elemento del tag <img>
  let imgElement = document.getElementById("imageSrc")

  // creamos el elemento del tag <input>
  let inputElement = document.getElementById("fileInput");

  // al cambiar el valor del <input> colocamos la nueva imagen
  inputElement.addEventListener("change", (e) => {
    imgElement.src = URL.createObjectURL(e.target.files[0]);
  }, false);
</script>
</body>
</html>
```

1021_cargar_imagen.html

Para ejecutar esta página web, copie el contenido anterior y guárdelo en un archivo index.html local. Para ejecutarlo, ábralo usando su navegador web.

Cuando se cargue la página anterior aparecerá un selector de archivos que nos permitirá elegir una imagen y mostrarla en el tag **img** con **id=”imageSrc”** del archivo o página HTML**.**

Nota

Aunque de momento no hemos utilizado **opencv.js** en documentos html, en el resto del tutorial alojamos opencv.js y el resto de archivos que necesitemos (imágenes, ficheros de estilo, etc) en la carpeta **assets** en la raiz del directorio donde se ubican todos los ficheros html. Seguramente se denominará este directorio **LIBRO** pero puede cambiar para ser más explícito.

1.2.3.- Visualizar una imagen en un canvas mediante opencv.js

El siguiente documento cargamos una imagen en un elemento **img** y cuando la carga se ha realizado con éxtito, leemos la imagen del img con OpenCV y la mostramos en un canvas también con OpenCV.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Visualizar una imagen en un canvas utilizando OpenCV</title>
<link rel="stylesheet" href="assets/js_example_style.css">
</head>
<body>
<h2>Visualizar una imagen en un canvas utilizando OpenCV</h2>
<p id="status">OpenCV.js se está cargando...</p>
<div>
  <table cellpadding="0" cellspacing="0" width="0" border="0">
  <tr>
      <td>
          <img id="imageSrc">
      </td>
      <td>
          <canvas id="canvasOutput"></canvas>
      </td>
  </tr>
  <tr>
      <td>
          <div class="caption">imgInput
            <input type="file"
                   id="fileInput"
                   name="file"
                   accept="image/*" />
          </div>
      </td>
      <td>
          <div class="caption">canvasOutput</div>
      </td>
  </tr>
  </table>
</div>
<script type="text/javascript">

// manipulación del HTML
let imgElement = document.getElementById('imageSrc');
let inputElement = document.getElementById('fileInput');
inputElement.addEventListener('change', (e) => {
```

```
    imgElement.src = URL.createObjectURL(e.target.files[0]);
}, false);

// OpenCV
imgElement.onload = function() {
    let mat = cv.imread(imgElement);
    cv.imshow('canvasOutput', mat);
    mat.delete();
};
// Se ejecuta cuando OpenCv está cargada y preparada
function onOpenCvReady() {
    document.getElementById('status').innerHTML =
     '<b>OpenCV.js está preparada</b>.' +
      'Puede cargar una imagen.<br>'
}
// Se ejecuta cuando se produce un error durante el proceso
// de carga de OpenCv
function onOpenCvError() {
    let element = document.getElementById('status');
    element.setAttribute('class', 'err');
    element.innerHTML = 'Falló la carga de opencv.js';
}
</script>
<script async src="assets/opencv.js"
              type="text/javascript"
              onload="onOpenCvReady();"
              onerror="onOpenCvError();">
</script>
</body>
</html>
```

1022_visualizar_imagen.html

Como ya hemos mencionado antes, la carpeta **assets** es donde ubicaremos normalmente los ficheros auxiliares que necesite nuestro programa. En este caso almacenamos el fichero **js_example_style.css**, que OpenCV utiliza a lo largo de todos sus tutoriales y también la propia librería **opencv.js** y algunas imágenes. Esta carpeta irá creciendo en archivos a medida que avancemos en el tutorial.

Vemos al final del documento la carga asíncrona de opencv.js. No siempre funcionará este método de carga, tal cual está planteado, como veremos más adelante.

En las siguientes líneas,

```
// OpenCV
imgElement.onload = function() {
    let mat = cv.imread(imgElement);
    cv.imshow('canvasOutput', mat);
    mat.delete();
}
```

Hacemos lo siguiente:

- imgElement.onload → espera hasta que la imagen esté cargada con éxito

- let mat = cv.imread(imgElement) → leemos la imagen asociada a imgElement y la asignamos a la matriz mat
- cv.imshow('canvasOutput', mat) → mostramos la matriz mat en el canvasOutput
- mat.delete() → borramos la matriz mat

1.3.- Construyendo OpenCV.js

Obteniendo opencv.js

En el tutorial de OpenCV.js se explica como construir el fichero **opencv.js** partiendo del reposotorio git de opencv e instalando previamente Emscripten. Es un proceso un poco laborioso y se puede seguir desde este enlace.

No obstante, dicho fichero (la versión 4.6.0) lo podemos obtener facilmente junto al resto de ficheros que acompañan este tutorial.

Otra forma de obtener el fichero es ejecutando “Guardar como” desde el navegador en cualquier página del tutorial web de OpenCV.js que utilice la librería opencv.js. En la carpeta que se guarde, junto al fichero HTML, tendremos el fichero opencv.js

1.4.- Usando OpenCV.js en Node.js

Utilizando OpenCV.js en Node.js

1.4.1.- Objetivos

En este tutorial, aprenderemos a:

- Utilizar OpenCV.js en una aplicación Node.js
- Cargue imágenes con jimp para usarlas con OpenCV.js
- Uso de jsdom y node-canvas para admitir cv.imread() y cv.imshow()
- Los conceptos básicos de las API emscripten, como Module y File System en los que se basa OpenCV.js para Node.js.
- Aprender los conceptos básicos de Node.js. Aunque este tutorial asume que el usuario conoce JavaScript, no se requiere experiencia con Node.js.

  Nota

  Además de dar instrucciones para ejecutar OpenCV.js en Node.js, otro objetivo de este tutorial es presentar a los usuarios los conceptos básicos de las API de emscripten, como Módulo y Sistema de archivos y también de Node.js.

1.4.2.- Ejemplo mínimo

Cree el directorio opencv-node

En su interior, cree la carpeta proyecto1 y dentro cree un archivo ejemplo10.js con el siguiente contenido:

```
// Define una variable global 'Module' con el método 'onRuntimeInitialized':
Module = {
  onRuntimeInitialized() {
    // esta es nuestra aplicación
    console.log(cv.getBuildInformation())
  }
}
// Carga ‘opencv.js‘ asignando su valor a la variable global ‘cv’
cv = require('./opencv.js')
```

opencv-node/proyecto1/ejemplo10.js

Para **ejecutar el programa** realice lo siguiente:

- Guarde el archivo como ejemplo10.js
- Asegúrese de que el archivo opencv.js esté en la misma carpeta
- Asegúrese de que Node.js esté instalado en su sistema

El siguiente comando, ejecutado desde el directorio proyecto1, debería imprimir la información de compilación de OpenCV:

```
    node ejemplo10.js
```

```
...
 OpenCV modules:
   To be built:                 calib3d core dnn features2d flann imgproc js objdetect photo video
   Disabled:                    highgui imgcodecs ml stitching videoio world
   Disabled by dependency:      ts
   Unavailable:                 gapi java python2 python3
   Applications:                examples
   Documentation:               js
   Non-free algorithms:         NO

  GUI:

  Media I/O:
    ZLib:                      build (ver 1.2.12)
    JPEG 2000:                 build (ver 2.4.0)
    HDR:                       YES
    SUNRASTER:                 YES
    PXM:                       YES
    PFM:                       YES

  Video I/O:

  Parallel framework:          none

  Other third-party libraries:
    VA:                        NO
```

```
  Custom HAL:                  NO
  Protobuf:                    build (3.19.1)

Python (for build):            /usr/bin/python

Install to:                    /usr/local
-----------------------------------------------------------------
```

¿Qué ha sucedido al ejecutar el prograna?

En la primera declaración: al definir una variable global llamada 'Módule', emscripten llamará a Module.onRuntimeInitialized() cuando la librería opencv.js esté lista para usar. Nuestro programa está en ese método y usa la variable global cv como en el navegador.

La sentencia cv = require('./opencv.js') requiere el archivo opencv.js y asigna el valor de retorno a la variable global cv. require() es una API de Node.js y se usa para cargar módulos y archivos. En este caso, cargamos el archivo opencv.js desde la carpeta actual y, como se dijo anteriormente, emscripten llamará a Module.onRuntimeInitialized() cuando esté listo.

Consulte la API del módulo emscripten para obtener más detalles.

1.4.3.- Trabajo con imágenes

OpenCV.js no admite formatos de imagen, por lo que no podemos cargar imágenes png o jpeg directamente. En el navegador utiliza el HTML DOM (como HTMLCanvasElement y HTMLImageElement para codificar y decodificar imágenes). En node.js necesitaremos usar una biblioteca para esto.

En este ejemplo, usamos **jimp**, que admite formatos de imagen comunes y es bastante fácil de usar.

Configuración del ejemplo

Desde el directorio opencv-node, ejecute los siguientes comandos para crear un nuevo paquete node.js e instalar la dependencia de jimp:

```
cd proyecto1
npm init -y
npm install jimp
```

El ejemplo11.js

```
const Jimp = require('jimp');
async function onRuntimeInitialized(){
  // carga el archivo de imagen local con jimp.
  // Admite jpg, png, bmp, tiff y gif:
  var jimpSrc = await Jimp.read('./lena.jpg');
  // La propiedad `jimpImage.bitmap` tiene el ImageData decodificado
  // que podemos usar para crear un cv:Mat
  var src = cv.matFromImageData(jimpSrc.bitmap);
  // Las siguientes líneas son copiar y pegar del tutorial de dilatación
  // de opencv.js:
  let dst = new cv.Mat();
  let M = cv.Mat.ones(5, 5, cv.CV_8U);
  let anchor = new cv.Point(-1, -1);
  cv.dilate(src, dst, M, anchor, 1, cv.BORDER_CONSTANT,
```

```
            cv.morphologyDefaultBorderValue());
  // Ahora que hemos terminado, queremos escribir `dst` en el archivo
  // `output.png`. Para ello creamos una imagen `Jimp`que acepta el image
  // data de la imagen dst como un Buffer:
  // [`Buffer`](https://nodejs.org/docs/latest-v10.x/api/buffer.html).
  // `write('output.png')` lo escribirá en el disco y Jimp deduce el formato
  // de salida del nombre de archivo dado
  new Jimp({
    width: dst.cols,
    height: dst.rows,
    data: Buffer.from(dst.data)
  })
  .write('output.png');
  src.delete();
  dst.delete();
}
// Finalmente, cargue open.js como antes. La función `onRuntimeInitialized`
// contiene nuestro programa.
Module = {
  onRuntimeInitialized
};
cv = require('./opencv.js');
```

ejemplo11.js

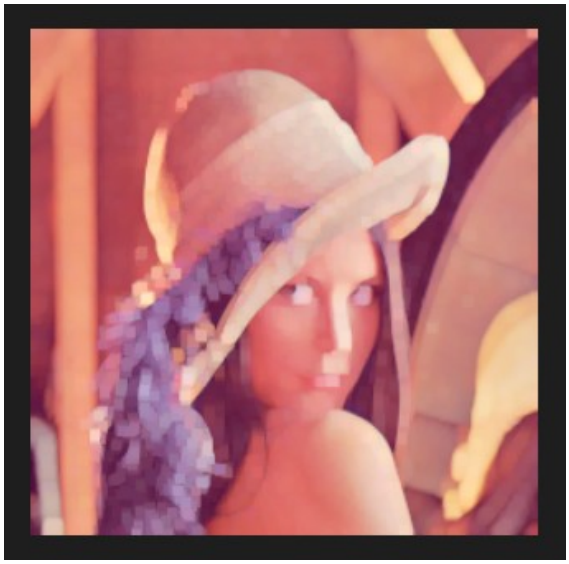
Ejecute el ejemplo

Guarde el archivo como ejemplo11.js.

Asegúrese de que exista una imagen de muestra lena.jpg en el directorio actual.

El siguiente comando debería generar el archivo output.png:

```
node ejemplo11.js
```



output.png

(el margen negro es añadido al copiar la imagen con **Shutter**)

1.4.4.- Emular HTML DOM y canvas

Como antes, desde el directorio opencv-node, creamos un nuevo proyecto Node.js e instalamos las dependencias que necesitamos:

```
mkdir proyecto2
cd proyecto2
npm init -y
npm install canvas jsdom
```

El código del ejemplo20.js

```
const { Canvas, createCanvas, Image, ImageData, loadImage } =
require('canvas');
const { JSDOM } = require('jsdom');
const { writeFileSync, existsSync, mkdirSync } = require("fs");
// Este es nuestro programa. Esta vez usamos JavaScript async/await y
// promesas para manejar la asincronía.
(async () => {
  // Antes de cargar opencv.js emulamos un HTML DOM mínimo. Consulte la
  // declaración de la función que viene a continuación.
  installDOM();
  await loadOpenCV();
  // Usando node-canvas, convertimos un archivo de imagen en un objeto
  // compatible con HTML DOM Image y, por lo tanto, con cv.imread()
  const image = await loadImage('./lena.jpg');
  const src = cv.imread(image);
  const dst = new cv.Mat();
  const M = cv.Mat.ones(5, 5, cv.CV_8U);
  const anchor = new cv.Point(-1, -1);
  cv.dilate(src, dst, M, anchor, 1, cv.BORDER_CONSTANT,
            cv.morphologyDefaultBorderValue());
  // Creamos un objeto compatible HTMLCanvasElement
  const canvas = createCanvas(300, 300);
  cv.imshow(canvas, dst);
  writeFileSync('output.jpg', canvas.toBuffer('image/jpeg'));
  src.delete();
  dst.delete();
})();
// Cargamos opencv.js como antes, pero usando Promise en lugar de callback:
function loadOpenCV() {
  return new Promise(resolve => {
    global.Module = {
      onRuntimeInitialized: resolve
    };
    global.cv = require('./opencv.js');
  });
}
// Using jsdom and node-canvas we define some global variables to emulat
// HTML DOM.
// Although a complete emulation can be archived, here we only define those
// globals used by cv.imread() and cv.imshow().
function installDOM() {
  const dom = new JSDOM();
  global.document = dom.window.document;
  // El resto habilita DOM image y canvas y es proporcionado por node-canvas
  global.Image = Image;
  global.HTMLCanvasElement = Canvas;
  global.ImageData = ImageData;
```

```
  global.HTMLImageElement = Image;
}
```

ejemplo20.js

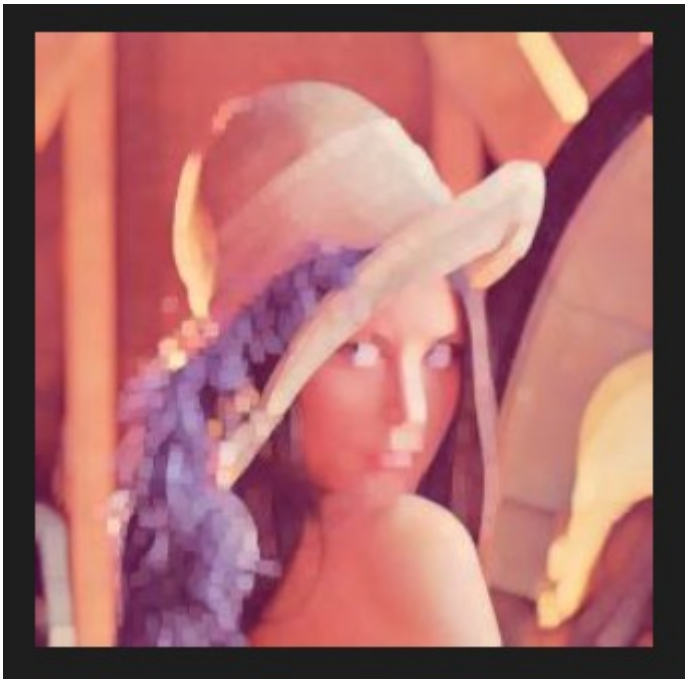
Ejecute el ejemplo

Guarde el archivo como ejemplo20.js.

Asegúrese de que exista una imagen de muestra lena.jpg en el directorio actual.

El siguiente comando debería generar el archivo output.jpg:

```
node ejemplo20.js
```



output.jpg

1.4.5.- Tratar con archivos

En este tutorial, aprenderá cómo configurar emscripten para que use el sistema de archivos local para operaciones de archivos en lugar de usar la memoria. También trata de describir cómo son soportados los archivos por las aplicaciones emscripten.

El acceso al sistema de archivos emscripten a menudo es necesario en las aplicaciones OpenCV, por ejemplo, para cargar modelos de aprendizaje automático como los que se usan en los modelos del marco Load Caffe y Cómo ejecutar redes profundas en el navegador.

Configuración de ejemplo30.js

Antes del ejemplo, vale la pena considerar primero cómo se manejan los archivos en aplicaciones emscripten como OpenCV.js. Recuerde que la biblioteca OpenCV está escrita en C++ y el archivo opencv.js es solo ese código C++ traducido a JavaScript o WebAssembly por el compilador emscripten C++.

Estas fuentes C++ usan APIs estándar para acceder al sistema de archivos y la implementación a menudo termina en llamadas al sistema que leen un archivo en el disco duro. Dado que las aplicaciones de JavaScript en el navegador no tienen acceso al sistema de archivos local, emscripten emula un sistema de archivos estándar para que el código compilado C++ funcione fuera de la caja.

En el navegador, este sistema de archivos se emula en la memoria, mientras que en Node.js también existe la posibilidad de usar el sistema de archivos local directamente. Esto suele ser preferible ya que no es necesario copiar el contenido del archivo en la memoria. Esta sección explica cómo hacer exactamente eso, esto es, configurar emscripten para que se acceda a los archivos directamente desde nuestro sistema de archivos local y las rutas relativas coincidan con los archivos relativos al directorio local actual como se esperaba.

Al igual que en apartado anterior, desde el directorio opencv-node cree un nuevo proyecto Node.js e instale las dependencias que necesitamos:

```
mkdir proyecto3
cd proyecto3
npm init -y
npm install canvas jsdom fs
```

El ejemplo

La siguiente es una adaptación de Face Detection using Haar Cascades.

Código del ejemplo

```
const { Canvas, createCanvas, Image, ImageData, loadImage } =
        require('canvas');
const { JSDOM } = require('jsdom');
const { writeFileSync, readFileSync, existsSync } = require('fs');

(async () => {
  await loadOpenCV();
  const image = await loadImage('lena.jpg');
  const src = cv.imread(image);
  let gray = new cv.Mat();
  cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
  let faces = new cv.RectVector();
  let eyes = new cv.RectVector();
  let faceCascade = new cv.CascadeClassifier();
  let eyeCascade = new cv.CascadeClassifier();
  // Load pre-trained classifier files. Notice how we reference local files
  // using relative paths just like we normally would do
  faceCascade.load('./haarcascade_frontalface_default.xml');
  eyeCascade.load('./haarcascade_eye.xml');
  let mSize = new cv.Size(0, 0);
  faceCascade.detectMultiScale(gray, faces, 1.1, 3, 0, mSize, mSize);
  for (let i = 0; i < faces.size(); ++i) {
    let roiGray = gray.roi(faces.get(i));
    let roiSrc = src.roi(faces.get(i));
    let point1 = new cv.Point(faces.get(i).x, faces.get(i).y);
    let point2 = new cv.Point(faces.get(i).x + faces.get(i).width,
faces.get(i).y + faces.get(i).height);
    cv.rectangle(src, point1, point2, [255, 0, 0, 255]);
    eyeCascade.detectMultiScale(roiGray, eyes);
    for (let j = 0; j < eyes.size(); ++j) {
      let point1 = new cv.Point(eyes.get(j).x, eyes.get(j).y);
      let point2 = new cv.Point(eyes.get(j).x + eyes.get(j).width,
eyes.get(j).y + eyes.get(j).height);
      cv.rectangle(roiSrc, point1, point2, [0, 0, 255, 255]);
    }
    roiGray.delete();
    roiSrc.delete();
  }
  const canvas = createCanvas(image.width, image.height);
  cv.imshow(canvas, src);
  writeFileSync('output3.jpg', canvas.toBuffer('image/jpeg'));
  src.delete(); gray.delete(); faceCascade.delete(); eyeCascade.delete();
faces.delete(); eyes.delete()
})();
/**
```

```
 * Loads opencv.js.
 *
 * Installs HTML Canvas emulation to support `cv.imread()` and `cv.imshow`
 *
 * Mounts given local folder `localRootDir` in emscripten filesystem folder
`rootDir`. By default it will mount the local current directory in emscripten
`/work` directory. This means that `/work/foo.txt` will be resolved to the
local file `./foo.txt`
 * @param {string} rootDir The directory in emscripten filesystem in which
the local filesystem will be mount.
 * @param {string} localRootDir The local directory to mount in emscripten
filesystem.
 * @returns {Promise} resolved when the library is ready to use.
 */
function loadOpenCV(rootDir = '/work', localRootDir = process.cwd()) {
  if(global.Module && global.Module.onRuntimeInitialized && global.cv &&
global.cv.imread) {
    return Promise.resolve()
  }
  return new Promise(resolve => {
    installDOM()
    global.Module = {
      onRuntimeInitialized() {
        // We change emscripten current work directory to 'rootDir' so
        // relative paths are resolved
        // relative to the current local folder, as expected
        cv.FS.chdir(rootDir)
        resolve()
      },
      preRun() {
        // preRun() is another callback like onRuntimeInitialized() but is
        // called just before the
        // library code runs. Here we mount a local folder in emscripten
        // filesystem and we want to do this before the library is executed
        // so the filesystem is accessible from the start
        const FS = global.Module.FS
        // create rootDir if it doesn't exists
        if(!FS.analyzePath(rootDir).exists) {
          FS.mkdir(rootDir);
        }
        // create localRootFolder if it doesn't exists
        if(!existsSync(localRootDir)) {
          mkdirSync(localRootDir, { recursive: true});
        }
        // FS.mount() is similar to Linux/POSIX mount operation. It basically
        // mounts an external filesystem with given format, in given current
        // filesystem directory.
        FS.mount(FS.filesystems.NODEFS, { root: localRootDir}, rootDir);
      }
    };
    global.cv = require('./opencv.js')
  });
}
function installDOM(){
  const dom = new JSDOM();
  global.document = dom.window.document;
  global.Image = Image;
  global.HTMLCanvasElement = Canvas;
  global.ImageData = ImageData;
```

```
  global.HTMLImageElement = Image;
}
```

ejemplo30.js

Nota: Al ejecutar este programa mediante **node ejemplo30.js** se obtiene el “error code”:

```
code: 'ERR_DLOPEN_FAILED'
```

Tal vez si usted ejecuta el ejemplo obtenga ese u otro error de compilación. Es la parte que no me gusta de **node** y **npm** en cuanto a la gestión de versiones de librerías en proyectos que se dejan por un tiempo y se retoman ejecutando un npm install: algunas versiones de librería se rompen y no funcionan. Lo más fácil sería retroceder en la versión de Node.js pero no lo voy a hacer.

El mismo código del ejemplo30.js funcionaba de maravilla con mi anterior versión de Node.js pero lo he reisntalado y con la nueva versión ya no funciona.

Le deseo a Ud. mejor suerte:

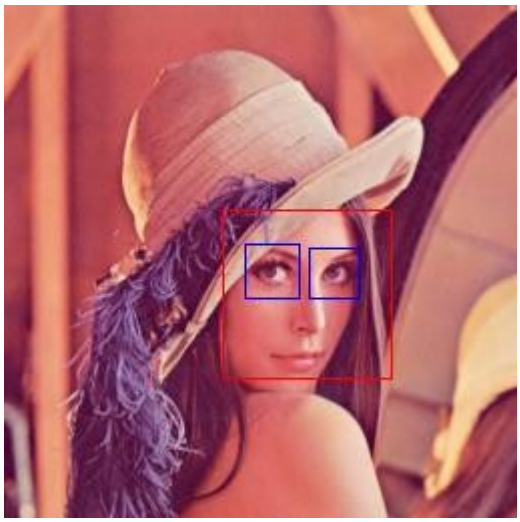
Ejecute el ejemplo

- Guarde el archivo como ejemplo30.js.
- Asegúrese de que los archivos aarcascade_frontalface_default.xml y haarcascade_eye.xml estén presentes en el directorio del proyecto. Se pueden obtener de las fuentes de OpenCV.
- Asegúrese de que exista un archivo de imagen de muestra lena.jpg en el directorio del proyecto. Debe mostrar las caras de las personas para que este ejemplo tenga sentido. Se sabe que la siguiente imagen lena.jpg funciona.

Resultado que obtuve antes de actualizar Node.js

Abajo mostramos la imagen original y la salida del programa:

lena.jpg



output3.jpg

2.- Características GUI (Interfaz Gráfico de Usuario)

Aquí aprenderá a leer y mostrar imágenes y videos, y a crear una barra de seguimiento.

2.1.- Empezando con imágenes

2.1.1.- Objetivos

Aprende a leer una imagen y a mostrarla en una web.

2.1.2.- Leer una imagen

Ya hizimos algo de esto en al anterior apartado. Profundizaremos aquí un poco más.

OpenCV.js guarda imágenes como de tipo **cv.Mat**. Usamos el elemento canvas de HTML para transferir cv.Mat a la web o al revés. La interfaz **ImageData** puede representar o establecer los píxeles de un área de un elemento de canvas.

En primer lugar creamos un objeto ImageData desde el canvas:

```
let canvas = document.getElementById(canvasInputId);
let ctx = canvas.getContext('2d');
let imgData = ctx.getImageData(0, 0, canvas.width, canvas.height);
```

Luego, utilizamos **cv.matFromImageData** para construir un cv.Mat:

```
let src = cv.matFromImageData(imgData);
```

Hasta aquí hemos construido la matriz **src** compatible con cv.Mat utilizando directamente el ImageData del canvaś. Luego veremos com esto puede realizarse mucho más fácil directamente desde OpenCV.js y el **id** del canvas. Lo mismo ocurrira en el proceso de mostrar la imagen que viene más abajo.

Nota

Debido a que el canvas solo admite imágenes RGBA de 8 bits con almacenamiento continuo, el tipo cv.Mat es cv.CV_8UC4. Es diferente del OpenCV nativo porque las imágenes devueltas y mostradas por **imread** e **imshow** nativos, tienen los canales almacenados en orden BGR.

2.1.3.- Mostrar una imagen

Primero, convertimos el tipo de src a cv.CV_8UC4 (8 bits, Unsigned, 4 Canales):

```
let dst = new cv.Mat();
// scale y shift son usados para mapear los datos a [0, 255].
src.convertTo(dst, cv.CV_8U, scale, shift);
// los *** significan GRAY, RGB, o RGBA, según src.channels() sea 1, 3 o 4.
cv.cvtColor(src, dst, cv.COLOR_***2RGBA);
```

Finalmente, obtenemos la matriz dst que será RGBA (4 canales)

A continuación creamos un nuevo ImageData a partir de dst:

```
let imgData = new ImageData(new Uint8ClampedArray(
                                dst.data, dst.cols, dst.rows
));
```

Finalmente mostramos la imagen:

```
let canvas = document.getElementById(canvasOutputId);
let ctx = canvas.getContext('2d');
ctx.clearRect(0, 0, canvas.width, canvas.height);
canvas.width = imgData.width;
canvas.height = imgData.height;
ctx.putImageData(imgData, 0, 0);
```

2.1.4.- Con OpenCV.js

OpenCV.js implementa la lectura y visualización de imágenes utilizando el método anterior.

Para leer una imagen del **canvas** o de un elemento **img** utilizamos la función:

cv.imread (imageSource)

Parámetros:

imageSource elemento canvas o id, o elemento img o id.

Retorno:

matriz con los canales almacenados en orden RGBA.

Para mostrar una imagen en un canvas utilizaremos la función:

cv.imshow (canvasSource, mat)

Parámetros:

canvasSource elemento canvas o id

mat matriz que se quiere mostrar

La función puede escalar la matriz, dependiendo de su profundidad, depth:

- Si la matriz es de 8 bits sin signo, se muestra tal cual.
- Si la matriz es de 16 bits sin signo o entero de 32 bits, los píxeles se dividen por 256. Es decir, el rango de valores [0, 255*256] se asigna a [0, 255].
- Si la matriz es de punto flotante de 32 bits, los valores de píxeles se multiplican por 255. Es decir, el rango de valores [0,1] se asigna a [0,255].

El código anterior de lectura y visualización de imágenes podría simplificarse como se muestra a continuación:

```
// Leemos la imagen desde imageSource y la almacenamos en la matriz img
let img = cv.imread(imageSource);
```

```
// mostramos la matriz img en el canvas canvasOutput
cv.imshow(canvasOutput, img);
img.delete();
```

2.1.5.- Ejemplo: leer y mostrar imágenes

En este apartado ampliaremos lo estudiado en el apartado anterior analizando un ejemplo concreto y también veremos algunas cosas nuevas que pueden ayudarnos a construir una interface de usuario simple para ejecutar los programas y visualizar los resultados en este ejemplo y en la mayoría de los siguientes.

En el listado que mostramos a continuación utilizamos opencv.js para leer una imagen de un canvas de entrada, transformarla a escala de grises y visualizarla en otro canvas de salida.

Los comentarios al programa se encuentran más abajo.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Leer y mostrar imágenes</title>
<link href="assets/js_example_style.css"
      rel="stylesheet"
      type="text/css"
/>
</head>
<body>
<h2>Leer y mostrar imágenes</h2>
<div>
<p class="err" id="errorMessage"></p>
</div>
<div>
    <table cellpadding="0" cellspacing="0" width="0" border="0">
    <tr>
        <td>
            <canvas id="canvasInput"></canvas>
        </td>
        <td>
            <canvas id="canvasOutput"></canvas>
        </td>
    </tr>
    <tr>
        <td>
            <div class="caption">canvasInput</div>
        </td>
        <td>
            <div class="caption">canvasOutput</div>
        </td>
    </tr>
    </table>
</div>

<script src="assets/utils.js" type="text/javascript"></script>
<script type="text/javascript">
let utils = new Utils('errorMessage');

// HTML
```

```
utils.loadImageToCanvas('assets/lena.png', 'canvasInput');

// OpenCV
utils.loadOpenCv(() => {
    let src = cv.imread('canvasInput');
    let dst = new cv.Mat();
    // Pasamos la imagen de salida a grises para
    // distinguirla de la entrada
    cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);
    cv.imshow('canvasOutput', dst);
    src.delete();
    dst.delete();
});
</script>
</body>
</html>
```

2011_leer_y_mostrar_imagenes.html

Resultado

Este es el resultado (se ha tomado como imagen de entrada lena.png)

Apartir de ahora utilizaremos un conjunto de utilidades que se usan en el tutorial de opencv.js que son sumamente prácticas para construir los programas. Las utilidades serán ficheros que almacenamos en el directorio **assets**. Veamos las más importantes por ahora.

- **Utils.js**

Es una función construtora de javascript que al instanciarse proporciona muchas funciones de utilidad. No es necesario entender el funcionamiento de estas funciones pero puede resultar

didáctico hacerlo. En el código veremos utilizar esta función, por ejemplo, como mostramos a continuación:

Instanciamos la función:

```
let utils = new Utils('cadena')
```

Cargamos la imagen assets/lena.png en el canvasInput

```
utils.loadImageToCanvas('assets/lena.png', 'canvasInput');
```

Añadimos un manejador input de tipo file

```
utils.addFileInputHandler('fileInput', 'canvasInput');
```

No vamos a utilizar de momento este manejador de fileInput.

Utils.js noos permite agrupar en una línea de código lo que equivale a varias líneas de javascript y los programas quedan más limpios.

- **js_example_style.css**

Ya dijimos que es una hoja de estilos que utiliza opencv y controla el aspecto de las tablas, los canvas, los botones, etc. La lincaremos sin más.

- **La función:**

```
utils.loadOpenCv (() => {})
```

Ejecuta su callback cuando opencv esta preparada.

Nos queda pues por analizar el código de ese callback que se jecutará cuando Opencv esté lista. Es el siguiente:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Pasaremos la imagen de salida a grises para
// distinguirla de la entrada
cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);
cv.imshow('canvasOutput', dst);
src.delete();
dst.delete();
```

Es autoexplicativo porque todas las instruciones salvo la que cambia la imagen de color a grises las hemos utilizado ya. Es la siguiente:

Función para cambiar la matriz de entrada a escala de grises:

cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);

Parámetros:

src	matriz de entrada
dst	matriz de salida
cv.COLOR_RGBA2GRAY	constante de opencv para pasar de color a grises

2.2.- Empezando con videos

2.2.1.- Objetivos

Aprender a capturar video de una cámara y visualizarlo.

2.2.2.- Capturar video desde la cámara

A menudo, tenemos que capturar transmisiones en vivo con una cámara. En OpenCV.js, usamos WebRTC y elementos de canvas HTML para implementar esto.

Capturaramos un video con la cámara (incorporada o USB), lo convertimos en un video en escala de grises y lo visualizamos.

Para capturar un video, debemos agregar algunos elementos HTML a la página web:

- un <video> para mostrar el video de la cámara directamente,
- un <canvas> para transferir video al ImageData del canvas frame por frame
- otro <canvas> para mostrar el video que obtiene OpenCV.js

En el programa que mostramos a continueción, simplemente usamos

> WebRTC navigator.mediaDevices.getUserMedia

para obtener el stream de video y mostrarlo en una etiqueta video.

El listado del programa se muestra a continuación:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Capturar video desde la cámara</title>
</head>
<body>
    <h2>Capturar video desde la cámara</h2>

    <video id="videoInput"></video>
    <script>
       let video = document.getElementById("videoInput");
        navigator.mediaDevices.getUserMedia({ video: true, audio: false })
            .then(function(stream) {
                video.srcObject = stream;
                video.play();
            })
            .catch(function(err) {
                console.log("An error occurred! " + err);
            });
    </script>

</body
</html>
```

2021_capturar_video_camara.html

2.2.3.- Reproducir video

El listado del programa se encuentra al final del apartado.

En este ejemplo, el navegador obtendrá el stream de la cámara y lo pasará a la propiedad **srcObject** de un elemento de **video** de entrada. Luego, utilizamos ese stream en la función **VideoCapure** para leerlo **frame por frame** y poder procesarlo. Para reproducir el video en un canvas, utilizamos **cv.imshow()** que debe ejecutarse cada un tcierto iempo **delay** en milisegundos. Utilizamos para ello el método **setTimeout()**. Si el video es de 30 fps, el delay en milisegundos debe ser:

```
let delay = 1000/FPS - (Date.now() - begin);
```

Y luego ejecutamos la función **processVideo** de la que hablaremos luego cada delay segundos así:

```
setTimeout(processVideo, delay);
```

2.2.3.1.- Comentarios sobre el listado del programa

Todos los programas que se listan para poder comentarlos se encuentran en la carpeta de **ejemplos_tutorial** que se descarga conjuntamente con este documento.

Concretamente, el programa que nos ocupa está en:

2022_reproducir_video.html

Listado más abajo.

Para entender el funcionamiento de este programa hay que analizar algunas funciones de la biblioteca **utils.js** que se utilizan como funciones auxiliares. El listado de utils.js se encuentra en el Anexo I.

Cuando se ejecuta el fichero **html**, queda a la espera de que ocurra un evento clic sobre el botón **startAndStop.** Si la cámara no está en streaming (emitiendo señal) se ejecuta la función:

```
utils.startCamera('qvga', onVideoStarted, 'videoInput');
```

Los argumentos que le pasamos son los siguientes:

- **‘qvga’**

Resolución que vamos a utilizar definida en la misma función **assets/utils/startCamera.**

- **onVideoStarted**

callback que será llamado por la función **navigator.mediaDevices.getUserMedia** dentro de la función startCamera cuando ya se haya obtenido el stream en el callback de la promesa:

```
callbacself.onCameraStartedCallback = callback;
```

- **videoInput**

El **id** del tag **<video>** del documento HTML donde ser reproducirá el video de entrada. Si no existe, la función crea un elemento **video** y le asigna a su **srcObject** el valor del **stream** obtenido antes.

Con esto estamos leyendo la señal de la camara en el elemento **video**.

Antes hemos visto que el segundo argumento que se le pasa a la función startCamera era el callbak **onVideoStarted**. Ahora es cuando se ejecuta.

Las primeras líneas de la función onVideoStarted deberían ser fáciles de comprender, no obstante haremos algunos comentarios:

```
let videoInput = document.getElementById('videoInput');
```

En la línea anterior creamos el elemento de video **videoInput** correspondiente al id='**videoInput**'. Ese mismo elemento es el que mandamos luego a **statrCamera** para renderizar el stream de la cámara.

La variable que captura un frame de video:

```
let cap = new cv.VideoCapture(video);
```

La utilizaremos en la función **processVideo** de la siguiente forma:

```
cap.read(src);
```

Con lo que la matriz de entrada **src** recibira el stream frame por frame para que se pueda procesar.

Esta función **processVideo** que realiza todo el proceso de vídeo se arranca desde onVideoStarted que es el callback que pasamos a startCamera.

La función processVideo se llama cíclicamente mediante la instrucción:

```
 setTimeout(processVideo, delay);
```

El tiempo de ciclo delay se calcula en la línea ubicada arriba de la línea anterior en el listado.

Las tres líneas siguientes muestran la captura del frame en src, su conversión a escala de grises y su visualización en el canvas canvasOutput:

```
cap.read(src);
cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);
cv.imshow('canvasOutput', dst);
```

2.2.3.2.- Listado de la función 2022_reproducir_video.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Reproducir vídeo de la cámara con OpneCV.js</title>
<link href="assets/js_example_style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h2>Reproducir vídeo de la cámara con OpneCV.js</h2>
<p>
    Pulse el botón <b>Start/Stop</b> para arrancar o parar la captura de
video por la cámara<br>
    El <b>videoInput</b> es un elemento &lt;video&gt; utilizazado como
entrada en OpenCV.js<br>
```

```
    El <b>canvasOutput</b> es un elemento &lt;canvas&gt; utilizado por
OpenCv.js como salida
</p>
<div>
<div class="control"><button id="startAndStop" disabled>Start</button></div>
</div>
<p class="err" id="errorMessage"></p>
<div>
    <table cellpadding="0" cellspacing="0" width="0" border="0">
    <tr>
        <td>
            <video id="videoInput" width=320 height=240></video>
        </td>
        <td>
            <canvas id="canvasOutput" width=320 height=240></canvas>
        </td>
    </tr>
    <tr>
        <td>
            <div class="caption">videoInput</div>
        </td>
        <td>
            <div class="caption">canvasOutput</div>
        </td>
    </tr>
    </table>
</div>
<script src="https://webrtc.github.io/adapter/adapter-5.0.4.js"
type="text/javascript"></script>
<script src="assets/utils.js" type="text/javascript"></script>
<script id="codeSnippet" type="text/code-snippet">

</script>
<script type="text/javascript">
let utils = new Utils('errorMessage');

let streaming = false;
let videoInput = document.getElementById('videoInput');
let startAndStop = document.getElementById('startAndStop');
let canvasOutput = document.getElementById('canvasOutput');
let canvasContext = canvasOutput.getContext('2d');

startAndStop.addEventListener('click', () => {
    if (!streaming) {
        utils.clearError();
        utils.startCamera('qvga', onVideoStarted, 'videoInput');
    } else {
        utils.stopCamera();
        onVideoStopped();
    }
});

function onVideoStarted() {
    streaming = true;
    startAndStop.innerText = 'Stop';
    videoInput.width = videoInput.videoWidth;
    videoInput.height = videoInput.videoHeight;

    let video = document.getElementById('videoInput');
```

```
    let src = new cv.Mat(video.height, video.width, cv.CV_8UC4);
    let dst = new cv.Mat(video.height, video.width, cv.CV_8UC1);
    let cap = new cv.VideoCapture(video);

    const FPS = 30;
    function processVideo() {
        try {
            if (!streaming) {
                // clean and stop.
                src.delete();
                dst.delete();
                return;
            }
            let begin = Date.now();
            // start processing.
            cap.read(src);
            cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);
            cv.imshow('canvasOutput', dst);
            // schedule the next one.
            let delay = 1000/FPS - (Date.now() - begin);
            setTimeout(processVideo, delay);
        } catch (err) {
            utils.printError(err);
        }
    };

    setTimeout(processVideo, 0);
}

function onVideoStopped() {
    streaming = false;
    canvasContext.clearRect(0, 0, canvasOutput.width, canvasOutput.height);
    startAndStop.innerText = 'Start';
}

utils.loadOpenCv(() => {
    startAndStop.removeAttribute('disabled');

});
</script>
</body>
</html>
```

2021_capturar_video_camara.html

2.3.- Añadir una Trackbar a una aplicación

En este apartado vamos a utilizar un trackbar para generar mediante su control deslizante un número **alfa** entre 0 y 1. Obtendremos también su complementario **beta** = 1 – alfa.

Luego utilizaremos la función **addWeighted()** para mezclar dos imágenes ponderadas con los dos coeficientes alpha y beta anteriores. La función tiene los siguientes parámetros:

cv.addWeighted(src1, alpha, src2, beta, gamma, dst, dtype)

Los parámetros de la función significan lo siguiente.

src1	primera matriz de entrada.
alpha	peso en la mezcla de cada uno de los elementos de la primera matriz
src2	segunda matriz de entrada.
beta	peso en la mezcla de cada uno de los elementos de la segunda matriz
gamma	escalar agregado a cada suma
dst	matriz de salida que tiene el mismo tamaño y número de canales que las matrices de entrada.
dtype	profundidad opcional de la matriz de salida; cuando ambas matrices de entrada tienen la misma profundidad, dtype se puede establecer en -1, que será equivalente a src1.depth().

Para obtener la matriz de salida **dst** a partir de las dos matrices de entrada **src1** y **src2** se aplica la siguiente fórmula:

```
dst(I) = saturate( src1(I)*alpha + src2(I)*beta + gamma )
```

El código del programa que listaremos a continuación es bastante intuitivo. Señalaremos simplemente la línea en donde se aplica la función anterior para que se identifiquen los argumentos pasados a la función:

```
cv.addWeighted( src1, alpha, src2, beta, 0.0, dst, -1);
```

2.3.1.- Listado de la aplicación con trackbar

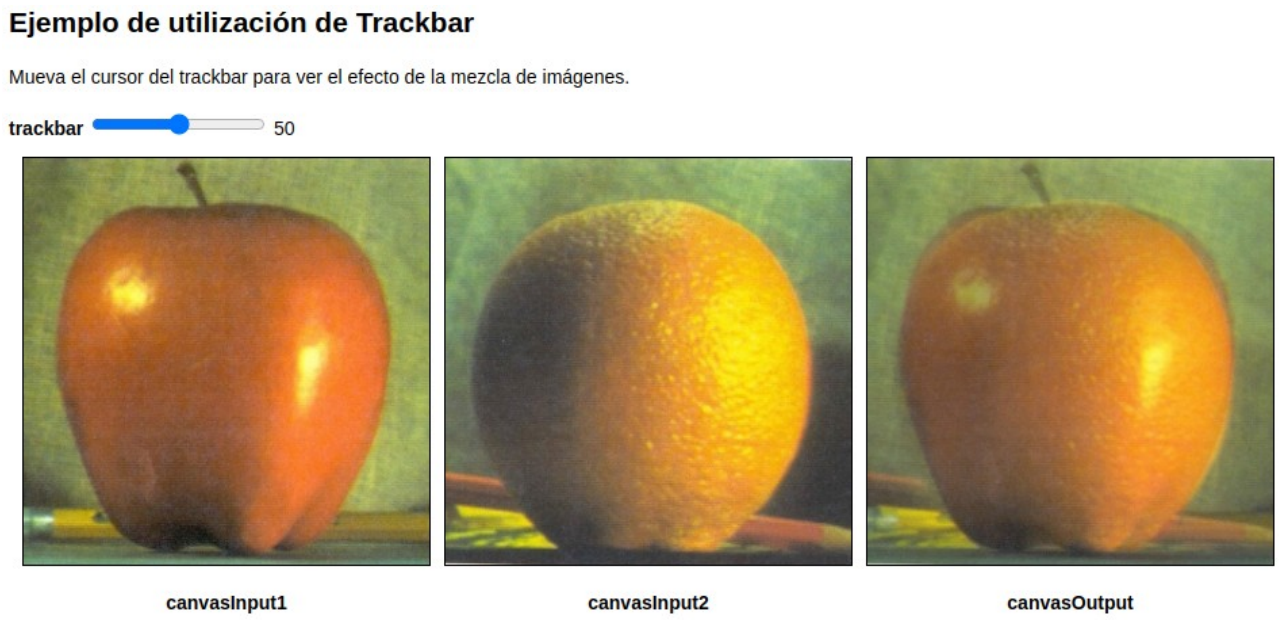
```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Ejemplo de utilización de Trackbar</title>
<link href="assets/js_example_style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h2>Ejemplo de utilización de Trackbar</h2>
<p>
    Mueva el cursor del trackbar para ver el efecto de la mezcla de imágenes.
</p>
<div>
<!-- <textarea class="code" rows="12" cols="80" id="codeEditor"
spellcheck="false"></textarea> -->
<p class="err" id="errorMessage"></p>
</div>
<div>
    <b>trackbar</b>
    <input type="range" id="trackbar" disabled value="50" min="0" max="100"
step="1">
    <label id="weightValue" ></label>
    <div>
        <table cellpadding="0" cellspacing="0" width="0" border="0">
        <tr>
```

```
            <td>
                <canvas id="canvasInput1" class="small"></canvas>
            </td>
            <td>
                <canvas id="canvasInput2" class="small"></canvas>
            </td>
            <td>
                <canvas id="canvasOutput" class="small"></canvas>
            </td>
        </tr>
        <tr>
            <td>
                <div class="caption">canvasInput1</div>
            </td>
            <td>
                <div class="caption">canvasInput2</div>
            </td>
            <td>
                <div class="caption">canvasOutput</div>
            </td>
        </tr>
        </table>
    </div>
</div>
<script src="assets/utils.js" type="text/javascript"></script>
<script type="text/javascript">
let utils = new Utils('errorMessage');

utils.loadImageToCanvas('assets/apple.jpg', 'canvasInput1');
utils.loadImageToCanvas('assets/orange.jpg', 'canvasInput2');

let trackbar = document.getElementById('trackbar');
trackbar.addEventListener('input', () => {

    // código que se ejecuta al mover el trackbar

    let alpha = trackbar.value/trackbar.max;
    let beta = ( 1.0 - alpha );
    let src1 = cv.imread('canvasInput1');
    let src2 = cv.imread('canvasInput2');
    let dst = new cv.Mat();
    cv.addWeighted( src1, alpha, src2, beta, 0.0, dst, -1);
    cv.imshow('canvasOutput', dst);
    dst.delete();
    src1.delete();
    src2.delete();
});

let weightValue = document.getElementById('weightValue');
weightValue.innerText = trackbar.value;
trackbar.addEventListener('input', () => {
    weightValue.innerText = trackbar.value;
});

utils.loadOpenCv(() => {
    trackbar.removeAttribute('disabled');
 });
</script>
</body>
```

```
</html>
```

2031_utilizar_trackbar.html

Resultado



2.4.- Plantilla HTML para OpenCV.js

En la mayoría de ejemplos que vienen a continuación utilizaremos un código HTML que es casi siempre el mismo en todos ellos. Generalmente incluirá una etiquea <img> para visualizar la imagen de entrada (que llamamos src) y una etiqueta <canvas> para visualizar la imagen de salida después del pocesamiento (dst).

Se utiliza el directorio assets para almacenar todos los ficheros auxiliares que requiere el programa, entre ellos la utilidad **utils.js**. Las funciones que utilizaremos con más frecuencia de utils.js son:

```
utils.loadImageToCanvas('path_imagen', 'canvas_id');
utils.addFileInputHandler('input_id', 'canvas_id');
utils.loadOpenCv(() => {
   // Código OpenCV.js
   // ...
});
// ...
```

La función **loadOpenCV()** es un callback que se llama cuando la librería opencv.js está cargada y preparada. Por tanto alojamos allí el código OpenCV.js.

En ocasiones se utiliza un botón **startAndStop** de forma que al cargar el dcumento el programa queda a la espera de que se pulse este boton en cuyo caso la plantilla sería algo diferente.

Listamos a continuancion la plantilla HTML más común que utilizaremos.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Título ejemplo</title>
<link href="assets/js_example_style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h2>Título ejemplo</h2>
<div>
<p class="err" id="errorMessage"></p>
</div>
<div>
    <table cellpadding="0" cellspacing="0" width="0" border="0">
    <tr>
        <td>
            <canvas id="canvasInput"></canvas>
        </td>
        <td>
            <canvas id="canvasOutput"></canvas>
        </td>
    </tr>
    <tr>
        <td>
            <div class="caption">canvasInput</div>
        </td>
        <td>
            <div class="caption">canvasOutput</div>
        </td>
    </tr>
    </table>
</div>
<script src="assets/utils.js" type="text/javascript"></script>
<script type="text/javascript">
let utils = new Utils('errorMessage');

// HTML
utils.loadImageToCanvas('assets/lena.png', 'canvasInput');

// OpenCV
// Ejemplo convertir color
utils.loadOpenCv(() => {
    // Inicio código OpenCV
    // Esteé codigo se reemplazará por el correspodiente de cada ejemplo
    let src = cv.imread('canvasInput');
    let dst = new cv.Mat();
    cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY, 0);
    cv.imshow('canvasOutput', dst);
    src.delete();
    dst.delete();
    // Fin códgo OpenCV.js
});
</script>
</body>
</html>
```

2041_plantilla-html.html

Para finalizar el tema de plantillas listamos a continuación otra plantilla que contiene en este caso el botón **startAndStop** y un **input** de tipo **file** para poder cambiar la imagen de entrada. Utilizaremos con más frecuecia la plantilla anterior porque no hay que esperar ni pulsar nada para ejecutar el programa y también porque dado que tenemos el código fuente, la imagen de entrada la podemos cambiar fácilmente desde allí.

En ambos casos, el copdigo OpenCV es solo de muestra para que funcione la plantilla y es el que hay que cambiar en cada caso por el de nuestro ejemplo concreto.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Título ejemplo</title>
<link href="assets/js_example_style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h2>Título ejemplo</h2>
<div>
<div class="control"><button id="tryIt" disabled>Try it</button></div>
<p class="err" id="errorMessage"></p>
</div>
<div>
    <table cellpadding="0" cellspacing="0" width="0" border="0">
    <tr>
        <td>
            <canvas id="canvasInput"></canvas>
        </td>
        <td>
            <canvas id="canvasOutput"></canvas>
        </td>
    </tr>
    <tr>
        <td>
            <div class="caption">canvasInput <input type="file"
id="fileInput" name="file" accept="image/*" /></div>
        </td>
        <td>
            <div class="caption">canvasOutput</div>
        </td>
    </tr>
    </table>
</div>
<script src="assets/utils.js" type="text/javascript"></script>
<script type="text/javascript">
let utils = new Utils('errorMessage');


utils.loadImageToCanvas('assets/lena.jpg', 'canvasInput');
utils.addFileInputHandler('fileInput', 'canvasInput');

let tryIt = document.getElementById('tryIt');
tryIt.addEventListener('click', () => {
    // Código OpenCV.js
    let src = cv.imread('canvasInput');
    let dst = new cv.Mat();
    cv.threshold(src, dst, 177, 200, cv.THRESH_BINARY);
    cv.imshow('canvasOutput', dst);
```

```
    src.delete();
    dst.delete();
    // Fin de código OpenCv.js
});

utils.loadOpenCv(() => {
    tryIt.removeAttribute('disabled');

});
</script>
</body>
</html>
```

2042_plantilla_html_starAndStop.html

3.- Operaciones Centrales

En esta sección aprenderemos algunas operaciones básicas sobre imágenes, algunas herramientas matemáticas y algunas estructuras de datos, etc.

3.1.- Operaciones básicas con imágenes

Aprenderemos a leer y editar valores de píxeles, trabajar con ROI (Regiones de Intersés) imágenes y otras operaciones básicas.

3.1.1.- Tipos de matrices en OpenCV.js

Antes de abordar las operaciones que podemos realizar con matrices de OpenCV.js es conveniente conocer la nomenclatura que utiliza OpenCV.js sobre matrices.

OpenCv.js puede trabajar con dos tipos de matrices o arrays:

- **Matrices formadas a partir de un array**

Por ejemplo, la instrucción:

```
let mat = cv.matFromArray(2, 2, cv.CV_8UC1, [1, 2, 3, 4]);
```

Crea una matriz mat de 2x2 elementos, cada uno de tipo cv.CV_8UC1, y con los valores 1 , 2, 3, 4.

Un poco más adelante explicaremos el significado la constante cv.CV_8UC1 y otras más que incluye OpenCV.js para definir el tipo de las matrices. Podemos adelantar que en este caso se trata de una matriz con un solo canal y datos de 8 bits sin signo.

- **Matrices formadas a partir de una imagen**

Por ejemplo la siguiente instrucción generaria una matriz **src** a partir de la imagen lena.png suponiendo que esta imagen se encuentra cargada en un canvas con el id “canvasInput”:

```
let src = cv.imread('canvasInput');
```

Las matrices que representan una imagen se clasifican en función del tipo de datos de sus elementos y el número de canales de la matriz, que puede ser: 1 para imágenes en grises, 3 para imágenes en color y 4 para imagenes en color que incluyan una canal alfa de opacidad. Así el ejemplo anterior sería:

cv.CV_8UC1 → matriz de 1 canal y elementos unsigned de 8 bits.

Otro ejemplo podría ser:

cv.CV_32FC3 → matriz de 3 canales y elementos float de 32 bits.

OpenCV asigna un tipo numerico a cada matriz según el tipo de datos y el número de canales. La clasificación es la siguiente:

	C1	C2	C3	C4
CV_8U	0	8	16	24
CV_8S	1	9	17	25
CV_16U	2	10	18	26
CV_16S	3	11	19	27
CV_32S	4	12	20	28
CV_32F	5	13	21	29
CV_64F	6	14	22	30

Tipos de matrices en OpenCV,js

Podemos obtener el tipo de la matriz src (número entero) con la instrucción:

```
console.log('type: ' + src.type())
```

El significado de las letras y números es el siguiente:

U	unsigned
S	signed
F	float
C1	1 canal
C2	2 canales
C3	3 canales
C4	4 canales

3.1.2.- Analizando las operaciones básicas con matrices

A continuación veremos como:

- Acceder a la propiedades de una imagen
- Crear una matriz
- Copiar una matriz
- Convertir el tipo de una matriz
- Utilizar MatVector
- Acceder y modificar los píxeles de una matriz
- Utilizar regiones de interés (ROI)
- Separar los canales de una matriz

Veamos cómo realizar estas operaciones.

Las que se muestran a continuación están extraidas del programa:

Que listaremos al final de este apartado.

3.1.3.- Averiguar el tipo de una matriz

En las siguientes líneas creamos la matriz src a partir de la imagen almacenada en 'canvasId' y a continuación mostramos por consola el tipo de la matriz:

```
let src = cv.imread('canvasInput');
console.log('tipo: ' + src.type())
```

3.1.4.- Leer las propiedades de la imagen (matriz)

```
console.log('width: ' + src.cols)
console.log('height: ' + src.rows)
console.log('size: ' + src.size().width + '*' + src.size().height)
console.log('depth: ' + src.depth())
console.log('channels ' + src.channels())
console.log('type: ' + src.type())
```

3.1.5.- Constructores

3.1.5.1.- Hay cuatro construcores básicos:

Construimos algunos valores auxiliares para que tengan coherencia las instruciones que mostramos más abajo.

```
// Valores auxiliares
let rows = 3
let cols = 5
let size = new cv.Size(cols, rows);
let type = cv.CV_32F
```

1. constructor por defecto

```
let mat1 = new cv.Mat();
```

2. matrices bidimensionales indicando size y type

```
let mat2 = new cv.Mat(size, cv.CV_8U);
```

3. matrices bidimensionales indicando rows, cols, y type

```
let mat3 = new cv.Mat(cols, rows, cv.CV_8UC3);
```

4. matrices bidimensionales indicando rows, cols, y type con valor de inicialización

```
let mat4 = new cv.Mat(cols, rows, cv.CV_8U, new cv.Scalar());
```

Veremos más adelante lo que es **new cv.Scalar()**

3.1.5.2.- Hay tres constructores estáticos

1. Crear una matriz con todos los elementos de valor cero

```
let mat1 = cv.Mat.zeros(rows, cols, type);
```

2. Crear una matriz con todos los elementos de valor uno

```
let mat2 = cv.Mat.ones(rows, cols, type);
```

3. Crear una matriz Identidad (diagonal principal unos y resto ceros)

```
let mat3 = cv.Mat.eye(rows, rows, type);
```

3.1.6.- Copiar una matriz

Hay dos formas de copiar una matriz:

1. Clonando la matriz

```
let src = cv.imread('canvasInput');
let dst = src.clone()
```

2. Con CopyTo (solo se copian las entradas indicadas en la máscara)

```
// Matriz fuente
let srcA = cv.Mat.zeros(8, 8, cv.CV_8UC3);
// Máscara
let mask = cv.Mat.ones(8, 8, cv.CV_8UC3); // todo unos
 // Matriz destino para la copia
let dstA = new cv.Mat()
// Copiamos srcA en destA teniendo en cuenta la máscara
// Copia todos los ceros de srcA en destA
srcA.copyTo(dstA, mask);
```

3.1.7.- Convertir el tipo de una matriz

Utilizamos la función:

`convertTo(dst, rtype, alpha = 1, beta = 0)`

Parámetros:

dst	matriz de salida; si no tiene un tamaño o tipo adecuado antes de la operación, se reasigna.
rtype	el tipo de matriz de salida deseado; si rtype es negativo, la matriz de salida tendrá el mismo tipo que la entrada
alpha	factor de escala opcional para todos los elementos. Se multiplican todos los valores por alpha
beta	delta opcional agregado a los valores escalados. Se suma beta a todos los valores.

Ejemplo de código:

```
let mat = cv.imread('canvasInput');
console.log('type mat: ' + mat.type())      // 24
let dst = new cv.Mat()
mat.convertTo(dst, cv.CV_64FC4)
console.log('type dst: ' + dst.type())    // 30
```

3.1.8.- Como utilizar MatVector

El código que sigue a continuación es autoexplicativo:

```
let mat = new cv.Mat();
// Initialise a MatVector
let matVec = new cv.MatVector();
// Push a Mat back into MatVector
matVec.push_back(mat);
// Get a Mat fom MatVector
let cnt = matVec.get(0);
mat.delete(); matVec.delete(); cnt.delete();

// No olvide eliminar mat, matVec y cnt cuando ya no quiera usarlos.
```

3.1.9.- Acceder y modificar los pixels de una matriz

En primer lugar, debemos conocer la relación de equivalencias entre tipos que se muestra a continuación:

Relación de Tipos

Data Properties	C++	TyJavaScript Typed Array	Mat Type
data	uchar	Uint8Array	CV_8U
data8S	char	Int8Array	CV_8S
data16U	ushort	Uint16Array	CV_16U
data16S	short	Int16Array	CV_16S
data32S	int	Int32Array	CV_32S
data32F	float	Float32Array	CV_32F
data64F	double	Float64Array	CV_64F

Cada columna representa una nomenclatura para acceder a los datos de una matriz cv.Mat.

Veamos algunos ejemplos

3.1.9.1.- Utilizando la propiedad data

Accedemos a los valores R, G, B, A del pixel de la tercera fila y cuarta columna. En primer lugar escribimos los valores 100 150 200 y 255 y a continuación leemos los canales R, G, B, A del mismo pixel comprobando que coincien con los valores que escribimos antes escribimos.

```
// La manipulación de datos como haremos a continuación solo es válida
// para una Mat continua.
// Debemos usar isContinuous() para verificarlo primero.
// Accedemos a los valores R, G, B, A del pixel de la tercera fila y cuarta
// columna
let row = 3, col = 4;
let src = cv.imread("canvasInput");


if (src.isContinuous()) {
  // Escribimos sobre la matriz
  src.data[row * src.cols * src.channels() + col * src.channels()] = 100
  src.data[row * src.cols * src.channels() + col * src.channels() + 1] = 150
  src.data[row * src.cols * src.channels() + col * src.channels() + 2] = 200
  src.data[row * src.cols * src.channels() + col * src.channels() + 3] == 255

  // Leemos de la matriz
  R = src.data[row * src.cols * src.channels() + col * src.channels()];
  G = src.data[row * src.cols * src.channels() + col * src.channels() + 1];
  B = src.data[row * src.cols * src.channels() + col * src.channels() + 2];
  A = src.data[row * src.cols * src.channels() + col * src.channels() + 3];
  // Los valores que hemos escrito
  console.log(R, G, B, A)     // 100 150 200 255
}
```

La manipulación con data solo es válida para una matriz continua.

Debemos usar primero **isContinuous()** para verificar que la matriz se almacena de una forma contínua.

3.1.9.2.- Utilizando la propiedad at

La manipulación con **at** es para un solo canal y los valores no se puede modificar.

Según el tipo de matriz hay que utilizar el siguiente manipulador at:

Manipuladores at

Mat Type	At Manipulation
CV_8U	ucharAt
CV_8S	charAt
CV_16U	ushortAt
CV_16S	shortAt
CV_32S	intAt

CV_32F	floatAt
CV_64F	doubleAt

Veamos el código de ejemplo:

```
let row = 3, col = 4;
let src = cv.imread("canvasInput");

let R = src.ucharAt(row, col * src.channels());
let G = src.ucharAt(row, col * src.channels() + 1);
let B = src.ucharAt(row, col * src.channels() + 2);
let A = src.ucharAt(row, col * src.channels() + 3);
```

3.1.9.3.- Utilizando la propiedad ptr

Pueden realizarse tanto operaciones de lectura como de escritura sobre los pixeles de la imagen.

Según el tipo de matriz hay que utilizar el siguiente manipulador ptr:

Mat Type	Ptr Manipulation	JavaScript Typed Array
CV_8U	ucharPtr	Uint8Array
CV_8S	charPtr	Int8Array
CV_16U	ushortPtr	Uint16Array
CV_16S	shortPtr	Int16Array
CV_32S	intPtr	Int32Array
CV_32F	floatPtr	Float32Array
CV_64F	doublePtr	Float64Array

Funciona como sigue, suponiendo que trabajamos con matrices CV_8U.

Obtiene la **k-ésima** fila de la matriz:

```
mat.ucharPtr(k)
```

Obtiene el pixel de la i-ésima fila y la j-ésima columna de la matriz

```
mat.ucharPtr(i, j)
```

Veamos su utilización en el código:

```
let row = 3, col = 4;
let src = cv.imread("canvasInput");
let pixel = src.ucharPtr(row, col);
let R = pixel[0];
```

```
let G = pixel[1];
let B = pixel[2];
let A = pixel[3];
```

El siguiente listado corresponde a un programa que maneja todos los conceptos visto en este apartado y se ejecuta sin errores, lo que es consecuencia de que todas las definiciones se han aplicado correctamente.

El programa está ámpliamente comentado y se encuentra también entre los ficheros que acompañan este documento.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Image Read and Show Example</title>
<link href="assets/js_example_style.css" rel="stylesheet" type="text/css" />
</head>
<body>
<h2>Image Read and Show Example</h2>
<div>
<p class="err" id="errorMessage"></p>
</div>
<div>
    <table cellpadding="0" cellspacing="0" width="0" border="0">
    <tr>
        <td>
            <canvas id="canvasInput"></canvas>
        </td>
        <td>
            <canvas id="canvasOutput"></canvas>
        </td>
    </tr>
    <tr>
        <td>
            <div class="caption">canvasInput
                <input type="file"
                       id="fileInput"
                       name="file"
                       accept="image/*" />
            </div>
        </td>
        <td>
            <div class="caption">canvasOutput</div>
        </td>
    </tr>
    </table>
</div>

<script src="assets/utils.js" type="text/javascript"></script>
<script type="text/javascript">
let utils = new Utils('errorMessage');

// HTML
utils.loadImageToCanvas('assets/lena.png', 'canvasInput');
utils.addFileInputHandler('fileInput', 'canvasInput');

// OpenCV
```

```
utils.loadOpenCv(() => {

    // Averiguar el tipo de una matriz
    let src = cv.imread('canvasInput');
    console.log('tipo: ' + src.type())   //=> 24

    // Leer las propiedades de la imagen (matriz)
    console.log('width: ' + src.cols)
    console.log('height: ' + src.rows)
    console.log('size: ' + src.size().width + '*' + src.size().height)
    console.log('depth: ' + src.depth())
    console.log('channels ' + src.channels())
    console.log('type: ' + src.type())

    // Cómo construir la matriz
    // Valores auxiliares
    let rows = 3
    let cols = 5
    let size = new cv.Size(cols, rows);
    let type = cv.CV_32F

    // Hay cuatro construcores básicos
    // ================================
    // 1. constructor por defecto
    let mat1 = new cv.Mat();
    // 2. matrices bidimensionales indicando size y type
    let mat2 = new cv.Mat(size, cv.CV_8U);
    // 3. matrices bidimensionales indicando rows, cols, y type
    let mat3 = new cv.Mat(cols, rows, cv.CV_8UC3);
    // 4. matrices bidimensionales indicando rows, cols, y type con valor de
inicialización
    let mat4 = new cv.Mat(cols, rows, cv.CV_8U, new cv.Scalar());

    // Hay tres constructores estáticos
    // ================================
    {
        // 1. Crear una matriz con todos los elementos de valor cero
        let mat1 = cv.Mat.zeros(rows, cols, type);
        // 2. Crear una matriz con todos los elementos de valor uno
        let mat2 = cv.Mat.ones(rows, cols, type);
        // 3. Crear una matriz Identidad (diagonal principal unos y resto
ceros)
        let mat3 = cv.Mat.eye(rows, rows, type);
    }

    // Hay dos factorías de funciones
    // ==============================
    // 1. Utiliza un JS array para construir una matriz.
    // Por ejemplo:
    {
        // let mat = cv.matFromArray(rows, cols, type, array);
        let mat1 = cv.matFromArray(2, 2, cv.CV_8UC1, [1, 2, 3, 4]);

        // 2. Utiliza el imgData de un canvas para construir la matriz
        let ctx = canvasInput.getContext("2d");
        let imgData = ctx.getImageData(0, 0, 100, 100);
        let mat2 = cv.matFromImageData(imgData);
    }
```

```
    // Hay dos formas de copiar una matriz
    // ===================================
    {
        // 1. Clonando la matriz
        let src = cv.imread('canvasInput');
        let dst = src.clone()

        // 2. Con CopyTo (solo se copian las entradas indicadas en la
máscara)
        // Fuente
        let srcA = cv.Mat.zeros(8, 8, cv.CV_8UC3);
        // Máscara
        let mask = cv.Mat.ones(8, 8, cv.CV_8UC3); // todo unos
        // Destino
        let destA = new cv.Mat()
        // Copiamos srcA en destA teniendo en cuenta la máscara
        // Copia todos los ceros de srcA en destA
        srcA.copyTo(destA, mask);
    }

    // Convertir el tipo de una matriz
    // ===============================
    /*
    Utilizar la función:
       convertTo(dest, rtype, alpha = 1, beta = 0)

       Parámetros:

        dest    matriz de salida; si no tiene un tamaño o tipo adecuado
                antes de la operación, se reasigna.

        rtype   el tipo de matriz de salida deseado o, mejor dicho, la
                profundidad ya que el número de canales es el mismo que
                tiene la entrada; si rtype es negativo, la matriz de salida
                tendrá el mismo tipo que la entrada

        alpha     factor de escala opcional

        beta    delta opcional agregado a los valores escalados.
    */

    {
        let mat = cv.imread('canvasInput');
        console.log('type mat: ' + mat.type())      // 24
        let dest = new cv.Mat()
        mat.convertTo(dest, cv.CV_64FC4)
        console.log('type dest: ' + dest.type())    // 30
    }


    // Como utilizar MatVector
    // =======================
    {
        let mat = new cv.Mat();
        // Initialise a MatVector
        let matVec = new cv.MatVector();
        // Push a Mat back into MatVector
        matVec.push_back(mat);
```

```
        // Get a Mat fom MatVector
        let cnt = matVec.get(0);
        mat.delete(); matVec.delete(); cnt.delete();

        // No olvide eliminar mat, matVec y cnt
        // cuando ya no quiera usarlos.
    }

    // Acceder y modificar los pixels de una matriz
    // ============================================
    /*
    En primer lugar, debe conocer la siguiente relación de tipos:

        Data Properties C++      TyJavaScript Typed Array   Mat Type
        =============== ======== ========================   ========
        data              uchar     Uint8Array                 CV_8U
        data8S            char      Int8Array                  CV_8S
        data16U           ushort    Uint16Array                  CV_16U
        data16S           short     Int16Array                 CV_16S
        data32S           int     Int32Array                   CV_32S
        data32F           float   Float32Array            CV_32F
        data64F         double      Float64Array               CV_64F
    {
    */
        {
            // 1. utilizando la propiedad data
            // ===============================
            // La manipulación de datos como haremos a continuación solo es
válida
            // para una Mat continua.
            // Debemos usar isContinuous() para verificarlo primero.
            // Accedemos a los valores R, G, B, A del pixel de la tercera
fila y cuarta columna
            let row = 3, col = 4;
            let src = cv.imread("canvasInput");


            if (src.isContinuous()) {
                // Escribimos sobre la matriz
                src.data[row * src.cols * src.channels() + col *
src.channels()] = 100
                src.data[row * src.cols * src.channels() + col *
src.channels() + 1] = 150
                src.data[row * src.cols * src.channels() + col *
src.channels() + 2] = 200
                src.data[row * src.cols * src.channels() + col *
src.channels() + 3] == 255

                // Leemos de la matriz
                R = src.data[row * src.cols * src.channels() + col *
src.channels()];
                G = src.data[row * src.cols * src.channels() + col *
src.channels() + 1];
                B = src.data[row * src.cols * src.channels() + col *
src.channels() + 2];
                A = src.data[row * src.cols * src.channels() + col *
src.channels() + 3];
                // Los valores que hemos escrito
                console.log(R, G, B, A)     // 100 150 200 255
```

```
            }

        }

        {
            // 2. utilizando la propiedad at
            // ============================
            // La manipulación con at es para un solo canal y los valores no
se puede modificar.
            // Según el tipo de matriz hay que utilizar el siguiente
manipulador at:
            /*
                Mat Type     At Manipulation
                CV_8U       ucharAt
                CV_8S       charAt
                CV_16U      ushortAt
                CV_16S      shortAt
                CV_32S      intAt
                CV_32F      floatAt
                CV_64F      doubleAt
            */
            let row = 3, col = 4;
            let src = cv.imread("canvasInput");

            let R = src.ucharAt(row, col * src.channels());
            let G = src.ucharAt(row, col * src.channels() + 1);
            let B = src.ucharAt(row, col * src.channels() + 2);
            let A = src.ucharAt(row, col * src.channels() + 3);
        }

        {
            // 3. ptr
            // mat.ucharPtr(k) obtiene la k-ésima fila de la matriz
            // mat.ucharPtr(i, j) obtiene el pixel de la i-ésima fila y la j-
ésima
            // columna de la matriz
            /*
                Mat Type     Ptr Manipulation  JavaScript Typed Array
                CV_8U       ucharPtr            Uint8Array
                CV_8S       charPtr           Int8Array
                CV_16U      ushortPtr           Uint16Array
                CV_16S      shortPtr            Int16Array
                CV_32S      intPtr            Int32Array
                CV_32F      floatPtr            Float32Array
                CV_64F      doublePtr           Float64Array
            */
            let row = 3, col = 4;
            let src = cv.imread("canvasInput");
            let pixel = src.ucharPtr(row, col);
            let R = pixel[0];
            let G = pixel[1];
            let B = pixel[2];
            let A = pixel[3];
        }

    /*
        Al principio del script hemos utilizado la instrucción:
            utils.loadImageToCanvas('assets/lena.png', 'canvasInput');
        que carga la imagen lena.jpg en el canvas con ia="canvasInput"
```

```
        y por tanto se visualiza.

        Abajo construimos la natriz dest transformando a grises la matriz
        src correspondiente a la imagen de lena.png y la mostramos en el
        canvas con id="canvasOutput"

        De esta manera, cuando no hay errores en el documento debe mostrar
        las dos imágenes cuando éste se cargue, y no es necesrio revisar la
        consola del navegador puesto que no tendremos errores.
    */
    let dst = new cv.Mat();
    // Pasamos la imagen a grises
    cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY);
    cv.imshow('canvasOutput', dst);
    src.delete();
    dst.delete();
});
</script>
</body>
</html>
```

3011_operaciones_basicas.html

3.1.10.- Capturar una región de interes

A veces, hay que capturar una cierta región de una imagen. Por ejemplo, para la detección de ojos en imágenes, en la primera detección de rostros se realiza en toda la imagen y cuando se obtiene el rostro, seleccionamos esa región y buscamos los ojos dentro de ella en lugar de buscar en toda la imagen. Mejora la precisión (porque los ojos siempre están en las caras) y el rendimiento (porque buscamos en un área pequeña).

Utilizaremos la funció:

roi(rect)

Parámetros:

rect rectángulo de la región de interes

En el programa, **3012_region_interes_roi.html** podemos ver un ejemplo de aplicación.

En lo sucesivo ya no mostraremos los listados completos de los programas. Toda la parte del HTML la hemos visto ya muchas veces y la explicamos también al hablar de las plantillas HTML. Entonces listaremos únicamente la parte correspondiente a OpenCV y daremos por supuesto la entrada y salida de imágenes.

Listado

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Puede probar diferentes parámetros
// vertice (sup,izq), vertice (inf, der)
let rect = new cv.Rect(100, 100, 200, 200);
dst = src.roi(rect);
cv.imshow('canvasOutput', dst);
```

```
src.delete();
dst.delete();
```

3012_region_interes_roi.html

3.1.11.- Dividir y fusionar canales de imágenes

A veces necesitaremos trabajar por separado en los canales de imagen R, G, B. Luego, debemos dividir las imágenes RGB en planos individuales. En otro momento, es posible que debamos unir estos canales individuales para formar la imagen RGB.

El código para realizar la tarea anterior es el siguiente:

```
let src = cv.imread('canvasInput');
let rgbaPlanes = new cv.MatVector();
// Divide la matriz en planos y los almacena en rgbaPlanes
cv.split(src, rgbaPlanes);
// Obtenemos el canal R
let R = rgbaPlanes.get(0);
// Fusionamos todos los canales
cv.merge(rgbaPlanes, src);
// Visualizamos el canal rojo
cv.imshow('canvasOutput', R);
src.delete(); rgbaPlanes.delete(); R.delete();
```

3013_dividir_fusionar_imagenes.html

3.1.12.- Poner bordes a una imagen

La función que se utiliza es:

copyMakeBorder(src, dst, top, bottom, left, right, borderType, Scalar(255, 0, 0, 255))

Parámetros:

src	imagen de entrada
dst	imagen de salida con bordes
borderType	tipo de borde p.ej. BORDER_REPLICATE (hay más tipos abajo)
Scalar(255, 0, 0)	el color, en este caso rojo
top, bottom, left, right	ancho de los bordes en pixels

Tipos de bordes (borderType)

- cv.BORDER_CONSTANT – Añade un borde de color constante indicado por el último argumento de copyMakeBorder():
- cv.BORDER_REFLECT - El borde será un reflejo especular de los elementos del borde, como este : fedcba|abcdefgh|hgfedcb

- cv.BORDER_REFLECT_101 or cv.BORDER_DEFAULT - Igual que el anterior, pero con un ligero cambio, como este : gfedcb|abcdefgh|gfedcba
- cv.BORDER_REPLICATE - El último elemento se replica en todas partes, así: aaaaaa| abcdefgh|hhhhhhh
- cv.BORDER_WRAP - No se puede explicar, se verá así: cdefgh|abcdefgh|abcdefg

Veamos un programa de aplicación:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Puede probar varios parámetros
let s = new cv.Scalar(255, 0, 0, 255);
cv.copyMakeBorder(src, dst, 10, 10, 10, 10, cv.BORDER_CONSTANT, s);
cv.imshow('canvasOutput', dst);
src.delete();
dst.delete();
```

3014_poner_bordes_imagen.html

3.2.- Operaciones aritméticas básicas

Aprenderemos a leer y editar valores de píxeles, trabajar con ROI de imágenes y otras operaciones básicas.

3.2.1.- Suma, resta y operaciones bit a bit sobre imágenes

Veremos varias operaciones aritméticas en imágenes como sumas, restas, operaciones bit a bit, etc. Comprenderemos las funciones: **cv.add()**, **cv.subtract(),** etc.

3.2.2.- Adición de imágenes con cv.add()

Podemos sumar dos imágenes mediante la función cv.add() de OpenCV.

resultado = img1 + img2.

Las dos imágenes que se suman deben tener la misma profundidad (depth) y el mismo tipo (type)

El siguiente código muestra como sumar dos imágenes:

La suma se realizará en los puntos donde la matriz mask sea distinto de cero.

```
let src1 = cv.imread("canvasInput1");
let src2 = cv.imread("canvasInput2");
let dst = new cv.Mat();
let mask = new cv.Mat();
let dtype = -1;
cv.add(src1, src2, dst, mask, dtype);
cv.imshow('canvasOutput1', dst);
src1.delete(); src2.delete(); dst.delete(); mask.delete();
```

E la parámetro dtype es opcional e indica propundidad desada en la matriz de salida.

Si si las dos matrices que se suman tienen el mismo depth, dtype se puede establecer en -1, que será equivalente a src1.depth().

3.2.3.- Substracción de imágenes con cv.subtract()

Podemos restar dos imágenes mediante la función OpenCV, cv.subtract()

resultado = img1 - img2.

Ambas imágenes deben ser de la misma profundidad y tipo. Tenga en cuenta que cuando se utiliza con imágenes RGBA, también se resta el canal alfa.

El siguiente código muestra como restar dos imágenes:

(La suma se realizara en los puntos donde la matriz mask sea distinto de cero.)

```
let src1 = cv.imread("canvasInput1");
let src2 = cv.imread("canvasInput2");

// Pasamos las imágenes a grises para evitar canal alfa
cv.cvtColor(src1, src1, cv.COLOR_RGBA2GRAY);
cv.cvtColor(src2, src2, cv.COLOR_RGBA2GRAY);

let dst = new cv.Mat();
let mask = new cv.Mat();
let dtype = -1;
cv.subtract(src1, src2, dst, mask, dtype);
cv.imshow('canvasOutput2', dst);
src1.delete(); src2.delete(); dst.delete(); mask.delete();
```

Si utilizamos como máscar **new cv.Mat()** tanto la suma como la resta se realiza en todos los puntos.

3.2.4.- Operaciones bit a bit

Incluyen las operaciones bit a bit AND, OR, NOT y XOR. Serán muy útiles al extraer cualquier parte de la imagen, definir y trabajar con un ROI no rectangular, etc.

A continuación veremos un ejemplo de cómo cambiar una región particular de una imagen.

Queremos poner el logotipo de OpenCV en la esquina superior iaquierda de la imagen de lena. Por el camino hemos jugado también con otra imagen aplicando operaciones bit a bit. La mejor forma de aprender lo que hemos explicado anteriormente es entender este ejemplo, modificarlo y hacer que haga cosas distinatas.

El significado de las constantes que se utilizan al humbralizar una imagen está muy bien explicado allí.

Existen las siguientes constantes:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO

- cv.THRESH_OTSU
- cv.THRESH_TRIANGLE

El código para realizar estas operaciones anteriores de adición, substracción y bit a bit de imágenes puede verse en el siguiente programa internamente comentado:

3021_add_sub_bitwise_image.html

Por ejemplo:

(Se utilizan bloques para poder redeclarar variables con el mismo nombre)

```
// Substracción de imágenes con cv.subtract()
    {
        let src1 = cv.imread("canvasInput1");
        let src2 = cv.imread("canvasInput2");

        // Pasamos las imágenes a grises para evitar canal alfa
        cv.cvtColor(src1, src1, cv.COLOR_RGBA2GRAY);
        cv.cvtColor(src2, src2, cv.COLOR_RGBA2GRAY);

        let dst = new cv.Mat();
        let mask = new cv.Mat();
        let dtype = -1;
        cv.subtract(src1, src2, dst, mask, dtype);
        cv.imshow('canvasOutput2', dst);
        src1.delete(); src2.delete(); dst.delete(); mask.delete();
    }

// Operaciones bit a bit
    {
        let src = cv.imread('canvasInput1');
        let logo = cv.imread('logoCanvasInput');
        let dst = new cv.Mat();
        let roi = new cv.Mat();
        let mask = new cv.Mat();
        let maskInv = new cv.Mat();
        let imgBg = new cv.Mat();
        let imgFg = new cv.Mat();
        let sum = new cv.Mat();
        let rect = new cv.Rect(0, 0, logo.cols, logo.rows);

        // Queremos pner el logo en la esquina superior izquierda, luego
         // creamos un ROI
        roi = src.roi(rect);

        // Creamos una máscara del logo y su inversa
        cv.cvtColor(logo, mask, cv.COLOR_RGBA2GRAY, 0);
        cv.threshold(mask, mask, 100, 255, cv.THRESH_BINARY);
        cv.bitwise_not(mask, maskInv);

        // Ocultar el área del logotipo en ROI
        cv.bitwise_and(roi, roi, imgBg, maskInv);

        // Tomamos solo la región del logotipo de la imagen del logotipo
        cv.bitwise_and(logo, logo, imgFg, mask);
```

```
        // Put logo in ROI and modify the main image
        cv.add(imgBg, imgFg, sum);

        dst = src.clone();
        for (let i = 0; i < logo.rows; i++) {
            for (let j = 0; j < logo.cols; j++) {
                dst.ucharPtr(i, j)[0] = sum.ucharPtr(i, j)[0];
            }
        }
        cv.imshow('canvasOutput3', dst);


        src.delete(); dst.delete(); logo.delete(); roi.delete();
        mask.delete();maskInv.delete(); imgBg.delete(); imgFg.delete();
        sum.delete();
    }
```

3021_add_sub_bitwise_image.html

3.3.- Algunas estructuras de datos

3.3.1.- Objetivos

Aprenderá algunas estructuras de datos: **Point**, **Scalar**, **Size**, **Circle**, **Rect**, **RotatedRect**, etc.Scalar es un tipo de vector en Javascript. Point, Size, Circle, Rect y RotatedRect son tipos de objetos en JavaScript.

3.3.2.- Point

Hay 2 formas de construir un **Point** y son equivalentes:

```
// La primera forma
let point = new cv.Point(x, y);
// la segunda forma
let point = {x: x, y: y};
```

Parámetros:

x coordenada x del punto. (el origen es la esquina superior izquierda de la imagen)

y coordenada y del punto

3.3.3.- scalar

Hay 2 formas de construir un **Scalar** y son equivalentes:

```
// La primera forna
let scalar = new cv.Scalar(R, G, B, Alpha);
// La segunda forma
let scalar = [R, G, B, Alpha];
```

Parámetros:

R valor de píxel del canal rojo

G valor de píxel del canal verde

B valor de píxel del canal azul

alfa valor de píxel alfa del canal alfa

3.3.4.- Size

Hay 2 formas de construir un **Size** y son equivalentes:

```
// La primera forma
let size = new cv.Size(width, height);
// La segunda forna
let size = {width : width, height : height};
```

Parametros:

width el ancho del size

height el alto del size

3.3.5.- Circle

Hay 2 formas de construir un Circle y son equivalentes:

```
// La primera forma
let circle = new cv.Circle(center, radius);
// La segunda forma
let circle = {center : center, radius : radius};
```

Parámetros:

center el centro del círculo

radius el radio del círculo

3.3.6.- Rect

Hay 2 formas de construir un Rect y son equivalentes:

```
// La primera forma
let rect = new cv.Rect(x, y, width, height);
// La segunda forma
let rect = {x : x, y : y, width : width, height : height};
```

Parámetros:

x coordenada x del vértice esquina superior izquierda del rectángulo

y coordenada y del vértice esquina superior izquierda del rectángulo

width el ancho del rectángulo

height la altura del rectángulo

3.3.7.- RotatedRect

Hay 2 formas de construir un RotatedRect y son equivalentes:

```
// La primera forma
let rotatedRect = new cv.RotatedRect(center, size, angle);
// La segunda forma
let rotatedRect = {center : center, size : size, angle : angle};
```

Parámetros:

center el centro de masas del rectángulo

size ancho y alto del rectángulo

angle el ángulo de rotación en el sentido de las agujas del reloj. Cuando el ángulo es 0, 90, 180, 270, etc., el rectángulo se convierte en un rectángulo vertical

a) Veamos cómo podemos obtener los vértices de un **rotatedRect**

Utilizaremos la función:

cv.RotatedRect.points(rotatedRect)

Parámetros:

rotatedRect rotatedRect rectángulo

Procederíamos de la siguiente forma:

```
let vertices = cv.RotatedRect.points(rotatedRect);
let point1 = vertices[0];
let point2 = vertices[1];
let point3 = vertices[2];
let point4 = vertices[3];
```

b) Veamos como obtener el rectángulo delimitador de un RotatedRect.

Utilizaremos la función:

cv.RotatedRect.boundingRect(rotatedRect)

Parámetros:

rotatedRect rotatedRect rectángulo

Procedemos así:

```
let boundingRect = cv.RotatedRect.boundingRect(rotatedRect);
```

3.3.8.- Funciones de dibujo

No hemos encontrado de momento documentos explicacitovos sobre las funciones de dibujo en OpenCV.js.

No obstante si existen documentos para Python y C++ que documentan funciones que se llaman igual y tienen los mismos parámetros que en OpenCV.js. Daremos algunos nlaces.

a) desde python

Funciones: cv.line(), cv.circle() , cv.rectangle(), cv.ellipse(), cv.polilines(), cv.putText()

- https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html

b) Desde C++

- https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html

Documentamos algunas funciones:

3.3.8.1.- cv.line()

cv.line (img, pt1, pt2, color[, thickness[, lineType[, shift]]])

Parámetros:

img	image donde se dibuja
pt1	punto inicial
pt2	punto final
color	color
thickness	espesor
lineType	tipo de línea
shift	Número de bits fraccionarios en las coordenadas del punto

Ejemplo:

```
let img = cv.Mat.zeros(512, 512, cv.CV_8U);
let p1 = new cv.Point(100, 100);
let p2 = new cv.Point(200, 200);
let scalar = new cv.Scalar(150, 190, 280, 255);
cv.line(img,p1,p2,scalar,5)
cv.imshow('canvasOutput', img);
```

3.3.8.2.- cv.rectangle()

Tenemos dos posibilidades según demos como entrada dos puntos correspondientes a vertices opuestos o directamente un objeto de tipo Rect.

cv.rectangle(img, pt1, pt2, color[, thickness[, lineType[, shift]]])

cv.rectangle(img, rec, color[, thickness[, lineType[, shift]]])

3.3.8.3.- cv.circle()

La función es la siguiente:

cv.circle(img, center, radius, color[, thickness[, lineType[, shift]]])

3.3.8.4.- cv.ellipse()

Tenemos dos funciones.

La siguiente función permite dibujar una elipse completa o una parte de ella:

cv.ellipse(img, center, axes, angle, startAngle, endAngle, color
[, thickness[, lineType[, shift]]])

Podemos tambier dibujar una elipse inscrita en un RotatedRect:

cv.ellipse(img, box, color[, thickness[, lineType]]

Parámetro:

box el RotatedRect que circunscribe la alipse.

3.3.8.5.- cv.putText()

Utilizamos la siguiente función:

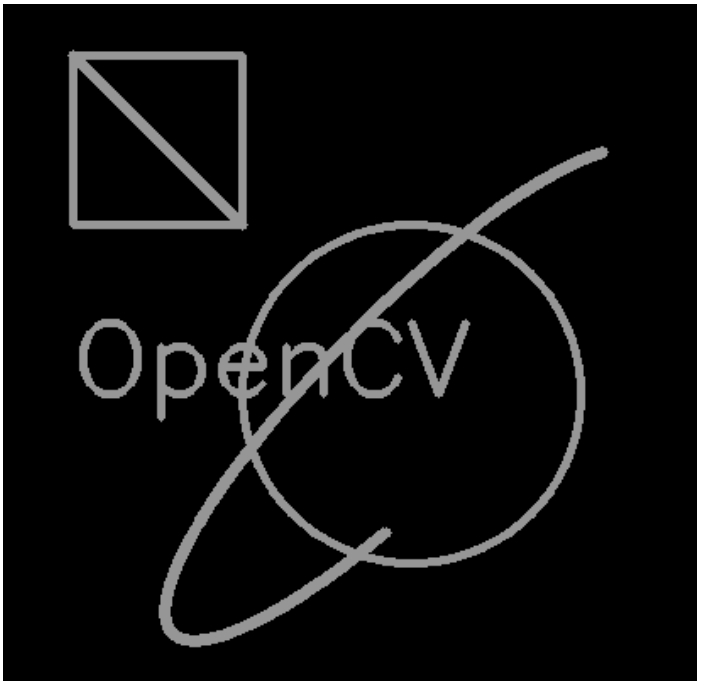
cv.putText (img, texto, origen, fontFace, fontScale, color, thickness = 1,
lineType = LINE_8, bottomLeftOrigin = false)

En el fichero

3023_some_data_structures.html

se utilizan la mayoria de funciones para crear estructuras y de dibujo citadas en este apatado.

Resultados



4.- Procesamiento de Imágenes

En esta sección aprenderá diferentes funciones de procesamiento de imágenes dentro de OpenCV.

4.1.- Cambio del espacio de color

Aprenderemos a cambiar imágenes entre diferentes espacios de color.

Objetivos

En este tutorial, aprenderemos cómo convertir imágenes de un espacio de color a otro, como de RGB a Gray, de RGB a HSV, etc. Aprenderemos las siguientes funciones:

cv.cvtColor(), cv.inRange()

4.1.1.- cvtColor

Hay más de 150 métodos de conversión de espacio de color disponibles en OpenCV. Pero nos fijaremos en el más utilizado: RGB a Gray y viceversa. Usamos la función:

cv.cvtColor (src, dst, code, dstCn = 0)

Parámetros:

src	imagen de entrada.
dst	imagen de salida del mismo tamaño y profundidad que src
code	el código de conversión del espacio de color (ver cv.ColorConversionCodes)
dstCn	número de canales en la imagen de destino; si el parámetro es 0, el número de canales se deriva automáticamente de src y code.

Para la conversión RGB a Gray hay que utilizar el code **cv.COLOR_RGBA2GRAY**.

Si miramos en:

```
      enum cv::ColorConversionCodes {
            cv.COLOR_RGBA2GRAY = 11
                        ...
     {
```

Vemos que la cosntante alfanumérica se corresponde con un número, en este caso el 11.

El siguiente código muestra la utilización de la función **cvtColor**.

```
let src = cv.imread('canvasInput');
    let dst = new cv.Mat();
    // Puede probar con diferentes parámetros
    cv.cvtColor(src, dst, cv.COLOR_RGBA2GRAY, 0);
    cv.imshow('canvasOutput', dst);
    src.delete();
    dst.delete();
```

4010_cambio_espacio_color.html

Resultados

canvasInput canvasOutput

4.1.2.- cv.inRange

Comprueba si los elementos de la matriz se encuentran entre los elementos de otras dos matrices. Si es así, el elemento o pixel se pinta en blanco, si no, en negro.

Utilizamos la función:

cv.inRange (src, lowerb, upperb, dst)

Parámetros:

src primera imagen de entrada.

lowerb Matriz inclusiva de límite inferior del mismo tamaño que src.

upperb Matriz inclusiva de límite superior tamaño que src

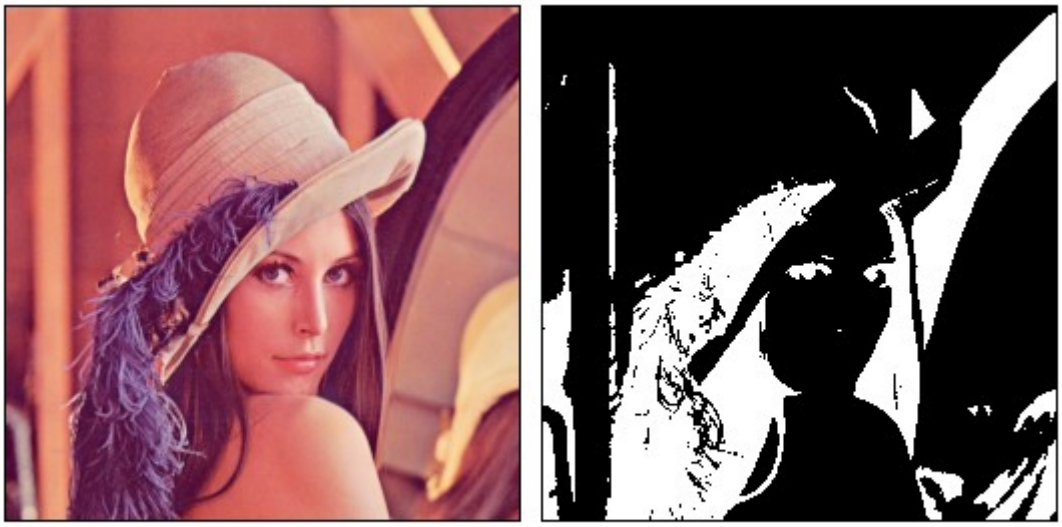
dst imagen de salida del mismo tamaño que src y tipo cv.CV_8U.

El siguiente código muestra la utilización de la función **cv.inRange**.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// [R, G, B, A]
let low = new cv.Mat(src.rows, src.cols, src.type(), [0, 0, 0, 0]);
// [R, G, B, A]
let high = new cv.Mat(src.rows, src.cols, src.type(), [150, 150, 150, 255]);
// Puede probar con diferentes parámetros
cv.inRange(src, low, high, dst);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); low.delete(); high.delete();
```

4011_Imagen_InRange.html

Resultados



4.2.- Transformaciones geométricas de imágenes

Objetivos

- Aprenderemos a aplicar diferentes transformaciones geométricas a imágenes, como rotación, traslación, etc.
- Aprenderemos estas funciones: **cv.resize**, **cv.warpAffine**, **cv.getRotationMatrix2D**, **cv.getAffineTransform** y **cv.warpPerspective**

4.2.1.- cv.resize

Escalar es simplemente cambiar el tamaño de la imagen. OpenCV viene con una función cv.resize() para este propósito.

El tamaño de la imagen se puede especificar manualmente o puede especificar el factor de escala. Se utilizan diferentes métodos de interpolación.

Los métodos de interpolación preferibles son **cv.INTER_AREA** para reducir y **cv.INTER_CUBIC** (lento) y **cv.INTER_LINEAR** para hacer zoom. (Ver más abajo en el enlace)

Utilizamos la función:

cv.resize (src, dst, dsize, fx = 0, fy = 0, interpolation = cv.INTER_LINEAR)

Parámetros:

src imagen de entrada

dst imagen de salida; tiene el tamaño dsize (cuando no es cero) o el tamaño calculado a partir de src.size(), fx y fy; el tipo de dst es el mismo que el de src.

dsize tamaño de la imagen de salida; si es igual a cero, se calcula como:

dsize=Size(round(fx * src.cols), round(fy * src.rows))

Tanto dsize como fx y fy deben ser distintos de cero

fx factor de escala a lo largo del eje horizontal; cuando es igual a 0, se calcula como:

fx = (double) dsize.width/src.cols

fy factor de escala a lo largo del eje vertical; cuando es igual a 0, se calcula como:

fy = (double) dsize.height/src.rows

interpolation método de interpolacion (ver cv.InterpolationFlags)

El siguiente código muestra la aplicación de la función **cv.resize**

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(300, 300);
// Puede probarcon diferentes parámetros
cv.resize(src, dst, dsize, 0, 0, cv.INTER_AREA);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4021_escalado_imagen.html

Resultados

canvasInput canvasOutput

4.2.2.- cv.warpAffine

La traslación es el cambio de ubicación del objeto. Si se conoce el desplazamiento en la dirección (x,y), sea (tx,ty), podemos crear la matriz de transformación M de la siguiente manera:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Entonces utilizamos la función:

cv.warpAffine (src, dst, M, dsize, flags = cv.INTER_LINEAR,
borderMode = cv.BORDER_CONSTANT, borderValue = new cv.Scalar())

Parámetros:

src	imagen de entrada.
dst	imagen de salida dst que tiene el tamaño dsize y el mismo tipo que src.
M	matriz de transformation 2 × 3 de type cv.CV_64FC1.
dsize	tamaño de la imagen de salida.
flags	combinación de métodos de interpolación (ver cv.InterpolationFlags) y el flag opcional WARP_INVERSE_MAP que significa que M es la transformación inversa (**dst** → **src**)
borderMode	método de extrapolación de píxeles borderMode (ver cv.BorderTypes); cuando borderMode = BORDER_TRANSPARENT, significa que la función no modifica los píxeles de la imagen de destino correspondientes a los "valores atípicos" de la imagen de origen.
borderValue	valor utilizado en el caso un border constante. Por defecto es 0..

El siguiente código muestra una aplicación de la función cv.warpAffine:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.matFromArray(2, 3, cv.CV_64FC1, [1, 0, 50, 0, 1, 100]);
let dsize = new cv.Size(src.rows, src.cols);
// Puede probar con parámetros diferentes
cv.warpAffine(src, dst, M, dsize, cv.INTER_LINEAR,
              cv.BORDER_CONSTANT, new cv.Scalar());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4022_traslacion.html

Resultados

4.2.3.- cv.getRotationMatrix2D

La rotación de una imagen un ángulo θ se logra mediante la matriz de transformación de la forma:

$$M = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Pero OpenCV proporciona una rotación escalada con un centro de rotación ajustable para que pueda rotar en cualquier lugar que queramos. La matriz de transformación modificada viene dada por:

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha) * center.x - \beta * center.y \\ -\beta & \alpha & \beta * center.x + (1-\alpha) * center.y \end{bmatrix}$$

donde:

$\alpha = escala*\cos\theta$

$\beta = escala*sen\theta$

Utilizaremos para la rotación escalada lafunción:

cv.getRotationMatrix2D (center, angle, scale)

Parámetros:

center centro de rotación en la imagen fuente.

angle ángulo de rotación en grados. Los valores positivos significan rotación en sentido contrario a las agujas del reloj (el origen de coordenadas se asume en la parte superior izquierda.).

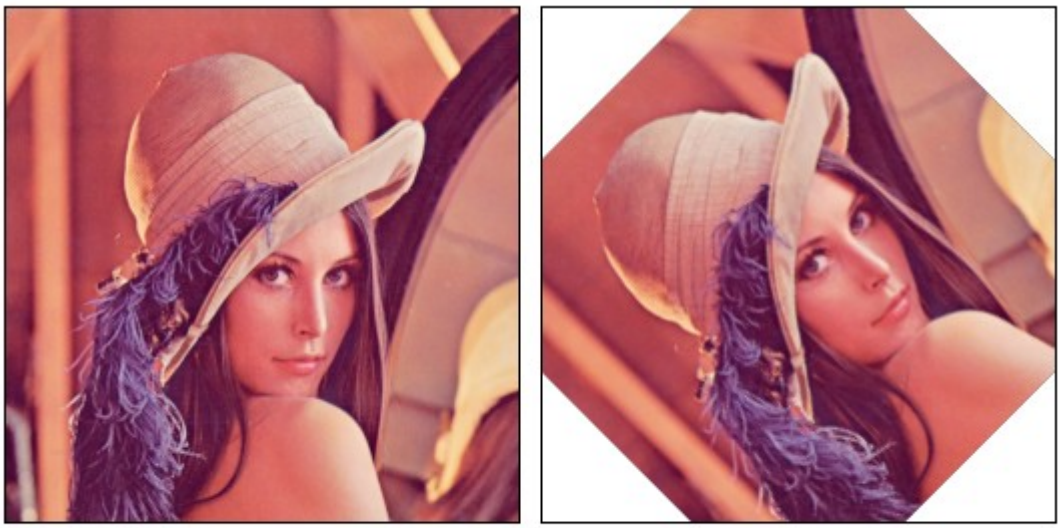
scale factor isotrópico de escala.

El siguiente código muestra una aplicación de la función cv.getRotationMatrix2D:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(src.rows, src.cols);
let center = new cv.Point(src.cols / 2, src.rows / 2);
// Puede probar con diferentes parámetros
let M = cv.getRotationMatrix2D(center, 45, 1);  // (centro, angulo
antihorario, escala)
cv.warpAffine(src, dst, M, dsize, cv.INTER_LINEAR,
              cv.BORDER_CONSTANT, new cv.Scalar());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4023_rotacion.html

Resultados



canvasInput canvasOutput

4.2.4.- cv.getAffineTransform

En la transformación afín, todas las líneas paralelas en la imagen original seguirán siendo paralelas en la imagen de salida. Para encontrar la matriz de transformación, necesitamos tres puntos de la imagen de entrada y sus ubicaciones correspondientes en la imagen de salida.

Luego, cv.getAffineTransform creará una matriz de 2x3 que se pasará a cv.warpAffine para generar la imagen de salida.

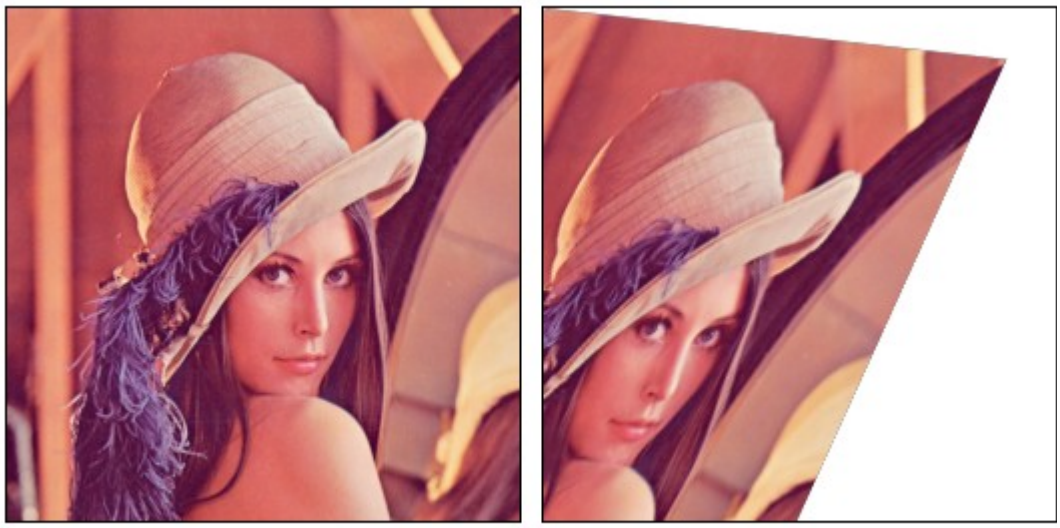
El siguiente código muestra una aplicación de la función cv.getAffineTransform:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
```

```
// (data32F[0], data32F[1]) es el primer punto
// (data32F[2], data32F[3]) es el segundo punto
// (data32F[4], data32F[5]) es el tercer punto
let srcTri = cv.matFromArray(3, 1, cv.CV_32FC2, [0, 0, 0, 1, 1, 0]);
let dstTri = cv.matFromArray(3, 1, cv.CV_32FC2, [0.6, 0.2, 0.1, 1.3, 1.5,
0.3]);
let dsize = new cv.Size(src.rows, src.cols);
let M = cv.getAffineTransform(srcTri, dstTri);
// Puede probar con diferentes valores
cv.warpAffine(src, dst, M, dsize, cv.INTER_LINEAR, cv.BORDER_CONSTANT,
new cv.Scalar());
cv.imshow('canvasOutput', dst);
 src.delete(); dst.delete(); M.delete(); srcTri.delete(); dstTri.delete();
```

4024_transformacion_afin.html

Resultados



4.2.5.- cv.warpPerspective

Para la transformación de perspectiva, necesitamos una matriz de transformación de 3x3. Las líneas rectas permanecerán rectas incluso después de la transformación.

Para encontrar esta matriz de transformación, necesitamos 4 puntos en la imagen de entrada y los puntos correspondientes en la imagen de salida. Entre estos 4 puntos, 3 de ellos no deben ser colineales.

Entonces la matriz de transformación se puede encontrar mediante la función **cv.getPerspectiveTransform**.

Luego aplicamos **cv.warpPerspective** con la matriz de transformación de 3x3 obtenida en el paso anterior.

Utilizaremos la función:

```
cv.warpPerspective (src, dst, M, dsize, flags = cv.INTER_LINEAR,
                    borderMode = cv.BORDER_CONSTANT,
                    borderValue = new cv.Scalar())
```

Parámetros:

src	imagen de entrada
dst	imagen de salida que tiene el tamaño dsize y el mismo tipo que src
M	Matriz de transformación 3 × 3 (tipo cv.CV_64FC1)
dsize	tamaño de la imagen de salida.
flags	combinación de métodos de interpolación (cv.INTER_LINEAR o cv.INTER_NEAREST) y la bandera opcional WARP_INVERSE_MAP, que

	establece M como la transformación inversa (dst → src)
borderMode	método de extrapolación de píxeles borderMode (cv.BORDER_CONSTANT o cv.BORDER_REPLICATE)
borderValue	utilizado en el caso de un borde constante; por defecto, es 0.

Y la función que nos proporciona la matriz de transformación M:

M = cv.getPerspectiveTransform (src, dst)

Parámetros:

src imagen de origen.

dst imagen de salida.

El siguiente código muestra una aplicación de la función cv.warpPerspective:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let dsize = new cv.Size(src.rows, src.cols);
// (data32F[0], data32F[1]) is the first point
// (data32F[2], data32F[3]) is the sescond point
// (data32F[4], data32F[5]) is the third point
// (data32F[6], data32F[7]) is the fourth point
let srcTri = cv.matFromArray(4, 1, cv.CV_32FC2, [56, 65, 368, 52, 28, 387,
389, 390]);
let dstTri = cv.matFromArray(4, 1, cv.CV_32FC2, [0, 0, 300, 0, 0, 300, 300,
300]);
let M = cv.getPerspectiveTransform(srcTri, dstTri);
// Puede probar con diferentes parámetros
cv.warpPerspective(src, dst, M, dsize, cv.INTER_LINEAR, cv.BORDER_CONSTANT,
new cv.Scalar());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete(); srcTri.delete(); dstTri.delete();
```

4025_transformacion_perspectiva.html

Resultados

4.3.- Humbralización de imagen

Objetivos

Aprenderemos a convertir imágenes en imágenes binarias mediante umbralización global, umbralización adaptativa, binarización de Otsu, etc.

Aplicaremos funciones como: **cv.threshold**, **cv.adaptiveThreshold**, etc.

4.3.1.- Umbralización simple

Aquí, el asunto es sencillo. Si el valor del píxel es mayor que un valor de umbral, se le asigna un valor (puede ser blanco), de lo contrario, se le asigna otro valor (puede ser negro). La forma de encontrar los valores que se asignan depende del tipo de umbralización en OpenCV.

Utilizamos la función:

cv.threshold (src, dst, thresh, maxval, type)

Parámetros:

src	matriz de entrada
dst	matriz de salida del ***mismo tamaño y tipo y el mismo número de canales*** que src.
thres	valor del umbral
maxval	valor máximo para usar con los tipos de umbral `cv.THRESH_BINARY` y `cv.THRESH_BINARY_INV`
type	el tipo de umbral (consulte cv.ThresholdTypes).

Tipos de umbralización en OpenCV.

OpenCV proporciona diferentes estilos de umbral y se seleccionan por el quinto parámetro de la función. Los diferentes tipos son:

- cv.THRESH_BINARY
- cv.THRESH_BINARY_INV
- cv.THRESH_TRUNC
- cv.THRESH_TOZERO
- cv.THRESH_OTSU
- cv.THRESH_TRIANGLE

NOTA:

La imagen de entrada debe ser de un solo canal solo en el caso de los tipos cv.THRESH_OTSU o cv.THRESH_TRIANGLE.

El siguiente código muestra cómo utilizar la función cv.threshold:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Puede probar con diferentes parámetros
cv.threshold(src, dst, 177, 200, cv.THRESH_BINARY);
cv.imshow('canvasOutput', dst);
```

4031_umbralización_simple.html

Resultados

En el ejemplo anterior, cuando el valor del pixel de un canal de la matriz src es inferior a 177, el correspondiente valor en dst se pone a cero. Por el contrario, si dcho valor es superior a 177 en el canal de src, su correspondiente valor es dst se pone a 200.

4.3.2.- Umbralización adaptativa

En la sección anterior, usamos un valor global como valor de umbral. Pero puede que no sea bueno en todas las condiciones en las que la imagen tiene diferentes condiciones de iluminación en diferentes áreas. En ese caso, optamos por el umbral adaptativo. En este caso, el algoritmo calcula el umbral para una pequeña región de la imagen.

Entonces obtenemos diferentes umbrales para diferentes regiones de la misma imagen y nos da mejores resultados para imágenes con iluminación variable.

Utilizamos la función:

cv.adaptiveThreshold (src, dst, maxValue, adaptiveMethod, thresholdType, blockSize, C)

Parámetros:

src	imagen fuente de **8-bit y un solo canal**
dst	imagen de salida del mismo tamaño y del mismo tipo que srcc.
maxValue	valor distinto de cero asignado a los píxeles para los que se cumple la condición
adaptiveMethod	algoritmo de umbralización adaptativa a utilizar (ver abajo)
thresholdType	tipo de umbral que debe ser cv.THRESH_BINARY o cv.THRESH_BINARY_INV
blockSize	tamaño de una vecindad de píxeles que se utiliza para calcular un valor de umbralx para el píxel: 3, 5, 7, etc.
C	constante restada de la media o media ponderada (ver los detalles a continuación). Normalmente, es positivo, pero también puede ser cero o negativo.

AdaptiveMethod - decide cómo se calcula el valor de umbral:

- **cv.ADAPTIVE_THRESH_MEAN_C**
- **cv.ADAPTIVE_THRESH_GAUSSIAN_C**

El siguiente código muestra la utilización de la función cv.adaptiveThreshold:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
```

```
// El último parámetro de la función es el número de canales de la matriz de
// salida. Si el valor es cero, el nº de canales se deduce de src y
// cv.COLOR_RGBA2GRAY
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
// Puede probar diferentes parámetros
cv.adaptiveThreshold(src, dst, 200, cv.ADAPTIVE_THRESH_GAUSSIAN_C,
cv.THRESH_BINARY, 3, 2);
cv.imshow('canvasOutput', dst);
src.delete();
dst.delete();
```

4032_umbralizacion_adaptativa.html

Resultados

canvasInput canvasOutput

4.4.- Suavizado de Imágenes

Objetivos

- Desenfocar las imágenes con varios filtros de paso bajo
- Aplicar filtros personalizados a las imágenes (convolución 2D)

4.4.1.- Convolución 2d (filtrado de imágenes)

Al igual que en las señales unidimensionales, las imágenes también se pueden filtrar con varios filtros de paso bajo (LPF), filtros de paso alto (HPF), etc. LPF ayuda a eliminar ruidos, desenfocar las imágenes, etc. Los filtros HPF ayudan a encontrar bordes en el imágenes.

OpenCV proporciona una función **cv.filter2D()** para convolucionar un kernel con una imagen. Como ejemplo, probaremos un filtro promedio en una imagen. Un núcleo de filtro de promedio de 5x5 se verá como a continuación:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Utilizaremos la función:

cv.filter2D (src, dst, ddepth, kernel, anchor = new cv.Point(-1, -1),
delta = 0, borderType = cv.BORDER_DEFAULT)

Parámetros:

src	imagen de entrada
dst	imagen de salida del mismo tamaño y el mismo número de canales que src
ddepth	profundidad deseada de la imagen de destino.
kernel	kernel de convolución (o más bien un kernel de correlación), una matriz de punto flotante de un solo canal; si desea aplicar diferentes núcleos a diferentes canales, divida la imagen en planos de color separados usando split y procéselos individualmente.
anchor	ancla del kernel que indica la posición relativa de un punto filtrado dentro del kernel; el ancla debe estar dentro del núcleo; el valor predeterminado es new cv.Point(-1, -1) y significa que el ancla está en el centro del kernel.
delta	valor opcional agregado a los píxeles filtrados antes de almacenarlos en dst.
borderType	método de extrapolación de píxeles (ver cv.BorderTypes).

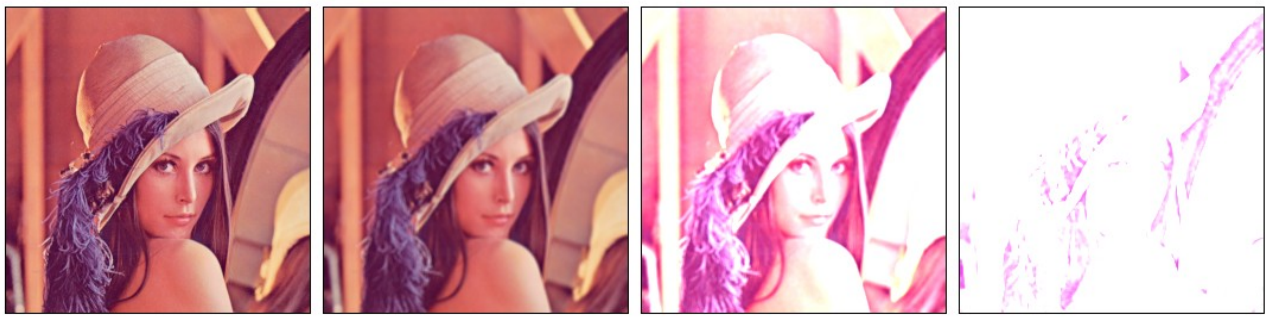
El código incluido a continuación muestra la utilización de l afunción cv.filter2D:

```
let src = cv.imread('canvasInput');
let src = cv.imread('canvasInput');
let dst = new cv.Mat();

// M es una matriz de unos de 3x3, 1 canal y datos cv.CV32F
let M = cv.Mat.ones(3, 3, cv.CV_32FC1);
const temp = new cv.Mat.ones(M.rows, M.cols, cv.CV_32FC1);

M1 = M.mul(temp,1/9);
M2 = M.mul(temp,1/4);
M3 = M.mul(temp,1)
```

```
let anchor = new cv.Point(-1, -1);

cv.filter2D(src, dst, cv.CV_8U, M1, anchor, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);

cv.filter2D(src, dst, cv.CV_8U, M2, anchor, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput2', dst);

cv.filter2D(src, dst, cv.CV_8U, M3, anchor, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput3', dst);
src.delete(); dst.delete(); M.delete();temp.delete()
M1.delete(); M2.delete(); M3.delete()
```

4041_convolucion_2D.html

Resultados



4.4.2.- Desenfoque de la imagen

El desenfoque de la imagen se logra convolucionando la imagen con un núcleo de filtro de paso bajo. Es útil para eliminar ruidos. En realidad, elimina el contenido de alta frecuencia (p. ej., ruido, bordes) de la imagen.

Así que los bordes estarán un poco borrosos en esta operación. (Bueno, hay técnicas de difuminado que tampoco difuminan los bordes). OpenCV proporciona principalmente cuatro tipos de técnicas de desenfoque. Los veremos de uno en uno en los siguientes apartados.

4.4.2.1.- Desenfoque por promedio

Esto se hace convolucionando la imagen con un filtro de kernel normalizado. Simplemente toma el promedio de todos los píxeles debajo del área del kernel y reemplaza el elemento central. Esto lo hacen las funciones **cv.blur()** o **cv.boxFilter()**.

Consulte los documentos para obtener más detalles sobre el kernel. Debemos especificar el ancho y la altura del kernel. Un filtro de kernel normalizado de 3x3 se vería como a continuación:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Usaremos la función:

```
cv.blur (src, dst, ksize, anchor = new cv.Point(-1, -1),
        borderType = cv.BORDER_DEFAULT)
```

Parámetros:

src imagen de entrada; puede tener cualquier número de canales, que se procesan de forma independiente, pero la profundidad debe ser: CV_8U, CV_16U, CV_16S, CV_32F o CV_64F.

dst imagen de salida del mismo tamaño y tipo que src

ksize tamaño del kernel.

anchor punto de anclaje; anchor = new cv.Point(-1, -1) significa que el ancla está en el centro del kernel.

borderType modo de borde utilizado para extrapolar píxeles fuera de la imagen (ver cv.BorderTypes).

La otra función:

```
cv.boxFilter (src, dst, ddepth, ksize, anchor = new cv.Point(-1, -1),
            normalize = true, borderType = cv.BORDER_DEFAULT)
```

Parámetros:

src imagen de entrada

dst imagen de salida del mismo tamaño y tipo que src.

ddepth la profundidad de la imagen de salida (-1 para usar src. depth()).

anchor punto de anclaje; anchor = new cv.Point(-1, -1) significa que el ancla está en el centro del kernel

normalize bandera, especificando si el núcleo está normalizado por su área o no.

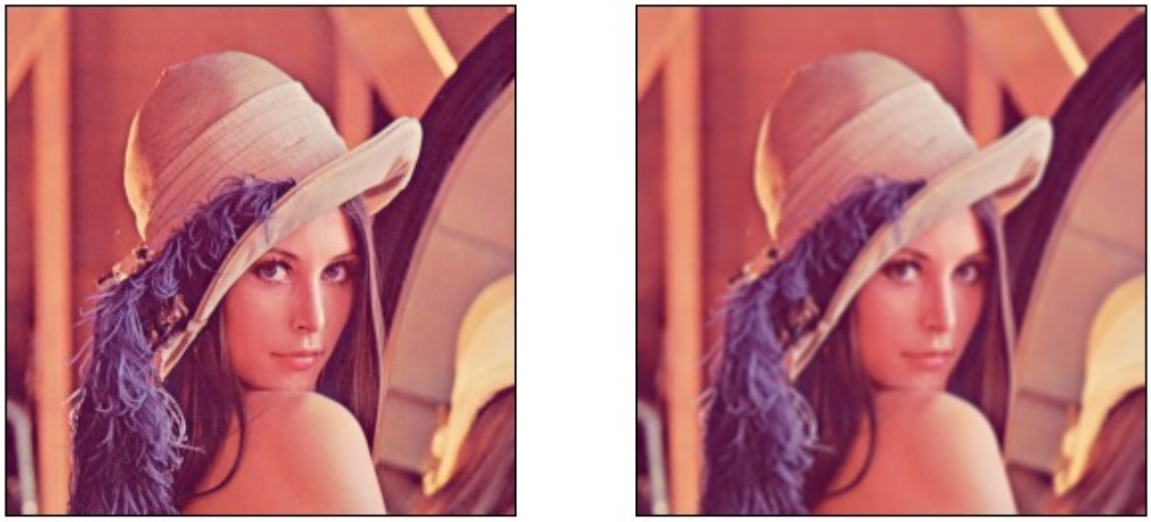
borderType modo de borde utilizado para extrapolar píxeles fuera de la imagen . (ver cv.BorderTypes).

El siguiente código muestra la utilización de la función cv.blur:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let ksize = new cv.Size(3, 3);
let anchor = new cv.Point(-1, -1);
// Puede probar con diferentes parámetros
cv.blur(src, dst, ksize, anchor, cv.BORDER_DEFAULT);
// cv.boxFilter(src, dst, -1, ksize, anchor, true, cv.BORDER_DEFAULT)
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4042_desenfoque_por_promedio.html

Resultados



4.4.2.2.- Desenfoque Gaussiano

En este caso, en lugar del filtro de caja, se usa el núcleo gaussiano.

Utilizamos la función:

cv.GaussianBlur (src, dst, ksize, sigmaX, sigmaY = 0, borderType = cv.BORDER_DEFAULT)

Parámetros:

src imagen de entrada; la imagen puede tener cualquier número de canales, que se procesan de forma independiente, pero la profundidad debe ser CV_8U, CV_16U, CV_16S, CV_32F o CV_64F.

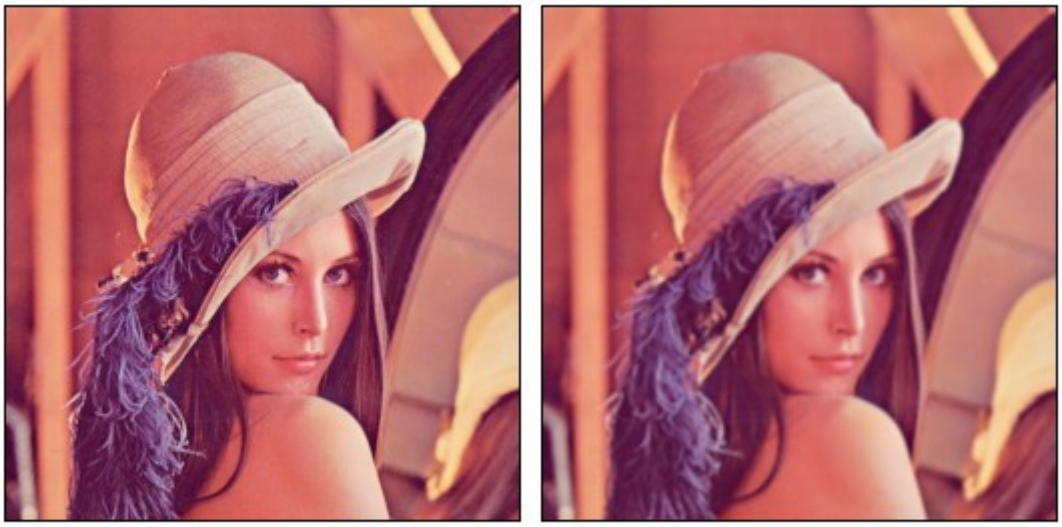
dst	imagen de salida del mismo tamaño y tipo que src.
ksize	tamaño del kernel
sigmaX	Desviación estándar del núcleo gaussiano en la dirección X.
sigmaY	Desviación estándar del núcleo gaussiano en la dirección Y; si sigmaY es cero, se establece para que sea igual a sigmaX, si ambos sigmas son ceros, se calculan a partir de ksize.width y ksize.height. Para controlar completamente el resultado independientemente de posibles modificaciones futuras de toda esta semántica, se recomienda especificar todo: ksize, sigmaX y sigmaY.
borderType	método de extrapolación de píxeles (ver cv.BorderTypes).

El siguiente código muestra una aplicación de la función cv.GaussianBlur.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let ksize = new cv.Size(3, 3);
// Puede probar con diferentes parámetros
cv.GaussianBlur(src, dst, ksize, 0, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4043_desenfoque_gaussiano.html

Resultados



canvasInput canvasOutput

4.4.2.3.- Desenfoque por mediana

Aquí, la función **cv.medianBlur()** toma la mediana de todos los píxeles debajo del área del kernel y su elemento central se reemplaza con este valor de la mediana. Esto es muy efectivo contra el ruido de sal y pimienta en las imágenes.

Lo interesante es que, en los filtros anteriores, el elemento central es un valor recién calculado que puede ser un valor de píxel que ya estaba en la imagen o un valor nuevo.

Pero en el desenfoque por mediana, el elemento central siempre se reemplaza por algún valor de píxel en la imagen. Reduce el ruido de manera efectiva. Su tamaño de núcleo debe ser un número entero impar positivo.

Utilizaremos la función:

cv.medianBlur (src, dst, ksize)

Parametros:

src imagen de entrada de 1, 3 o 4 canales; cuando ksize es 3 o 5, la profundidad de la imagen debe ser cv.CV_8U, cv.CV_16U o cv.CV_32F, para tamaños de apertura más grandes, solo puede ser cv.CV_8U.

dst matriz de destino del mismo tamaño y tipo que src.

ksize tamaño de apertura lineal; debe ser impar y mayor que 1, por ejemplo: 3, 5, 7…

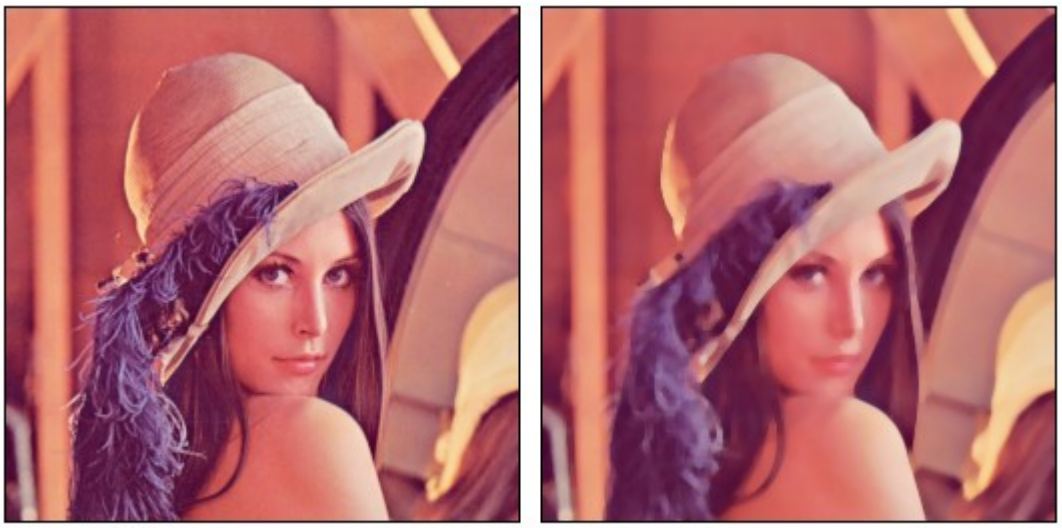
El siguiente código muestra una aplicación de la función medianBlur:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Puede probar con diferentes parámetros
```

```
cv.medianBlur(src, dst, 5);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4044_desenfoque_por_mediana.html

Resultados



canvasInput **canvasOutput**

4.4.2.4.- Filtrado bilateral

cv.bilateralFilter() es altamente efectivo en la eliminación de ruido mientras mantiene los bordes nítidos. Pero el funcionamiento es más lento en comparación con otros filtros. Ya vimos que el filtro gaussiano toma la vecindad alrededor del píxel y encuentra su promedio ponderado gaussiano. El filtro gaussiano es una función del espacio únicamente, es decir, los píxeles cercanos se consideran durante el filtrado. No considera si los píxeles tienen casi la misma intensidad. No considera si el píxel es un píxel de borde o no. Entonces también difumina los bordes, lo cual no queremos hacer. La función cv.bilateralFilter() resuelve este problema.

Utilizamos la función:

cv.bilateralFilter (src, dst, d, sigmaColor, sigmaSpace,
borderType = cv.BORDER_DEFAULT)

Parámetros:

src	imagen fuente de 8 bits o punto flotante, de 1 canal o de 3 canales.
dst	imagen de salida del mismo tamaño y tipo que src
d	diámetro de cada vecindad de píxeles que se utiliza durante el filtrado. Si no es positivo, se calcula a partir de sigmaSpace.

sigmaColor filtro sigma en el espacio de color. Un valor mayor del parámetro significa que se mezclarán más colores dentro de la vecindad de píxeles, lo que dará como resultado áreas más grandes de color semi-igual.

sigmaSpace filtro sigma en el espacio de coordenadas. Un valor mayor del parámetro significa que los píxeles más lejanos se influirán entre sí siempre que sus colores estén lo suficientemente cerca. Cuando d>0, especifica el tamaño de la vecindad independientemente de sigmaSpace. De lo contrario, d es proporcional a sigmaSpace.

borderType tipo de borde utilizado para extrapolar píxeles fuera de la imagen (ver cv.BorderTypes).

El siguiente código muestra un ejemplo de utilización de la función bilateralFilter:

```
 // Bilateral Filter Example
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB, 0);
// Pude probar con diferentes parámetros
cv.bilateralFilter(src, dst, 9, 75, 75, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4045_filtrado_bilateral.html

Resultados

canvasInput canvasOutput

4.5.- Transformaciones morfológicas

Aprenderemos sobre transformaciones morfológicas como Erosión, Dilatación, Apertura, Cierre, etc.

4.5.1.- Erosión

La idea básica de la erosión es como la erosión del suelo, erosiona los límites del objeto en primer plano (siempre trate de mantener el primer plano en blanco). Entonces, ¿qué hace? El núcleo se desliza a través de la imagen (como en la convolución 2D). Un píxel en la imagen original (ya sea 1 o 0) se considerará 1 solo si todos los píxeles debajo del núcleo son 1; de lo contrario, se erosiona (se hace cero).

Entonces, lo que sucede es que todos los píxeles cercanos a los límites se descartarán según el tamaño del kernel.

Por tanto, el grosor o el tamaño del objeto en primer plano disminuye o simplemente disminuye la región blanca en la imagen. Es útil para eliminar pequeños ruidos blancos (como hemos visto en el capítulo de espacio de color), separar dos objetos conectados, etc.

Utilizamos la función:

```
cv.erode (src, dst, kernel, anchor = new cv.Point(-1, -1), iterations = 1,
          borderType = cv.BORDER_CONSTANT,
          borderValue = cv.morphologyDefaultBorderValue())
```

Parámetros:

src imagen de entrada; el número de canales puede ser arbitrario, pero la profundidad debe ser cv.CV_8U, cv.CV_16U, cv.CV_16S, cv.CV_32F o cv.CV_64F.

dst imagen de salida del mismo tamaño y tipo que src.

kernel elemento estructurante utilizado para la erosión.

anchor posición del ancla dentro del elemento; el valor predeterminado new cv.Point(-1, -1) significa que el ancla está en el centro del elemento.

iterations número de veces que se aplica la erosión.

borderType método de extrapolación de píxeles (ver cv.BorderTypes).

borderValue valor de borde en caso de un borde constante

Las siguientes líneas de codigo muestran una aplicación de la función **cv.erode**:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.Mat.ones(5, 5, cv.CV_8U);
let anchor = new cv.Point(-1, -1);
// You can try more different parameters
cv.erode(src, dst, M, anchor, 1, cv.BORDER_CONSTANT,
cv.morphologyDefaultBorderValue());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4051_erosion.html

Resultados

4.5.2.- Dilatación

Es justo lo contrario de la erosión. Aquí, un elemento de píxel de salida es '1' si al menos un píxel debajo del kernel es '1'. Por lo tanto, aumenta la región blanca en la imagen o aumenta el tamaño del objeto en primer plano.

Normalmente, en casos como la eliminación de ruido, la erosión es seguida por la dilatación. Porque la erosión elimina los ruidos blancos, pero también encoge nuestro objeto. Entonces lo dilatamos. Dado que el ruido se ha ido, no volverá, pero el área de nuestro objeto aumenta. También es útil para unir partes rotas de un objeto.

Utilizamos la función:

```
cv.dilate (src, dst, kernel, anchor = new cv.Point(-1, -1), iterations = 1,
        borderType = cv.BORDER_CONSTANT,
        borderValue = cv.morphologyDefaultBorderValue())
```

Parámetros:

src	imagen de entrada; el número de canales puede ser arbitrario, pero la profundidad debe ser cv.CV_8U, cv.CV_16U, cv.CV_16S, cv.CV_32F o cv.CV_64F.
dst	imagen de salida del mismo tamaño y tipo que la entrada
kernel	elemento estructurante utilizado para la dilatación.
anchor	posición del ancla dentro del elemento; el valor predeterminado new cv.Point(-1, -1) significa que el ancla está en el centro del elemento.
iterations	número de veces que se aplica la dilatación.
borderType	método de extrapolación de píxeles (ver cv.BorderTypes).
borderValue	valor de borde en caso de un borde constante

Las siguientes líneas de codigo muestran una aplicación de la función **cv.dilate:**

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.Mat.ones(5, 5, cv.CV_8U);
let anchor = new cv.Point(-1, -1);
// Puede probar con diferentes parámetros
cv.dilate(src, dst, M, anchor, 1, cv.BORDER_CONSTANT,
          new cv.Scalar(255,0,0));
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4052_dilatacion.html

Resultados

canvasInput canvasOutput

4.5.3.- Apertura

Apertura = Erosión + Dilatación

La apertura es solo otro nombre para la erosión seguida de la dilatación. Es útil para eliminar el ruido.

Utilizamos la función:

cv.morphologyEx (src, dst, op, kernel,
anchor = new cv.Point(-1, -1),
iterations = 1,
borderType = cv.BORDER_CONSTANT,
borderValue = cv.morphologyDefaultBorderValue())

Parámetros:

src	imagen de entrada. El número de canales puede ser arbitrario. La profundidad debe ser una de cv.CV_8U, cv.CV_16U, cv.CV_16S, cv.CV_32F o cv.CV_64F
dst	imagen de destino del mismo tamaño y tipo que la imagen de origen.
op	tipo de una operación morfológica, (ver cv.MorphTypes).

kernel	elemento estructurante. Se puede crear usando cv.getStrucuringElement.
anchor	posición de anclaje con el núcleo. Los valores negativos significan que el ancla está en el centro del kernel.
iterations	número de veces que se aplica la dilatación.
borderType	método de extrapolación de píxeles (ver cv.BorderTypes).
borderValue	valor de borde en caso de un borde constante. El valor predeterminado tiene un significado especial.

cv.MorphTypes debe tomar uno de los valores de la siguiente enumeración:

```
enum MorphTypes
{
   MORPH_ERODE    = 0,
   MORPH_DILATE   = 1,
   MORPH_OPEN     = 2,
   MORPH_CLOSE    = 3,
   MORPH_GRADIENT = 4,
   MORPH_TOPHAT   = 5,
   MORPH_BLACKHAT = 6,
   MORPH_HITMISS  = 7,
};
```

Las siguientes líneas de codigo muestran una aplicación de la función **cv.morphologyEx** en donde hemos aplicado también **op = cv.MORPH_OPEN.**

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.Mat.ones(5, 5, cv.CV_8U);
let anchor = new cv.Point(-1, -1);
// Puede probar con diferentes parámetros
cv.morphologyEx(src, dst, cv.MORPH_OPEN, M, anchor, 1,
       cv.BORDER_CONSTANT, cv.morphologyDefaultBorderValue());
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4053_apertura.html

Resultados

canvasInput canvasOutput

El ruido blanco desaparece mientras que la letra situada en la región sin ruido queda igual.

4.5.4.- Cierre

Cierre = Dilatación + Erosión

El cierre es el reverso de la apertura, la dilatación seguida de la erosión. Es útil para cerrar pequeños agujeros dentro de los objetos de primer plano o pequeños puntos negros en el objeto.

La función es la misma utilizada en los apartados anteriores en la que hemos dejado algunos parámetros con su valor por defecto y hemos utilizado la constante cv.MORPH_CLOSE.

cv.morphologyEx(src, dst, cv.MORPH_CLOSE, M);

Las siguientes líneas de código muestran la utilización de la función cv.morphologyEx para aplicar un cierre a una imagen:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let M = cv.Mat.ones(5, 5, cv.CV_8U);
// Puede probar con diferentes parámetros
cv.morphologyEx(src, dst, cv.MORPH_CLOSE, M);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

Resultados

canvasInput canvasOutput

El ruido negro desaparece mientras que la letra situada en la región sin ruido queda igual.

4.5.5.- Gradiente morfológico

Gradiente morfológico = Dilatación - Erosión

El gradiente es la diferencia entre dilatación y erosión de una imagen.

El resultado se verá como el contorno del objeto.

La función es la misma utilizada en los apartados anteriores en la que hemos dejado algunos parámetros con su valor por defecto y hemos utilizado la constante cv.MORPH_GRADIENT.

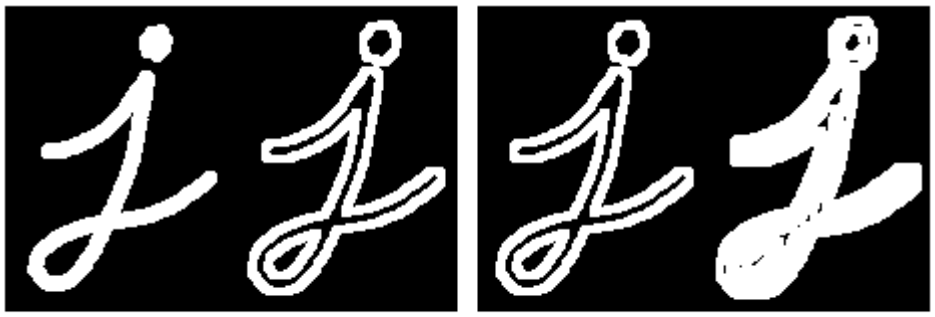
cv.morphologyEx(src, dst, cv.MORPH_GRADIENT, M);

El siguiente código muestra una aplicación del gradiente morfológico:

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB);
let dst = new cv.Mat();
let M = cv.Mat.ones(5, 5, cv.CV_8U);
// Puede probar con diferentes parámetros
cv.morphologyEx(src, dst, cv.MORPH_GRADIENT, M);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4055_gradiente_morfologico.html

Resultados



canvasInput canvasOutput

4.5.6.- TopHat

TopHat = Imagen Entrada - Apertura

TopHat es la diferencia entre imagen de entrada y la apertura de la imagen.

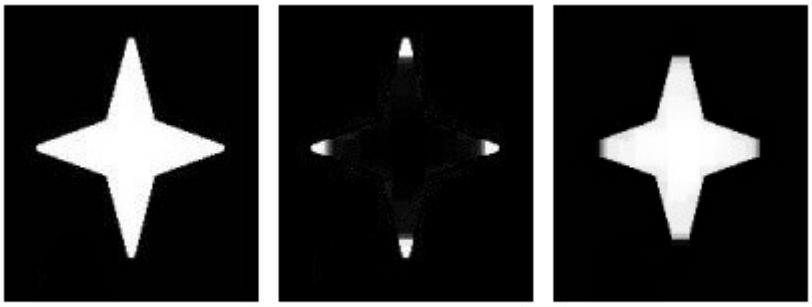
Las siguientes líneas de código muestran la imagen de entrada en canvasInput, el TopHat en canvasOutput y la apertura en canvasOutput2.

```
let src = cv.imread('canvasInput');
```

```
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB);
let dst = new cv.Mat();
let M = cv.Mat.ones(9, 9, cv.CV_8U);
// Puede probar con diferentes parámetros
cv.morphologyEx(src, dst, cv.MORPH_TOPHAT, M);
cv.imshow('canvasOutput', dst);

// La apertura
cv.morphologyEx(src, dst, cv.MORPH_OPEN, M)
cv.imshow('canvasOutput2', dst)
src.delete(); dst.delete(); M.delete();
```

4056_top_hat.html

Resultados



canvasInput canvasOutput canvasOutput2

4.5.7.- BlackHat

BlackHat = Cierre – Imagen Entrada

El BlackHat es la diferencia entre el cierre de la imagen de entrada y la imagen de entrada.

Se utiliza la misma función que en apartados anteriores con la constante cv.MORPH_BLACKHAT :

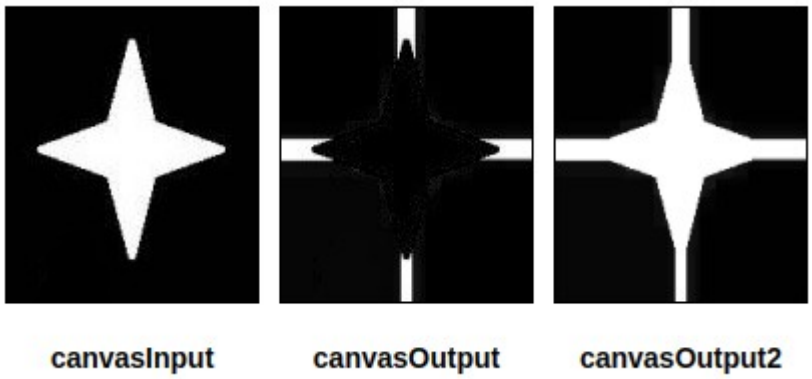
cv.morphologyEx(src, dst, cv.MORPH_BLACKHAT, M);

Las siguientes líneas de código muestran la imagen de entrada en canvasInput, el BlackHat en canvasOutput y el cierre en canvasOutput2.

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB);
let dst = new cv.Mat();
let M = cv.Mat.ones(53, 53, cv.CV_8U);
// You can try more different parameters
cv.morphologyEx(src, dst, cv.MORPH_BLACKHAT, M);
cv.imshow('canvasOutput', dst);

// Cierre
cv.morphologyEx(src, dst, cv.MORPH_CLOSE, M);
cv.imshow('canvasOutput2', dst);
src.delete(); dst.delete(); M.delete();
```

4057_black_hat.html

Resultados



4.5.8.- Elementos estructurales

En los ejemplos anteriores creamos manualmente elementos estructurantes con la ayuda de **cv.Mat.ones**. Son de forma rectangular. Pero en algunos casos, es posible que necesite núcleos de forma elíptica o circular. Entonces, para este propósito, OpenCV tiene la función:

cv.getStructuringElement().

Pasando a la función la forma y el tamaño del kernel, se obtiene el kernel deseado.

La funcion que vamos a utilizar tiene los siguientes parámetros:

cv.getStructuringElement (shape, ksize,

anchor = new cv.Point(-1, -1))

Parámetros:

shape forma del elemento que podría ser una de cv.MorphShapes

- MORPH_RECT -> elemento estructural rectangular

  (Eij=1)

- MORPH_CROSS -> elemento estructural en forma de cruz

  (Eij = { 1, if i=anchor.y or j=anchor.x ; 0 otherwise}

- MORPH_ELLIPSE -> elíptico, es decir, una elipse rellena inscrita en el rectángulo Rect(0, 0, ksize.width, ksize.height)

ksize tamaño del elemento estructurante.

anchor posición de anclaje dentro del elemento. El valor predeterminado [−1,−1] significa que

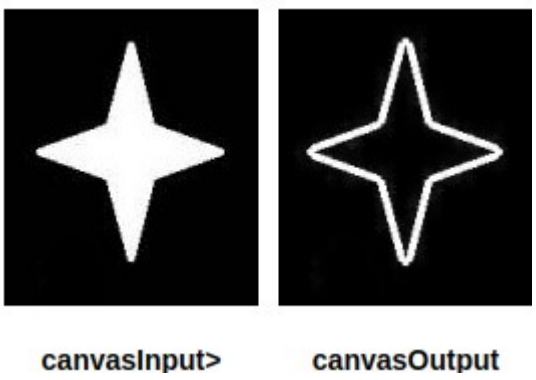
el ancla está en el centro. Tenga en cuenta que solo la forma de un elemento en forma de cruz depende de la posición del ancla. En otros casos, el ancla simplemente regula cuánto se desplaza el resultado de la operación morfológica.

El siguiente código muestra el efecto de procesar una imagen de entrada mediante la función morphologyEx , la constante cv.MORPH_GRADIENT y un núcleo cv.MORPH_CROSS de tamaño 5 x 5:

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB);
let dst = new cv.Mat();
let M = new cv.Mat();
let ksize = new cv.Size(5, 5);
// Puede probar con diferentes valores de los parámetros
M = cv.getStructuringElement(cv.MORPH_CROSS, ksize);
cv.morphologyEx(src, dst, cv.MORPH_GRADIENT, M);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); M.delete();
```

4058_elemento_estructural.html

Resultados



4.6.- Gradiente de imagen

Objetivo

- Encontrar los gradientess de imagen, bordes, etc.
- Aprender las siguientes funciones: **cv.Sobel(), cv.Scharr()**, **cv.Laplacian()** etc.

OpenCV proporciona tres tipos de filtros de gradiente o filtros de paso alto: Sobel, Scharr y Laplacian.

Veremos cada uno de ellos.

4.6.1.- Derivadas de Sobel y Scharr

Los operadores de Sobel son una operación conjunta de suavizado gaussiano más diferenciación, por lo que es más resistente al ruido. Puede especificar la dirección de las derivadas a tomar, vertical u horizontal (mediante los argumentos, **yorder** y **xorder** respectivamente). También puede especificar el tamaño del núcleo mediante el argumento **ksize**. Si ksize = -1, se utiliza un filtro Scharr de 3x3 que da mejores resultados que un filtro Sobel de 3x3. Consulte los documentos para los núcleos utilizados.

Utilizaremos las funciones:

cv.Sobel (src, dst, ddepth, dx, dy, ksize = 3, scale = 1,

delta = 0, borderType = cv.BORDER_DEFAULT)

Parámetros

src	imagen de entrada
dst	imagen de salida del mismo tamaño y el mismo número de canales que src.
ddepth	profundidad de la imagen de salida (ver más abajo Combinaciones de profundidad de pixel); en el caso de imágenes de entrada de 8 bits, dará como resultado derivadas truncadas.
dx	orden de la derivada x.
dy	orden de la derivada y.
ksize	tamaño del núcleo Sobel extendido; debe ser 1, 3, 5 o 7.
scale	factor de escala opcional para los valores derivados calculados.
delta	valor delta opcional que se agrega a los resultados antes de almacenarlos en dst.
borderType	método de extrapolación de pixeles (see cv.BorderTypes).

La otra función:

cv.Scharr (src, dst, ddepth, dx, dy, scale = 1, delta = 0,

borderType = cv.BORDER_DEFAULT)

Parámetros

src	imagen de entrada
dst	imagen de salida del mismo tamaño y el mismo número de canales que src.
ddepth	profundidad de la imagen de salida (ver más abajo Combinaciones de profundidad

	de pixel); en el caso de imágenes de entrada de 8 bits, dará como resultado derivadas truncadas.
dx	orden de la derivada x.
dy	orden de la derivada y.
scale	factor de escala opcional para los valores derivados calculados.
delta	valor delta opcional que se agrega a los resultados antes de almacenarlos en dst.
borderType	pixel extrapolation method (see cv.BorderTypes).

Nota

Las funciones y clases descritas en esta sección se utilizan para realizar varias operaciones de filtrado lineal o no lineal en imágenes 2D (representadas como Mat). Significa que para cada ubicación de píxel (x,y) en la imagen de origen (normalmente, rectangular), se considera su vecindad y se usa para calcular la respuesta. En el caso de un filtro lineal, es una suma ponderada de valores de píxeles. En el caso de operaciones morfológicas, son los valores mínimo o máximo, y así sucesivamente. La respuesta calculada se almacena en la imagen de destino en la misma ubicación (x,y). Significa que la imagen de salida tendrá el mismo tamaño que la imagen de entrada. Normalmente, las funciones admiten matrices multicanal, en cuyo caso cada canal se procesa de forma independiente. Por tanto, la imagen de salida también tendrá el mismo número de canales que la de entrada.

Otra característica común de las funciones y clases descritas en esta sección es que, a diferencia de las funciones aritméticas simples, necesitan extrapolar valores de algunos píxeles no existentes. Por ejemplo, si desea suavizar una imagen con un filtro gaussiano de 3 × 3, al procesar los píxeles más a la izquierda en cada fila, necesita píxeles a la izquierda de ellos, es decir, fuera de la imagen. Puede dejar que estos píxeles sean los mismos que los píxeles de la imagen que se encuentran más a la izquierda (método de extrapolación de "borde duplicado"), o suponer que todos los píxeles no existentes son ceros (método de extrapolación de "borde constante"), y así sucesivamente. OpenCV le permite especificar el método de extrapolación. Para obtener más información, consulte más arriba cv.BorderTypes.

Combinaciones de profundidad de pixel

Profundidad de entrada (depth)	Profundidad de salida (**ddepth**)
CV_8U	-1/CV_16S/CV_32F/CV_64F
CV_16U/CV_16S	-1/CV_32F/CV_64F
CV_32F	-1/CV_32F/CV_64F
CV_64F	-1/CV_64F

Cuando ddepth = -1, la imagen de salida tendrá la misma profundidad que la de entrada.

La tabla anterior nos da la profundidad que puede tener la imagen de salida en función de la profundidad de la imagen de entrada.

El siguiente código muestra una aplicación de las funciones cv.Sobel y cv.Scharr.

```
let src = cv.imread('canvasInput');
let dstx = new cv.Mat();
let dsty = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGB2GRAY, 0);
// Puede probar con diferentes parámetros
cv.Sobel(src, dstx, cv.CV_8U, 1, 0, 3, 1, 0, cv.BORDER_DEFAULT);
cv.Sobel(src, dsty, cv.CV_8U, 0, 1, 3, 1, 0, cv.BORDER_DEFAULT);
// cv.Scharr(src, dstx, cv.CV_8U, 1, 0, 1, 0, cv.BORDER_DEFAULT);
// cv.Scharr(src, dsty, cv.CV_8U, 0, 1, 1, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutputx', dstx);
cv.imshow('canvasOutputy', dsty);
src.delete(); dstx.delete(); dsty.delete();
```

4061_derivadas_sobel_scharr.html

Resultados

canvasInput canvasOutputx canvasOutputy

4.6.2.- Laplaciano

Calcula el Laplaciano de la imagen dado por la relación: Δ(src) = ∂2src/∂2x2 + ∂2src/∂2y2, donde cada derivada se encuentra utilizando derivadas de Sobel. Si **ksize = 1**, se utiliza el siguiente kernel para filtrar:

$$\text{kernel} = \begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

Utilizamos la función:

cv.Laplacian (src, dst, ddepth, ksize = 1, scale = 1,

delta = 0, borderType = cv.BORDER_DEFAULT)

Parámetros

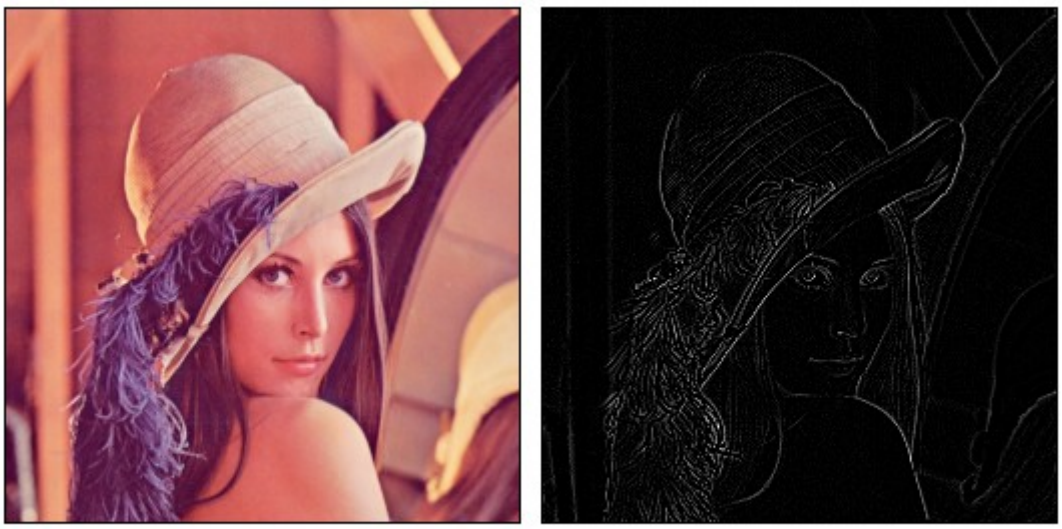
src	imagen de entrada
dst	imagen de salida del mismo tamaño y el mismo número de canales que src.
ddepth	profundidad de la imagen de salida.
ksize	tamaño de apertura utilizado para calcular los filtros de segunda derivada.
scale	factor de escala opcional para los valores laplacianos calculados.
delta	valor delta opcional que se agrega a los resultados antes de almacenarlos en dst.
borderType	método de extrapolación de píxeles (ver cv.BorderTypes).

El siguiente código muestra una aplicación de la función cv.Laplacian:

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGB2GRAY, 0);
// Puede probar diferentes parámetros
cv.Laplacian(src, dst, cv.CV_8U, 1, 1, 0, cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4062_laplaciano.html

Resultados



canvasInput canvasOutput

4.6.3.- AbsSobel

¡Un asunto importante!

En nuestro último ejemplo, el tipo de datos de salida es cv.CV_8U. Pero hay un pequeño problema con eso.

La transición de negro a blanco se toma como una pendiente positiva (tiene un valor positivo) mientras que la transición de blanco a negro se toma como una pendiente negativa (tiene un valor negativo). Entonces, cuando convierte datos a cv.CV_8U, todas las pendientes negativas se vuelven cero. En palabras simples, pierdes ese borde.

Si desea detectar ambos bordes, la mejor opción es mantener el tipo de datos de salida en algunas formas superiores, como cv.CV_16S, cv.CV_64F, etc., tome su valor absoluto y luego vuelva a convertirlo a cv.CV_8U. El siguiente código demuestra este procedimiento para un filtro Sobel horizontal y la diferencia en los resultados.

Utilizaremos la función auxiliar cv.convertScaleAbs:

(Escala, calcula valores absolutos y convierte el resultado a 8 bits.)

cv.convertScaleAbs (InputArray src, OutputArray dst,

double alpha=1, double beta=0)

Parámetros

src matriz de entrada

dst matriz de salida

alpha factor de escala opcional

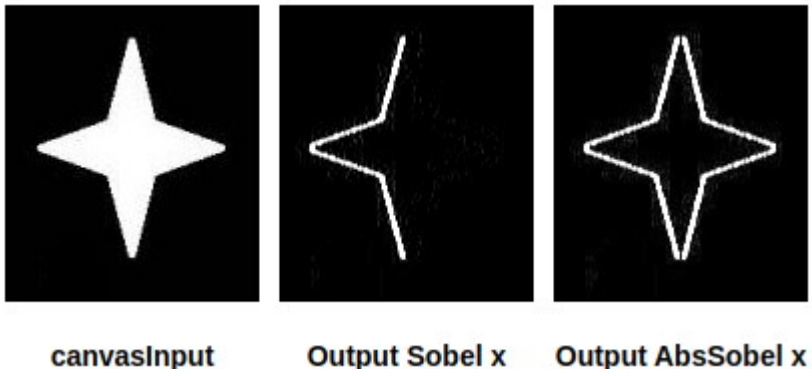
beta delta opcional añadido a los valores escalados

El siguiente código muestra un ejemplo de aplicación:

```
let src = cv.imread('canvasInput');
let dstx = new cv.Mat();
let absDstx = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGB2GRAY, 0);
// Puede probar con diferentes parámetros
cv.Sobel(src, dstx, cv.CV_8U, 1, 0, 3, 1, 0, cv.BORDER_DEFAULT);
cv.Sobel(src, absDstx, cv.CV_64F, 1, 0, 3, 1, 0, cv.BORDER_DEFAULT);
cv.convertScaleAbs(absDstx, absDstx, 1, 0);
cv.imshow('canvasOutput8U', dstx);
cv.imshow('canvasOutput64F', absDstx);
src.delete(); dstx.delete(); absDstx.delete();
```

4063_abs_sobel.html

Resultados



4.7.- Detección de bordes Canny

Aprenderemos a encontrar bordes con Canny Edge Detection.

Ojetivos

- Aprender sobre el concepto de detección de bordes
- Utilizar la funcion Canny de OpenCV para este objetivo: **cv.Canny()**

“Canny Edge Detection” es un popular algoritmo de detección de bordes. Fue desarrollado por John F. Canny en 1986.

Es un algoritmo de varias etapas y pasaremos por cada etapa.

1. Reducción de ruido

Dado que la detección de bordes es susceptible al ruido en la imagen, el primer paso es eliminar el ruido en la imagen con un filtro gaussiano de 5x5. Ya lo hemos visto en capítulos anteriores.

2. Hallar el gradiente de intensidad de la imagen

La imagen suavizada se filtra luego con un kernel de Sobel en dirección horizontal y vertical para obtener la primera derivada en dirección horizontal (Gx) y en dirección vertical (Gy). A partir de estas dos imágenes, podemos encontrar el gradiente del borde y la dirección del gradiente en cada píxel de la siguiente manera:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

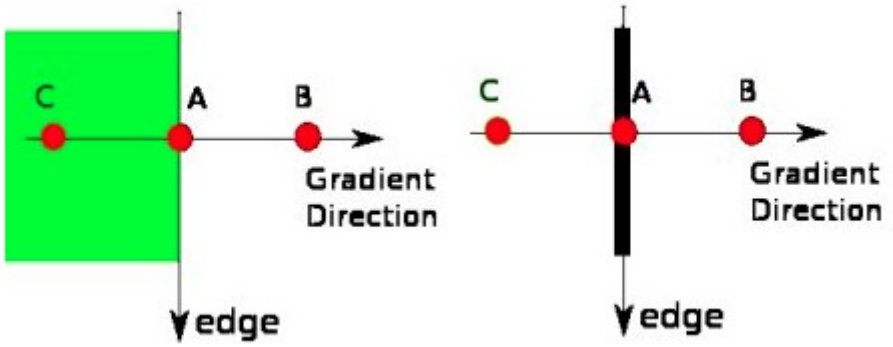
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

La dirección del gradiente siempre es perpendicular al borde. Se redondea a uno de los cuatro ángulos que representan las direcciones vertical, horizontal y dos diagonales.

3. Supresión no máxima

Después de obtener la dirección y la magnitud del gradiente, se realiza un escaneo completo de la imagen para eliminar los píxeles no deseados que pueden no constituir el borde. Para esto, en cada píxel, se verifica si el píxel es un máximo local en su vecindad en la dirección del gradiente. Revise la imagen a continuación:

El punto A está en el borde (en dirección vertical). La dirección del gradiente es normal al borde.



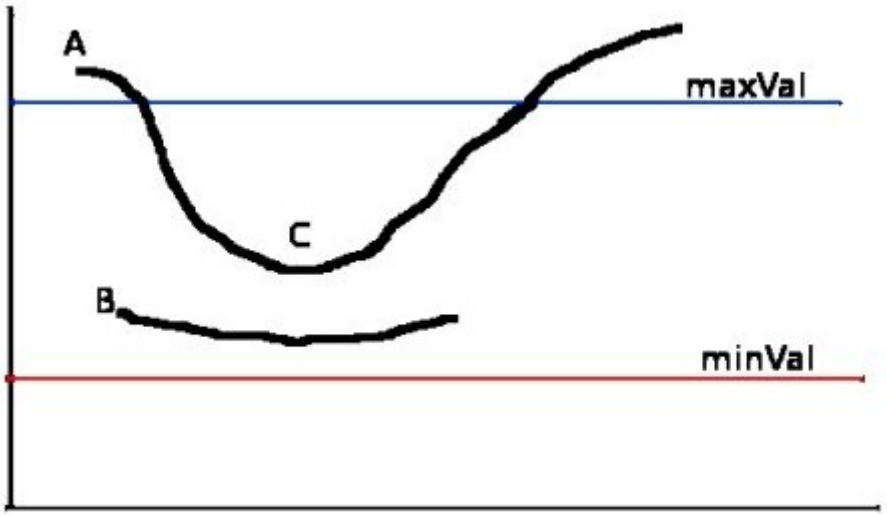
Los puntos B y C están en direcciones de gradiente. Entonces, el punto A se verifica con el punto B y C para ver si forma un máximo local. Si es así, se considera para la siguiente etapa, de lo contrario, se suprime (se pone a cero).

En resumen, el resultado que se obtiene es una imagen binaria con "bordes delgados".

4. Histéresis Umbral

Esta etapa decide cuáles son todos los bordes que son realmente bordes y cuáles no lo son. Para esto, necesitamos dos valores de umbral, **minVal** y **maxVal.** Cualquier borde con un gradiente de intensidad superior a maxVal seguramente será borde y aquellos por debajo de minVal seguramente no serán bordes, por lo tanto, deséchelos. Aquellos que se encuentran entre estos dos umbrales se clasifican como bordes o no bordes en función de su conectividad. Si están conectados a píxeles de

"borde seguro", se consideran parte de los bordes. De lo contrario, también se desechan. Vea la imagen a continuación:



El borde A está por encima de maxVal, por lo que se considera "borde seguro". Aunque el borde C está por debajo de maxVal, está conectado al borde A, por lo que también se considera un borde válido y obtenemos esa curva completa.

Pero el borde B, aunque está por encima de minVal y está en la misma región que el borde C, no está conectado a ningún "borde seguro", por lo que se descarta. Por lo tanto, es muy importante que seleccionemos minVal y maxVal en consecuencia para obtener el resultado correcto.

Esta etapa también elimina los pequeños ruidos de los píxeles asumiendo que los bordes son líneas largas.

Entonces, lo que finalmente obtenemos son bordes fuertes en la imagen.

4.7.1.- Detección de bordes Canny en OpenCV

Utilizaremos la función:

cv.Canny (image, edges, threshold1, threshold2, apertureSize = 3, L2gradient = false)

Parámetros:

image 8-bit matriz de entrada

edges mapa de bordes de salida; imagen de 8 bits de un solo canal, que tiene el mismo tamaño que la imagen de entrada

thresold1 primer umbral para el procedimiento de histéresis.

thresold2 segundo umbral para el procedimiento de histéresis.

apertureSize tamaño de apertura para el operador Sobel.

L2gradient especifica la ecuación para encontrar la magnitud del gradiente. Si es True, usa la ecuación mencionada anteriormente que es más precisa, de lo contrario usa esta función:

Edge_Gradient(G)=|Gx|+|Gy|

El siguiente código muestra un ejemplo de utilización de la funcion cv.Canny()

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGB2GRAY, 0);
// Puede probar con diferentes parámetros
cv.Canny(src, dst, 50, 100, 3, false);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4071_deteccion_bordes_canny.html

Resultados

4.8.- Pirámide de imágenes

Objetivos

- Aprenderemos sobre las pirámides de imágenes y cómo usarlas para combinar imágenes.
- Aprenderemos estas funciones: **cv.pyrUp()**, **cv.pyrDown()**

Normalmente, solíamos trabajar con una imagen de tamaño constante. Pero en algunas ocasiones, necesitamos trabajar con las mismas imágenes pero en diferente resolución. Por ejemplo, mientras buscamos algo en una imagen, como una cara, no estamos seguros de en qué tamaño estará presente el objeto en dicha imagen. En ese caso, necesitaremos crear un conjunto de mágenes de la misma imagen con diferentes resoluciones y buscar el objeto en todas ellas. Este conjunto de imágenes con diferentes resoluciones se denomina “Pirámides de imágenes “ (porque cuando se mantienen en una pila con la imagen de mayor resolución en la parte inferior y la imagen de menor resolución en la parte superior, parece una pirámide).

Hay dos tipos de pirámides de imágenes:

1. Pirámides Gaussianas
2. Pirámides Laplacianas

El nivel superior (baja resolución) en una pirámide gaussiana se forma eliminando filas y columnas consecutivas en la imagen de nivel inferior (resolución superior). Luego, cada píxel en el nivel superior está formado por la contribución de 5 píxeles en el nivel subyacente con pesos gaussianos. Al hacerlo, una imagen M×N se convierte en una imagen de M/2 x N/2. Entonces el área se reduce a un cuarto del área original. Se llama Octava. El mismo patrón continúa a medida que ascendemos en la pirámide (es decir, la resolución disminuye). De manera similar, mientras se expande, el área se vuelve 4 veces mayor en cada nivel. Podemos encontrar pirámides de Gauss usando las funciones **cv.pyrDown()** y **cv.pyrUp()**.

Las pirámides laplacianas se forman a partir de las pirámides gaussianas. No hay una función exclusiva para eso.

Las imágenes de la pirámide laplaciana son solo imágenes de borde. La mayoría de sus elementos son ceros.

Se utilizan en la compresión de imágenes. Un nivel en la Pirámide Laplaciana está formado por la diferencia entre ese nivel en la Pirámide Gaussiana y la versión expandida de su nivel superior en la Pirámide Gaussiana.

4.8.1.- Ejemplo pyrDown

Utilizaremos la función:

cv.pyrDown (src, dst, dstsize = new cv.Size(0, 0), borderType = cv.BORDER_DEFAULT)

Parámetros:

src imagen de entrada

dst imagen de salida; tiene el tamaño especificado y el mismo tipo que src.

dstsize tamaño de la imagen de salida. De forma predeterminada, el tamaño de la imagen de salida se calcula como Size ((src.cols+1)/2, (src.rows+1)/2), pero en cualquier caso, se deben cumplir las siguientes condiciones:

$$|\texttt{dstsize.width} * 2 - src.cols| \leq 2$$
$$|\texttt{dstsize.height} * 2 - src.rows| \leq 2$$

borderType método de extrapolación de píxeles (consulte cv.BorderTypes,

cv.BORDER_CONSTANT no es compatible).

El siguiente código muestra un ejemplo de aplicación de la función **cv.pyrDown**.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// You can try more different parameters
cv.pyrDown(src, dst, new cv.Size(0, 0), cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

Resultados

4.8.2.- Ejemplo PyrUp

La función que utilizamos es:

cv.pyrUp (src, dst, dstsize = new cv.Size(0, 0), borderType = cv.BORDER_DEFAULT)

Parámetros:

src	imagen de entrada
dst	imagen de salida; debe ser del mismo tamaño y del mismo tipo que la imagen de entrada src
dstsize	tamaño de la imagen de salida
borderType	método de extrapolación de pixeles; (ver cv.BorderTypes, solo cv.BORDER_DEFAULT está soportado).

El siguiente código muestra un ejemplo de aplicación de la función **cv.pyrUp**.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
// Puede probar con diferentes parámetros
cv.pyrUp(src, dst, new cv.Size(0, 0), cv.BORDER_DEFAULT);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

Resultados

4.9.- Contornos en OpenCV.js

Aprenderos sobre contornos en OpenCV.js.

En este apartado veremos los siguientes subapartados:

- **Empezando con contornos**

  Aprender a encontrar y dibujar contornos.

- **Características de los contornos**

  Aprender a encontrar diferentes características de contornos como área, perímetro, rectángulo delimitador, etc.

- **Propiedades de los contornos**

  Aprender a encontrar diferentes propiedades de contornos como Solidez, Intensidad media, etc.

- **Más funciones sobre contornos**

  Aprender a encontrar defectos de convexidad, pointPolygonTest, encontrar coincidencias en diferentes formas, etc.

- **Jerarquía de contornos**

  Más información sobre la jerarquía de contornos

4.9.1.- Empezando con contornos

Objetivo:

- Comprender qué son los contornos.
- Aprender a encontrar contornos, dibujar contornos, etc.
- Aprender estas funciones: **cv.findContours()**, **cv.drawContours()**

4.9.1.1.- ¿Qué son los contornos?

Los contornos se pueden explicar simplemente como una curva que une todos los puntos continuos (a lo largo del límite), que tienen el mismo color o intensidad. Los contornos son una herramienta útil para el análisis de formas y la detección y reconocimiento de objetos.

Para una mayor precisión, utilice **imágenes binarias**. Entonces, antes de encontrar contornos, aplique umbral o detección de bordes canny.

Desde opencv 3.2 la imagen de origen no se modifica al procesar con esta función.

En OpenCV, encontrar contornos es como encontrar un objeto blanco en un fondo negro. Así que recuerda, el objeto a encontrar debe ser blanco y el fondo debe ser negro.

4.9.1.2.- ¿Cómo dibujar los contornos?

Para dibujar los contornos, se utiliza la función **cv.drawContours**. También se puede utilizar para dibujar cualquier forma siempre que tenga sus puntos de contorno.

- Para buscar contornos utilizaremos:

cv.findContours (image, contours, hierarchy, mode, method, offset = new cv.Point(0, 0))

Parámetros:

image	imagen fuente, una imagen de un solo canal de 8 bits. Los píxeles distintos de cero se tratan como 1. Los píxeles cero siguen siendo 0, por lo que la imagen se trata como binaria.
contours	contornos detectados
hierarchy	contiene información sobre la topología de la imagen. Tiene tantos elementos como número de contornos.
mode	modo de recuperación de contornos (ver cv.RetrievalModes).
method	método de aproximación de contorno (ver cv.ContourApproximationModes).

offset Desplazamiento opcional por el que se desplaza cada punto del contorno. Esto es útil si los contornos se extraen del ROI de la imagen y luego deben analizarse en el contexto de la imagen completa.

Método de aproximación de contorno

Nos referimos al quinto argumento **method** en la función **cv.findContours**. ¿Qué denota en realidad?

Arriba, dijimos que los contornos son los límites de una forma con la misma intensidad. Almacena las coordenadas (x,y) del límite de una forma. Pero, ¿almacena todas las coordenadas? Eso se especifica mediante este método de aproximación de contorno **method**.

Si pasa como cv.ContourApproximationModes a **CHAIN_APPROX_NONE**, se almacenan todos los puntos de límite. Pero, ¿realmente necesitamos todos los puntos? Por ejemplo, encontraste el contorno de una línea recta. ¿Necesitas todos los puntos en la línea para representar esa línea? No, solo necesitamos dos puntos finales de esa línea. Esto es lo que hace **cv.CHAIN_APPROX_SIMPLE**. Elimina todos los puntos redundantes y comprime el contorno, ahorrando así memoria.

- Para dibujar contornos utilizaremos:

```
cv.drawContours (image, contours, contourIdx, color,
                 thickness = 1, lineType = cv.LINE_8,
                  hierarchy = new cv.Mat(), maxLevel = INT_MAX,
                  offset = new cv.Point(0, 0))
```

Parámetros:

image imagen de destino

contours todos los contornos de entrada

contourIdx parámetro que indica un contorno a dibujar. Si es negativo, se dibujan todos los contornos.

color color de los contornos

thickness grosor de las líneas con las que se dibujan los contornos. Si es negativo, se dibujan los interiores del contorno.

lineType	conectividad de línea (ver cv.LineTypes)
hierarchy	información opcional sobre la jerarquía. Solo es necesario si desea dibujar solo algunos de los contornos (consulte maxLevel).
maxLevel	nivel máximo para contornos dibujados. Si es 0, solo se dibuja el contorno especificado. Si es 1, la función dibuja el(los) contorno(s) y todos los contornos anidados. Si es 2, la función dibuja los contornos, todos los contornos anidados, todos los contornos anidados anidados, etc. Este parámetro solo se tiene en cuenta cuando hay hierarchy disponible.
offset	parámetro de cambio de contorno opcional.

4.9.2.- Encontrar y dibujar contornos

Las siguientes líneas de código muestran como encontrar y dibujar contornos en una imagen:

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.cols, src.rows, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 120, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
// Puede probar con diferentes parámetros
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);

// dibuja contornos con un Scalar (color) aleatorio
for (let i = 0; i < contours.size(); ++i) {
     let color = new cv.Scalar(Math.round(Math.random() * 255),
                               Math.round(Math.random() * 255),
                               Math.round(Math.random() * 255));
     cv.drawContours(dst, contours, i, color, 1, cv.LINE_8, hierarchy, 100);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
```

Resultados

4.9.3.- Características de los contornos

Objetivos

- Encontrar las diferentes características de los contornos, como el área, el perímetro, el centroide, el rectángulo delimitador, etc. Aprenderemos muchas funciones relacionadas con los contornos.

4.9.3.1.- Momentos del contorno

Los momentos de una imagen le ayudan a calcular algunas características como el centro de masa del objeto, el área del objeto, etc. Consulte la página de wikipedia sobre Momentos de imagen.

Utilizamos la función:

cv.moments (array, binaryImage = false)

Parámetros:

array	imagen ráster (matriz 2D de un solo canal, 8 bits o coma flotante) o una matriz (1×N o N×1) de puntos 2D.
binaryImage	si es cierto, todos los píxeles de la imagen distintos de cero se tratan como 1. El parámetro se utiliza solo para imágenes.

Las siguientes líneas de código muestran como encontrar el contorno 0 de la imagen y representarlo. Posteriormente se calculan los momentos m00, m10 y m01 a partir de los cuales se calcula el centro de masas del contorno de la siguiente forma:

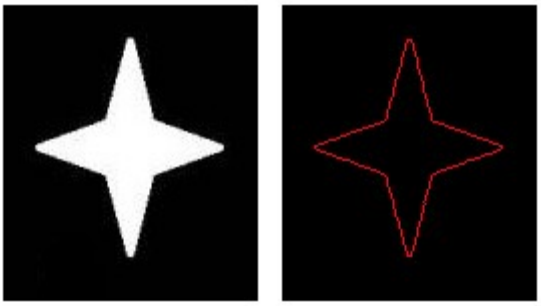
```
let cx = Moments.m10/M.m00
let cy = Moments.m01/M.m00
console.log(cx, cy)
```

El código de la función:

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
let Moments = cv.moments(cnt, false);
momentsOutput.innerHTML = Moments.m00;

// Representamos el contorno 0
cv.drawContours(dst, contours, 0, new cv.Scalar(255, 0, 0), 1, cv.LINE_8,
hierarchy, 1);
cv.imshow('canvasOutput', dst);

// Calculamos las coordenadas del centro de massas del contorno
let cx = Moments.m10/Moments.m00
let cy = Moments.m01/Moments.m00

console.log(cx, cy)

src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
```

4092a_momentos.html

Resultados



El m00 es: 2640

Obtenemos además:

cx = 61.65

cy = 69.80

Teniendo en cuenta que la imagen **shape.png cargada** en cancasInput es de 125x146 pixels el centro sale bastante bien colocado.

4.9.3.2.- Area del contorno

Para encontrar el área de un contorno utilizamos la imagen circulos.png. Calculamos todos los contornos y luego escogemos el contorno 0 con la instrucción:

```
let cnt = contours.get(0);
```

El listado del código que realiza dicha tarea es el siguiente

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
let area = cv.contourArea(cnt, false);
areaOutput.innerHTML = area;
src.delete(); contours.delete(); hierarchy.delete(); cnt.delete();
```

4092b_area.html

Resultados

canvasInput

El área es: 1117

4.9.3.3.- Perímetro del contorno

También se le llama longitud de arco. Se puede averiguar usando la función **cv.arcLength()**.

Utilizamos la función:

cv.arcLength (curve, closed)

Parámetros:

curve vector de entrada de puntos 2D.

closed Bandera que indica si la curva está cerrada o no.

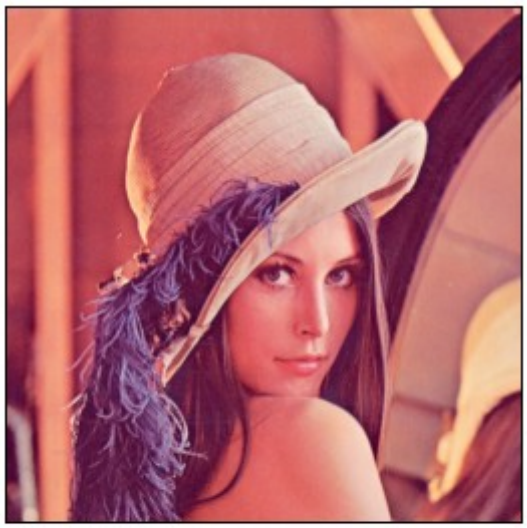
En el programa se utiliza la imagen de lena.png, se obtienen todos los contornos y finalmente se calcula el perímetro del contorno Nº 20.

Listado del programa:

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(20);
// Puede probar con diferentes parámetros
let perimeter = cv.arcLength(cnt, true);    // cv.arcLength (curve, closed)
perimeterOutput.innerHTML = perimeter;
src.delete(); contours.delete(); hierarchy.delete(); cnt.delete();
```

4092c_perimetro.html

Resultados



canvasInput

El perímetro es: 5.656854152679443

4.9.3.4.- Contorno aproximado

Aproxima una forma de contorno a otra forma con menos número de vértices dependiendo de la precisión que especifiquemos. Es una implementación del algoritmo de Douglas-Peucker. Consulte la página de wikipedia para ver el algoritmo y la demostración.

Utilice la función:

cv.approxPolyDP (curve, approxCurve, epsilon, closed)

Parámetros:

curve	vector de entrada de puntos 2D almacenados en cv.Mat.
approxCurve	resultado de la aproximación. El tipo debe coincidir con el tipo de la curva de entrada.
epsilon	parámetro que especifica la precisión de la aproximación. Esta es la distancia máxima entre la curva original y su aproximación.
closed	Si es verdadero, la curva aproximada se cierra (su primer y último vértices están conectados). De lo contrario, no está cerrado.

El siguiente códgo encuentra todos los contornos en la imagen lena.png, aproxima cada contorno a un polígono y finalmente dibuja todos los contornos aproximados.

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 100, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
let poly = new cv.MatVector();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
// aproxima cada contorno a un polígono
for (let i = 0; i < contours.size(); ++i) {
   let tmp = new cv.Mat();
   let cnt = contours.get(i);
   // You can try more different parameters
   cv.approxPolyDP(cnt, tmp, 3, true);
   poly.push_back(tmp);
   cnt.delete(); tmp.delete();
}
// dibuja lo contornos con un Scalar (color) aleatorio
for (let i = 0; i < contours.size(); ++i) {
   let color = new cv.Scalar(Math.round(Math.random() * 255),
                             Math.round(Math.random() * 255),
                             Math.round(Math.random() * 255));
    cv.drawContours(dst, poly, i, color, 1, 8, hierarchy, 0);
}
cv.imshow('canvasOutput', dst);
```

```
src.delete(); dst.delete(); hierarchy.delete(); contours.delete();
poly.delete();
```

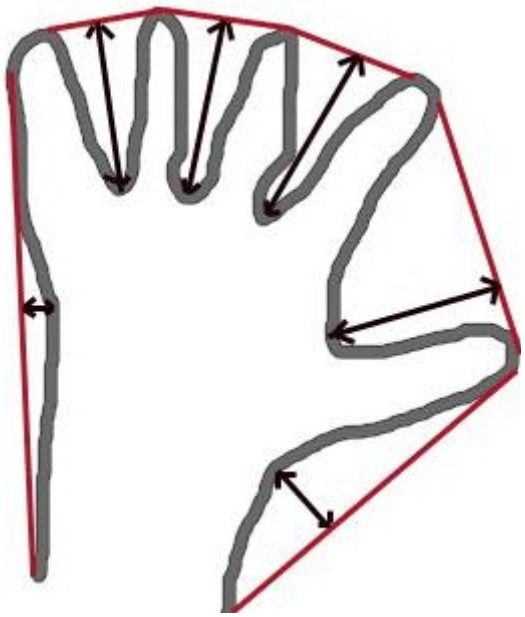
4092d_aproximacion_contorno.html

Resultados

canvasInput canvasOutput

4.9.3.5.- Convex Hull

Convex Hull se verá similar a la aproximación de contorno, pero no lo es (Ambos pueden proporcionar los mismos resultados en algunos casos). Aquí, la función **cv.convexHull()** verifica una curva en busca de defectos de convexidad y la corrige. En términos generales, las curvas convexas son las curvas que siempre están abultadas, o al menos planas. Y si está abombado por dentro, se llama defectos de convexidad. Por ejemplo, compruebe la siguiente imagen de la mano. La línea roja muestra el casco convexo de la mano. Las marcas de flechas de dos puntas muestran los defectos de convexidad, que son las desviaciones máximas locales del casco con respecto a los contornos.



Utilice la función:

cv.convexHull (points, hull, clockwise = false, returnPoints = true)

Parámetros:

points	conjunto de puntos 2D de entrada.
hull	casco convexo de salida.
clockwise	bandera de orientacion. Si es verdadero, el casco convexo de salida está orientado en el sentido de las agujas del reloj. De lo contrario, se orienta en sentido contrario a las agujas del reloj. El sistema de coordenadas asumido tiene su eje X apuntando hacia la derecha y su eje Y apuntando hacia arriba.
returnPoints	bandera de operación. En el caso de una matriz, cuando la bandera es verdadera, la función devuelve puntos de casco convexos. De lo contrario, devuelve índices de los puntos del casco convexo.

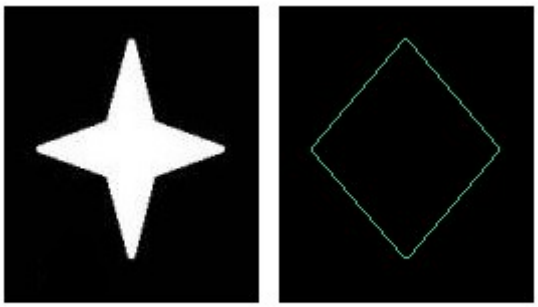
El siguiente código muestra una aplicación de la función **cv.convexHull**

```
let src = cv.imread('canvasInput');
```

```
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 100, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
let hull = new cv.MatVector();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
// aproxima cada contorno a convex hull
for (let i = 0; i < contours.size(); ++i) {
    let tmp = new cv.Mat();
    let cnt = contours.get(i);
    // Puede probar con diferentes parámetros
    cv.convexHull(cnt, tmp, false, true);
    hull.push_back(tmp);
     cnt.delete(); tmp.delete();
}
// dibuja los contornos con un Scalar (color) aleatorio
for (let i = 0; i < contours.size(); ++i) {
    let colorHull = new cv.Scalar(Math.round(Math.random() * 255),
                                  Math.round(Math.random() * 255),
                                  Math.round(Math.random() * 255));
    cv.drawContours(dst, hull, i, colorHull, 1, 8, hierarchy, 0);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); hierarchy.delete(); contours.delete();
hull.delete();
```

4092e_convex_hull.html

Resultados



canvasInput canvasOutput

4.9.3.6.- Chequeando convexidad

La funión **cv.isContourConvex()** permite determinar si una curva es convexa o no. Simplemente devuelve si es Verdadero o Falso cuando se le pasa un contorno como parámetro.

La función:

cv.isContourConvex(cnt)

Parámetros:

cnt contorno del que se quiere averiguar su convexidad

4.9.3.7.- Rectángulo delimitador recto

Es un rectángulo que circunscribe al objeto pero que no considera la rotación del objeto. Entonces, el área del rectángulo delimitador no será mínima.

Utilizamos la función:

cv.boundingRect (points)

Parámetros:

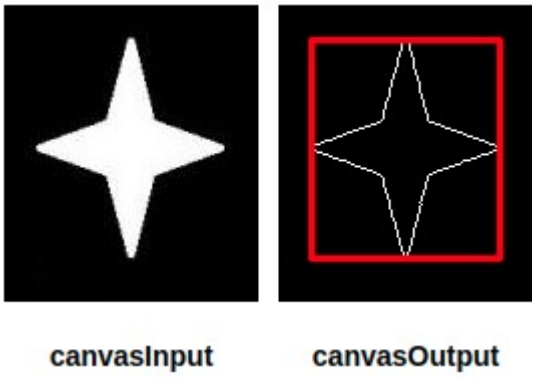
points conjunto de puntos 2D de entrada.

El siguiente código muestra un ejemplo de aplicación de la función **cv.isContourConvex()**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con otros valores de los parámetros
let rect = cv.boundingRect(cnt);
let contoursColor = new cv.Scalar(255, 255, 255);
let rectangleColor = new cv.Scalar(255, 0, 0);
cv.drawContours(dst, contours, 0, contoursColor, 1, 8, hierarchy, 100);
let point1 = new cv.Point(rect.x, rect.y);
let point2 = new cv.Point(rect.x + rect.width, rect.y + rect.height);
cv.rectangle(dst, point1, point2, rectangleColor, 2, cv.LINE_AA, 0);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
cnt.delete();
```

4092f_rectangulo_recto.html

Resultados



4.9.3.8.- Rectángulo delimitador girado

Aquí, el rectángulo delimitador se dibuja con un área mínima, por lo que también considera la rotación del objeto.

Utilizamos la función:

cv.minAreaRect (points)

Parámetros:

points conjunto de puntos 2D de entrada.

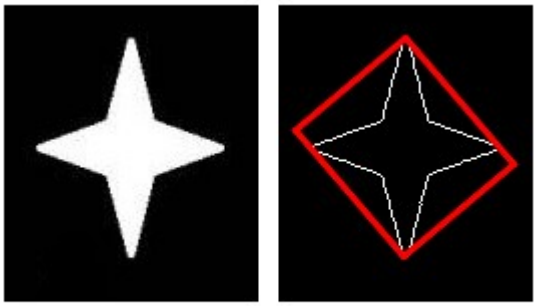
El siguiente código muestra un ejemplo de aplicación de la función cv.minAreaRect

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
let rotatedRect = cv.minAreaRect(cnt);
let vertices = cv.RotatedRect.points(rotatedRect);

// Ver los valores en la consola para entender el dibujo con cv.line()
console.log(vertices)

let contoursColor = new cv.Scalar(255, 255, 255);
let rectangleColor = new cv.Scalar(255, 0, 0);
cv.drawContours(dst, contours, 0, contoursColor, 1, 8, hierarchy, 100);
// draw rotatedRect
for (let i = 0; i < 4; i++) {
```

```
    cv.line(dst, vertices[i], vertices[(i + 1) % 4], rectangleColor, 2,
cv.LINE_AA, 0);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
cnt.delete();
```

4092g_rectangulo_girado.html

Resultados



4.9.3.9.- Mínimo círculo circunscrito

A continuación, encontramos el círculo circunscrito de un objeto mediante la función **cv.minEnclosingCircle()**. Es un círculo que cubre completamente el objeto con un área mínima.

Utilizamos las funciónes:

cv.minEnclosingCircle (points)

Parámetros:

points conjunto de puntos 2D de entrada.

cv.circle (img, center, radius, color, thickness = 1, lineType = cv.LINE_8, shift = 0)

Parámetros:

img imagen en donde se dibuja el círculo

center centro del círculo

radius radio del círculo.

color color del circulo

thickness grosor del contorno del círculo, si es positivo. Grosor negativo significa que se va a dibujar un círculo relleno.

lineType tipo de línea para dibujo del círculo.

shift número de bits fraccionarios en las coordenadas del centro y en el valor del radio.

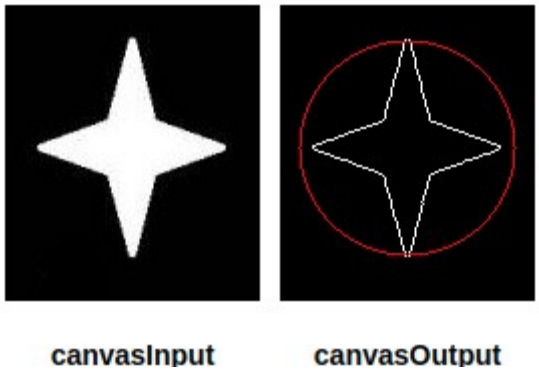
Las siguientes líneas muestran un ejemplo de aplicación de la funcion **cv.minEnclosingCircle**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
```

```
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
let circle = cv.minEnclosingCircle(cnt);
let contoursColor = new cv.Scalar(255, 255, 255);
let circleColor = new cv.Scalar(255, 0, 0);
cv.drawContours(dst, contours, 0, contoursColor, 1, 8, hierarchy, 100);
cv.circle(dst, circle.center, circle.radius, circleColor);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
cnt.delete();
```

4092h_minimo_circulo_circunscrito.html

Resultados



4.9.3.10.- Ajustar una elipse

Lo siguiente es ajustar una elipse a un objeto (contorno). La función **cv.fitEllipse (puntos)** devuelve el rectángulo girado en el que se inscribe la elipse para el conjunto de puntos (contorno) pasado com parámetro.

Utilizamos la función:

cv.fitEllipse (puntos)

Parámetros:

points conjunto de puntos 2D de entrada.

Y la otra función para dibujar la elipse:

cv.ellipse1 (img, box, color, thickness = 1, lineType = cv.LINE_8)

Parámetros:

img imagen

box representación de elipse alternativa a través de RotatedRect. Esto significa que la

función dibuja una elipse inscrita en el rectángulo rotado.

color color de elipse.

thickness espesor del contorno del arco de elipse, si es positivo. De lo contrario, esto indica que se va a dibujar un sector de elipse lleno.

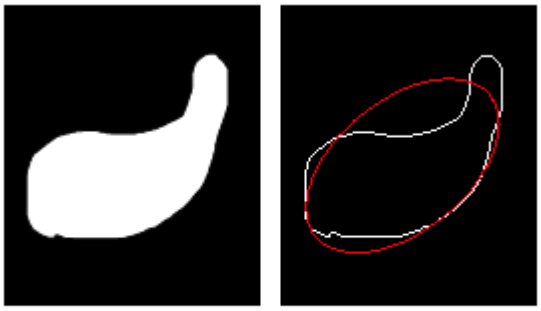
lineType tipo del línea de la elipse.

El siguiente código muestra una aplicación de las dos funciones anteriores.

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
let rotatedRect = cv.fitEllipse(cnt);
let contoursColor = new cv.Scalar(255, 255, 255);
let ellipseColor = new cv.Scalar(255, 0, 0);
cv.drawContours(dst, contours, 0, contoursColor, 1, 8, hierarchy, 100);
cv.ellipse1(dst, rotatedRect, ellipseColor, 1, cv.LINE_8);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
cnt.delete();
```

4092i_elipse.html

Resultados



4.9.3.11.- Ajustar una línea

De manera similar, podemos ajustar una línea recta a un conjunto de puntos.

Utilizaremos la funcion:

cv.fitLine (points, line, distType, param, reps, aeps)

Parámetros:

points	conjunto de puntos 2D de entrada.
line	parámetros de la línea de salida. Debe ser una Mat de 4 elementos[vx, vy, x0, y0], donde [vx, vy] es un vector normalizado colineal a la línea y [x0, y0] es un punto en la línea.
distType	distancia utilizada por el estimador M (ver cv.DistanceTypes).
param	parámetro numérico (C) para algunos tipos de distancias. Si es 0, se elige un valor óptimo.
reps	precisión suficiente para el radio (distancia entre el origen de coordenadas y la línea).
aeps	precisión suficiente para el ángulo. 0,01 sería un buen valor predeterminado para reps y aeps.

Y la otra función para dibujar la línea recta:

cv.line (img, pt1, pt2, color, thickness = 1, lineType = cv.LINE_8, shift = 0)

Parámetros:

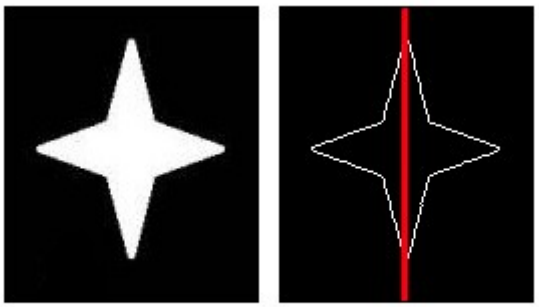
img	imagen
pt1	primer punto del segmento de línea.
pt2	segundo punto del segmento de línea.
color	color de linea.
thickness	grosor de la línea.
lineType	tipo de línea.
shift	número de bits fraccionarios en las coordenadas del punto.

El siguiente código muestra un ejemplo de aplicación de la funciones **cv.fitLine** y **cv.line**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
let line = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let cnt = contours.get(0);
// Puede probar con diferentes parámetros
cv.fitLine(cnt, line, cv.DIST_L2, 0, 0.01, 0.01);
let contoursColor = new cv.Scalar(255, 255, 255);
let lineColor = new cv.Scalar(255, 0, 0);
cv.drawContours(dst, contours, 0, contoursColor, 1, 8, hierarchy, 100);
let vx = line.data32F[0];
let vy = line.data32F[1];
let x = line.data32F[2];
let y = line.data32F[3];
let lefty = Math.round((-x * vy / vx) + y);
let righty = Math.round(((src.cols - x) * vy / vx) + y);
let point1 = new cv.Point(src.cols - 1, righty);
let point2 = new cv.Point(0, lefty);
cv.line(dst, point1, point2, lineColor, 2, cv.LINE_AA, 0);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
line.delete(); cnt.delete();
```

4092j_ajustar_recta.html

Resultados



4.9.4.- Propiedades de los contornos

Aquí aprenderemos a extraer algunas propiedades de los objetos que se usan con frecuencia, como la solidez, el diámetro equivalente, la imagen de máscara, la intensidad media, etc.

4.9.4.1.- Relación de aspecto

Es la relación entre el ancho y la alto del rectángulo delimitador del objeto (contorno).

$$Aspect\ Ratio = \frac{Width}{Height}$$

```
let rect = cv.boundingRect(cnt);
let aspectRatio = rect.width / rect.height;
```

4.9.4.2.- Extensión

La extensión es la relación entre el área del contorno y el área del rectángulo delimitador.

$$Extent = \frac{Object\ Area}{Bounding\ Rectangle\ Area}$$

```
let area = cv.contourArea(cnt, false);
let rect = cv.boundingRect(cnt);
let rectArea = rect.width * rect.height;
let extent = area / rectArea;
```

4.9.4.3.- Solidez

La solidez es la relación entre el área del contorno y el área convexa del casco.

$$Solidity = \frac{Contour\ Area}{Convex\ Hull\ Area}$$

```
let area = cv.contourArea(cnt, false);
cv.convexHull(cnt, hull, false, true);
```

```
let hullArea = cv.contourArea(hull, false);
let solidity = area / hullArea;
```

4.9.4.4.- Diámetro equivalente

El diámetro equivalente es el diámetro del círculo cuya área es igual al área del contorno.

$$Equivalent\ Diameter = \sqrt{\frac{4 \times Contour\ Area}{\pi}}$$

```
let area = cv.contourArea(cnt, false);
let equiDiameter = Math.sqrt(4 * area / Math.PI);
```

4.9.4.5.- Orientación

La orientación es el ángulo al que se dirige el objeto. El siguiente método también proporciona las longitudes del eje mayor y del eje menor.

```
let rotatedRect = cv.fitEllipse(cnt);
let angle = rotatedRect.angle;
```

4.9.4.6.- Valor Máximo, Valor Mínimo y sus ubicaciones

Utilizaremos la función:

cv.minMaxLoc(src, mask)

Parámetros:

src matriz de entrada de un solo canal.

mask máscara opcional utilizada para seleccionar un subarreglo.

```
let result = cv.minMaxLoc(src, mask);
let minVal = result.minVal;
let maxVal = result.maxVal;
let minLoc = result.minLoc;
let maxLoc = result.maxLoc;
```

4.9.4.7.- Color medio o intensidad media

Aquí, podemos encontrar el color promedio de un objeto. O puede ser la intensidad promedio del objeto en el modo de escala de grises. Volvemos a utilizar la misma máscara.

Utilizaremos la función:

cv.mean (src, mask)

Parámetros:

src matriz de entrada que debe tener de 1 a 4 canales para que el resultado se pueda almacenar en Scalar

mask máscara de operación opcional

```
let average = cv.mean(src, mask);
```

4.9.4.8.- Programa para el cálculo de las propiedades anteriores

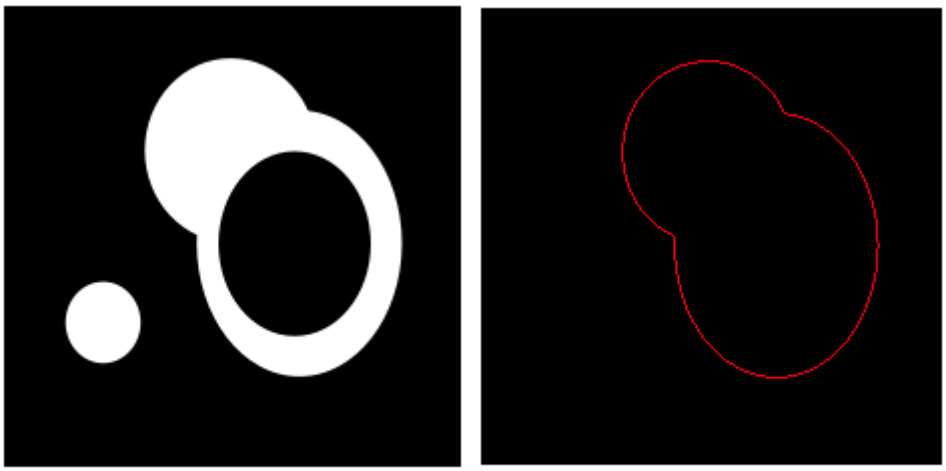
El siguiente programa calcula y muestra todas las propiedades definidas en los puntos anteriores.

```
let dst = cv.Mat.zeros(src.cols, src.rows, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 120, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
cv.drawContours(dst, contours, 1, new cv.Scalar(255, 0, 0), 1, cv.LINE_8,
hierarchy, 0);
cv.imshow('canvasOutput', dst);

// Elegimos el contorno 0 para los cálculos
let cnt = contours.get(0);

// Relación de aspecto
{
    let rect = cv.boundingRect(cnt);
    let aspectRatio = rect.width / rect.height;
    console.log('Relación de aspecto: ' + aspectRatio)
}

// Extensión
{
    let area = cv.contourArea(cnt, false);
    let rect = cv.boundingRect(cnt);
    let rectArea = rect.width * rect.height;
    let extent = area / rectArea;
    console.log(`Extensón: ${extent}`)
}

// Solidez
{
    let hull = new cv.Mat()
    let area = cv.contourArea(cnt, false);
    cv.convexHull(cnt, hull, false, true);
    let hullArea = cv.contourArea(hull, false);
    let solidity = area / hullArea;
    console.log(`Solidez: ${solidity}`)
}

// Diámetro equivalente
```

```
{
    let area = cv.contourArea(cnt, false);
    let equiDiameter = Math.sqrt(4 * area / Math.PI);
    console.log(`Diámetro equivalente: ${equiDiameter}`)
}

// Orientación
{
    let rotatedRect = cv.fitEllipse(cnt);
    let angle = rotatedRect.angle;
    console.log(`Orientación: ${angle}`)
}

// Valor Máximo, Valor Mínimo y sus ubicaciones
{
    let result = cv.minMaxLoc(src);
    let minVal = result.minVal;
    let maxVal = result.maxVal;
    let minLoc = result.minLoc;
    let maxLoc = result.maxLoc;
    console.log(`minVal: ${minVal}, maxVal: ${maxVal}`)
    console.log('minLoc:', minLoc)
    console.log('maxLoc', maxLoc)
}

// Color medio o intensidad media
let average = cv.mean(src);
console.log('average', average)

src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
```

4093a_propiedades.html

Resultados



canvasInput canvasOutput

En la consola del navegador:

- Relación de aspecto: 0.925
- Extensón: 0.7702702702702703
- Solidez: 0.9840310746655158
- Diámetro equivalente: 38.09846559899868
- Orientación: 1.392520785331726
- minVal: 0, maxVal: 200
- minLoc: {x: 0, y: 0}
- maxLoc {x: 107, y: 25}
- average (4) [38.13460642757336, 0, 0, 0]

4.9.4.9.- Máscara y puntos de pixel

En algunos casos, podemos necesitar la matriz transpuesta de una determinada imagen.

Utilizaremos la función:

cv.transpose (src, dst)

Parámetros:

src matriz de entrada

dst matriz de salida del mismo tipo que src

El siguiente códgo es una aplicación de la función **cv.transpose** citada anteriormente.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 120, 200, cv.THRESH_BINARY);
cv.transpose(src, dst);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4093b_imagen_transpuesta.html

Resultados

canvasInput canvasOutput

4.9.5.- Contornos: más funciones

Ojetivos

- Defectos de convexidad y como encontralos
- Encontrar la distancia más corta de un punto a un polígono
- Encontrar coincidencias entre diferentes formas

4.9.5.1.- Defectos de convexidad

Vimos lo que es un casco convexo en el segundo capítulo sobre contornos. Cualquier desviación del objeto de este casco se puede considerar como un defecto de convexidad. Podemos visualizarlo usando una imagen.

Las marcas de flechas de dos puntas muestran los defectos de convexidad, que son las desviaciones máximas locales del casco con respecto a los contornos.

Nota

Recuerde que tenemos que pasar **returnPoints = False** al encontrar un casco convexo, para encontrar defectos de convexidad.

Utilizamos la función:

cv.convexityDefects (contour, convexhull, convexityDefect)

Parámetros:

contour	contorno de entrada.
convexhull	casco convexo obtenido usando convexHull que debería contener índices de los puntos de contorno que forman el casco
convexityDefect	el vector de salida de los defectos de convexidad. Cada defecto de convexidad se representa con 4 elementos (start_index, end_index, farthest_pt_index, fixpt_ depth), donde los índices son índices basados en 0 en el contorno original donde el defecto de convexidad que comienza, end y farthest poin, y fixpt_ depth es una aproximación de punto fijo (con 8 bits fraccionarios) de la distancia entre el punto más lejano del contorno y el casco. Es decir, para obtener el valor de punto flotante de la profundidad será fixpt_depth/256.0.

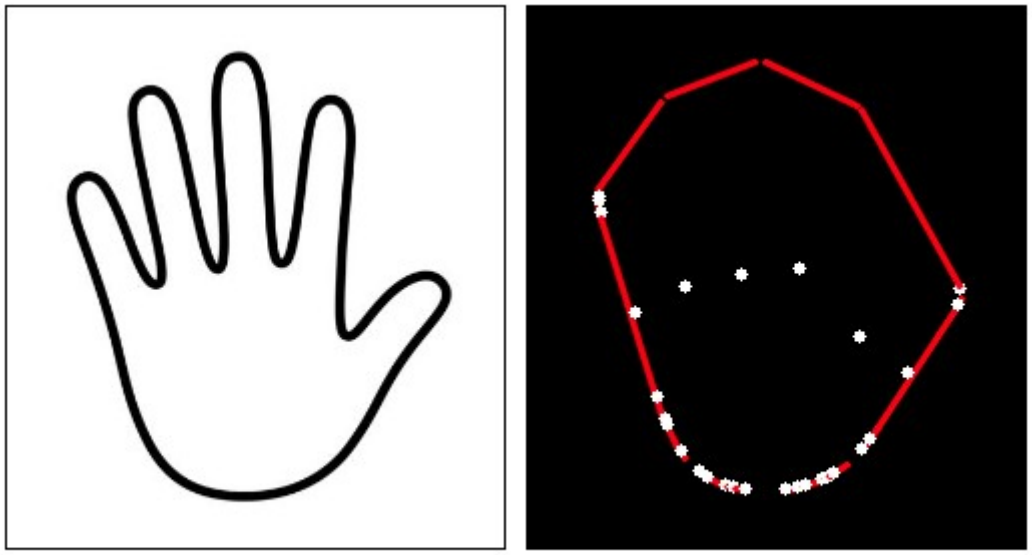
El siguiente código es una aplicación de la función **cv.convexityDefects**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 100, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let hull = new cv.Mat();
let defect = new cv.Mat();
```

```
let cnt = contours.get(0);
let lineColor = new cv.Scalar(255, 0, 0);
let circleColor = new cv.Scalar(255, 255, 255);
cv.convexHull(cnt, hull, false, false);
cv.convexityDefects(cnt, hull, defect);
for (let i = 0; i < defect.rows; ++i) {
    let start = new cv.Point(cnt.data32S[defect.data32S[i * 4] * 2],
                             cnt.data32S[defect.data32S[i * 4] * 2 + 1]);
    let end = new cv.Point(cnt.data32S[defect.data32S[i * 4 + 1] * 2],
                           cnt.data32S[defect.data32S[i * 4 + 1] * 2 + 1]);
    let far = new cv.Point(cnt.data32S[defect.data32S[i * 4 + 2] * 2],
                           cnt.data32S[defect.data32S[i * 4 + 2] * 2 + 1]);
    cv.line(dst, start, end, lineColor, 2, cv.LINE_AA, 0);
    cv.circle(dst, far, 3, circleColor, -1);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); hierarchy.delete(); contours.delete();
hull.delete(); defect.delete();
```

4094a_convexity_defects.html

Resultados



canvasInput canvasOutput

4.9.5.2.- Point Polygon Test

Esta función encuentra la distancia más corta entre un punto en la imagen y un contorno. Devuelve la distancia que es negativa cuando el punto está fuera del contorno, positiva cuando el punto está dentro y cero si el punto está en el contorno.

Utilizaremos la función:

cv.pointPolygonTest (contour, pt, measureDist)

Parámetros:

contour	contorno de entrada.
pt	punto probado contra el contorno.
measureDist	si es verdadero, la función estima la distancia con signo desde el punto hasta el borde del contorno más cercano. De lo contrario, la función solo comprueba si el punto está dentro de un contorno o no.

Por ejemplo:

```
let dist = cv.pointPolygonTest(cnt, new cv.Point(50, 50), true);
```

El siguiente códgo muestra como calcular la distancia de un punto a un contorno.

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.cols, src.rows, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 120, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
cv.drawContours(dst, contours, 1, new cv.Scalar(255, 0, 0), 1, cv.LINE_8,
hierarchy, 0);
cv.imshow('canvasOutput', dst);

let cnt = contours.get(1);
let P = new cv.Point(100, 100)
let dist = cv.pointPolygonTest(cnt, P, true);
console.log(`Distancia: ${dist}`)   // Distancia: 13.038404810405298


src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
```

4094b_punto_poligono_test.html

Resultados

Observar la consola del navegador.

4.9.5.3.- Coincidencia de formas

OpenCV viene con una función **cv.matchShapes()** que nos permite comparar dos formas o dos contornos y devuelve una métrica que muestra la similitud. Cuanto más bajo es el resultado, mejor coincidencia es. Se calcula en base a los valores de momento hu. Los diferentes métodos de medición se explican en un enlace más abajo.

Utilizamos la función:

cv.matchShapes (contour1, contour2, method, parameter)

Parámetros:

contour1 primer ontorno o imagen en escala de grises.

contour2 segundo ontorno o imagen en escala de grises.

method método de comparación, consulte cv::ShapeMatchModes

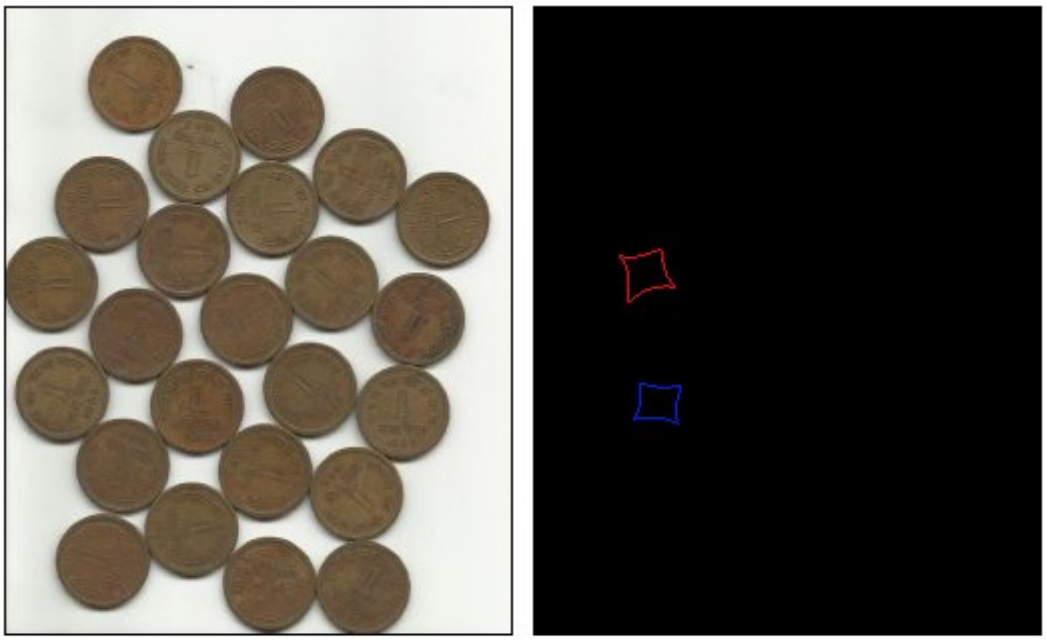
parameter parámetro específico del método (no soportado ahora).

Las siguientes líneas de código muestran una aplicación de la función **cv.matchShapes**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(src, src, 177, 200, cv.THRESH_BINARY);
let contours = new cv.MatVector();
let hierarchy = new cv.Mat();
cv.findContours(src, contours, hierarchy, cv.RETR_CCOMP,
cv.CHAIN_APPROX_SIMPLE);
let contourID0 = 10;
let contourID1 = 5;
let color0 = new cv.Scalar(255, 0, 0);
let color1 = new cv.Scalar(0, 0, 255);
// You can try more different parameters
let result = cv.matchShapes(contours.get(contourID0),
contours.get(contourID1), 1, 0);
matchShapesOutput.innerHTML = result;
cv.drawContours(dst, contours, contourID0, color0, 1, cv.LINE_8, hierarchy,
100);
cv.drawContours(dst, contours, contourID1, color1, 1, cv.LINE_8, hierarchy,
100);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); contours.delete(); hierarchy.delete();
```

4094c_match_shape.html

Resultados



Los contornos elegidos son huecos entre monedas.

4.9.6.- Jerarquía de contornos

4.9.6.1.- Objetivos

Esta vez, aprendemos sobre la jerarquía de contornos, es decir, la relación padre-hijo en Contornos.

4.9.6.2.- Teoría

En los últimos artículos sobre contornos, hemos trabajado con varias funciones relacionadas con los contornos proporcionadas por OpenCV. Pero cuando encontramos los contornos en la imagen usando la función **cv.findContours()**, pasamos un argumento **Contour Retrieval Mode** (Modo de recuperación de contorno). Por lo general, pasamos **cv.RETR_LIST** o **cv.RETR_TREE** y funcionó bien. Pero, ¿qué significa realmente?

Además, en la salida, obtuvimos tres matrices, la primera es la imagen, la segunda son nuestros contornos y una salida más que llamamos **hierarchy** (jerarquía). Puede consultar los códigos en apartados anteriores. Pero nunca usamos esta jerarquía en ninguna parte. Entonces, ¿qué es esta jerarquía y para qué sirve? ¿Cuál es su relación con el argumento de función mencionado anteriormente?

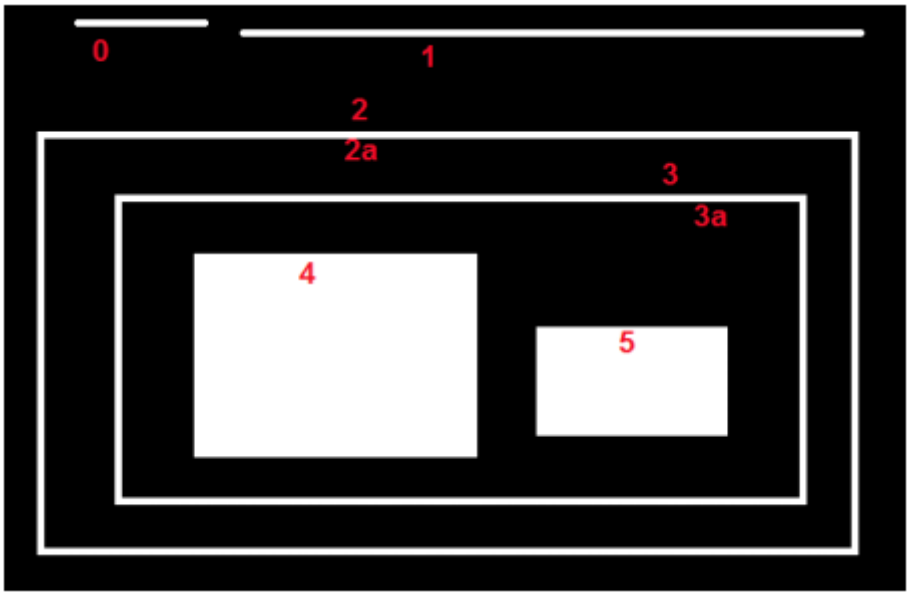
Eso es lo que vamos a tratar en este artículo.

4.9.6.2.1.- ¿Qué es Jerarquía?

Normalmente usamos la función cv.findContours() para detectar objetos en una imagen, ¿verdad? A veces los objetos están en diferentes lugares. Pero en algunos casos, algunas formas están dentro de

otras formas. Al igual que las figuras anidadas. En este caso, llamamos al exterior como padre y al interior como hijo. De esta manera, los contornos de una imagen tienen alguna relación entre sí. Y podemos especificar cómo un contorno está conectado entre sí, por ejemplo, si es hijo de otro contorno, o es un padre, etc. La representación de esta relación se llama Jerarquía.

Considere la imagen de ejemplo a continuación:



En esta imagen, hay algunas formas que hemos numerado del 0 al 5. *2* y *2a* denota los contornos externo e interno de la caja más exterior.

Aquí, los contornos 0,1,2 son externos o exteriores. Podemos decir que están en jerarquía-0 o simplemente están en el mismo nivel de jerarquía.

Luego viene el contorno-2a. Se puede considerar como un hijo del contorno-2 (o de manera opuesta, el contorno-2 es el padre del contorno-2a). Así que está en la jerarquía-1. De manera similar, el contorno-3 es hijo del contorno-2a y viene en la siguiente jerarquía. Finalmente los contornos 4,5 son los hijos del contorno-3a, y vienen en el último nivel de jerarquía. Por la forma en que numeramos los cuadros, diría que el contorno-4 es el primer hijo del contorno-3a (también puede ser el contorno-5).

Mencionamos estas cosas para comprender términos como el mismo nivel de jerarquía, contorno externo, contorno secundario, contorno principal, primer hijo, etc. Ahora entremos en OpenCV.

4.9.6.2.2.- Representación de jerarquía en OpenCV

Entonces, cada contorno tiene su propia información sobre qué jerarquía es, quién es su hijo, quién es su padre, etc. OpenCV lo representa como una matriz de cuatro valores: [Next, Previous, First_Child, Parent]

"Next denota el siguiente contorno en el mismo nivel jerárquico".

Por ejemplo, tome el contorno-0 en nuestra imagen. ¿Quién es el próximo contorno en su mismo nivel? Es contorno-1. Así que simplemente ponga Next = 1. De manera similar para contorno-1, el siguiente es contorno-2. Entonces Next = 2.

¿Qué pasa con el contorno-2? No hay contorno siguiente en el mismo nivel. Simplemente, ponga Next = -1. ¿Qué pasa con el contorno-4? Está en el mismo nivel que el contorno-5. Entonces su siguiente contorno es contorno-5, entonces Next = 5.

"Previous denota el contorno anterior en el mismo nivel jerárquico".

Es lo mismo que arriba. El contorno anterior del contorno-1 es el contorno-0 en el mismo nivel. De manera similar para el contorno-2, es el contorno-1. Y para el contorno-0, no hay anterior, así que póngalo como -1.

"First_Child denota su primer contorno hijo".

No hay necesidad de ninguna explicación. Para contorno-2, el niño es contorno-2a. Entonces obtiene el valor de índice correspondiente de contorno-2a. ¿Qué pasa con el contorno-3a? Tiene dos hijos. Pero tomo solamente al primer niño. Y es contorno-4. Entonces First_Child = 4 para contorno-3a.

"Parent denota el índice de su contorno padre".

Es justo lo contrario de First_Child. Tanto para el contorno-4 como para el contorno-5, el contorno Parent es el contorno-3a. Para el contorno-3a, es el contorno-3 y así sucesivamente.

Nota

Si no hay hijo o padre, ese campo se toma como -1

Entonces, ahora que conocemos el estilo de jerarquía utilizado en OpenCV, podemos verificar los modos de recuperación de contorno en OpenCV con la ayuda de la misma imagen que se muestra arriba. es decir, ¿qué significan indicadores como cv.RETR_LIST, cv.RETR_TREE, cv.RETR_CCOMP, cv.RETR_EXTERNAL, etc.?

4.9.6.3.- Modo de recuperación de contorno

4.9.6.3.1.- RETR_LIST

Esta es la más simple de las cuatro banderas (desde el punto de vista de la explicación). Simplemente recupera todos los contornos, pero no crea ninguna relación padre-hijo. Padres e hijos son iguales bajo esta regla, y son solo contornos. es decir, todos pertenecen al mismo nivel de jerarquía.

Entonces, aquí, el tercer y cuarto término en la matriz de jerarquía siempre es -1. Pero obviamente, los términos Next y Previous tendrán sus valores correspondientes.

4.9.6.3.2.- RETR_EXTERNAL

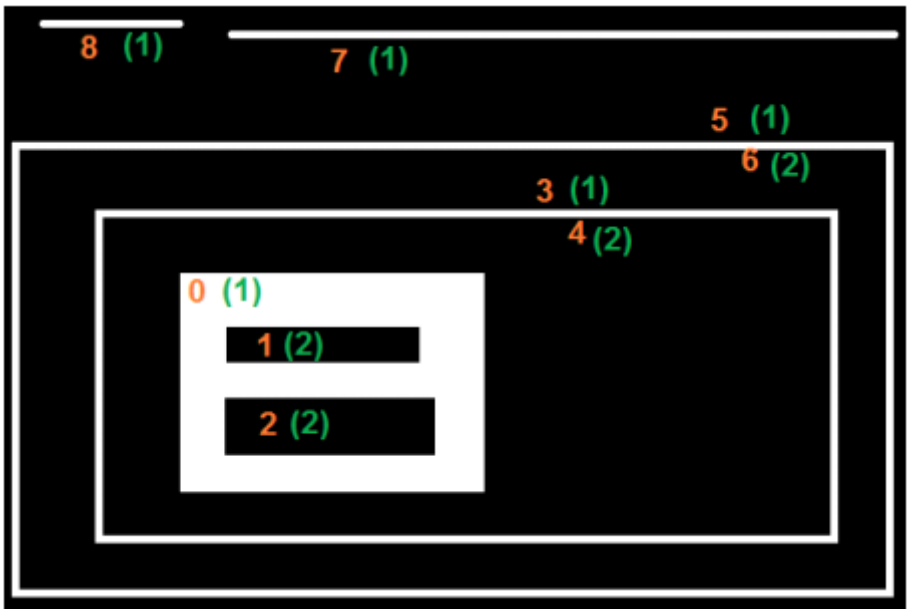
Si usa esta bandera, solo devuelve los contornos externos. Todos los contornos secundarios se dejan atrás. Podemos decir, bajo esta ley, que sólo se cuida al mayor de cada familia. No le importan los demás miembros de la familia.

4.9.6.3.3.- RETR_CCOMP

Esta bandera recupera todos los contornos y los organiza en una jerarquía de 2 niveles. es decir, los contornos externos del objeto (es decir, su límite) se colocan en la jerarquía-1. Y los contornos de los agujeros dentro del objeto (si los hay) se colocan en la jerarquía-2. Si hay algún objeto dentro de él, su contorno se coloca nuevamente en la jerarquía-1 y su agujero, si existe, en la jerarquía-2 y así sucesivamente.

Solo considere la imagen de un "gran cero blanco" sobre un fondo negro. El círculo exterior del cero pertenece a la primera jerarquía y el círculo interior de cero pertenece a la segunda jerarquía.

Podemos explicarlo con una simple imagen. Aquí hemos etiquetado el orden de los contornos en color rojo y la jerarquía a la que pertenecen, en color verde (ya sea 1 o 2). El orden es el mismo que el orden en que OpenCV detecta los contornos.



Así que considere el primer contorno, es decir, el contorno-0. Es jerarquía-1. Tiene dos agujeros, contornos 1 y 2, y pertenecen a la jerarquía-2. Entonces, para el contorno-0, el siguiente contorno en el mismo nivel de jerarquía es el contorno-3. Y no hay anterior. Y su primer hijo es contorno-1 en jerarquía-2. No tiene padre, porque está en la jerarquía-1. Entonces su matriz de jerarquía es [3,-1,1,-1]

Ahora toma el contorno-1. Está en la jerarquía-2. El siguiente en la misma jerarquía (bajo la paternidad de contorno-1) es contorno-2. Ninguno anterior. Ningún hijo, pero el padre es contorno-0. Entonces la matriz es [2,-1,-1,0].

Del mismo modo contorno-2: Está en la jerarquía-2. No existe el siguiente contorno en la misma jerarquía bajo el contorno-0. Así que no tiene Siguiente. El anterior es contorno-1. Ningún hijo, el padre es contorno-0. Entonces la matriz es [-1,1,-1,0].

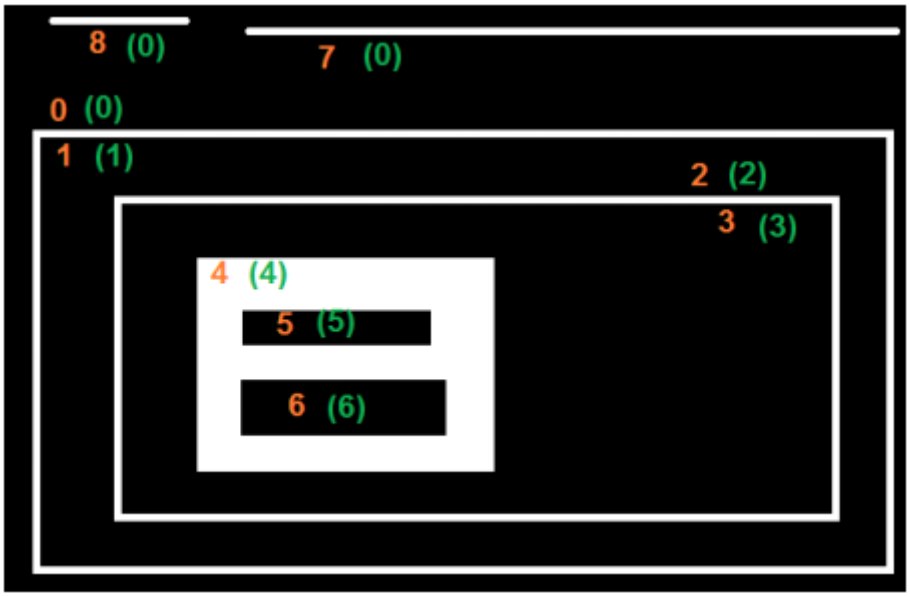
Contorno-3: El siguiente en la jerarquía-1 es el contorno-5. El anterior es contorno-0. El niño es contorno-4 y no tiene padre. Entonces la matriz es [5,0,4,-1].

Contorno - 4: Está en la jerarquía 2 bajo contorno-3 y no tiene hermanos. Entonces, no hay siguiente, no hay anterior, no hay hijo, el padre es contorno-3. Entonces la matriz es [-1,-1,-1,3].

4.9.6.3.4.- RETR_TREE

Y este es el chico final, Mr.Perfect. Recupera todos los contornos y crea una lista de jerarquía familiar completa. Incluso dice quién es el abuelo, el padre, el hijo, el nieto e incluso más allá... :).

Por ejemplo, tomemos la imagen de arriba, reescriba el código para cv.RETR_TREE, reordenemos los contornos según el resultado dado por OpenCV y lo analicémoslo. Nuevamente, las letras rojas dan el número de contorno y las letras verdes dan el orden jerárquico.



Tome el contorno-0: Está en jerarquía-0. El siguiente contorno en la misma jerarquía es el contorno-7. Sin contornos previos. El hijo es contorno-1. Y sin padre. Entonces la matriz es [7,-1,1,-1].

Tomar contorno-1: Está en jerarquía-1. Sin contorno en el mismo nivel. Ninguna anterior. El niño es contorno-2. El padre es contorno-0. Entonces la matriz es [-1,-1,2,0].

4.10.- Histogramas en OpenCV.js

Aprenderemos sobre los histogramas en OpenCV.js.

4.10.1.- Histogramas - 1 : Encontrar, dibujar y analizar

Aprenderemos los conceptos básicos de los histogramas

4.10.1.1.- Objetivos

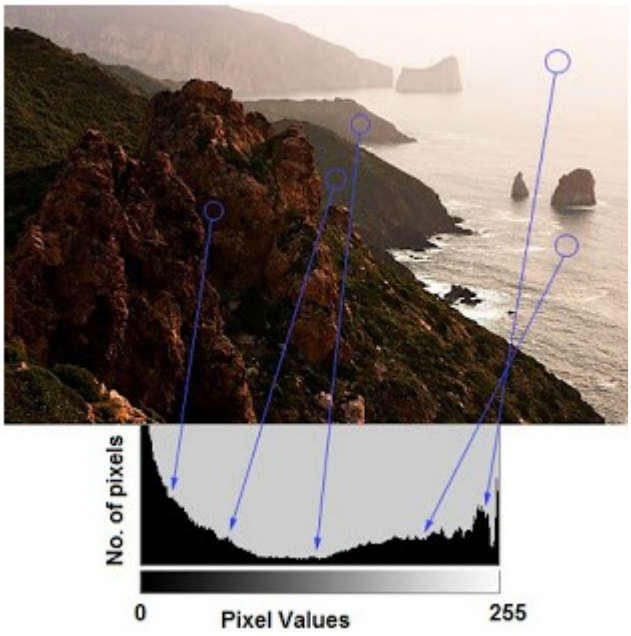
- Encontrar histogramas
- Trazar histogramas
- Aprender la función: **cv.calcHist()**

4.10.1.2.- Teoría

Entonces, ¿qué es el histograma? Puede considerar el histograma como un gráfico o diagrama que le da una idea general sobre la distribución de intensidad de una imagen. Es un gráfico con valores de píxeles (que van de 0 a 255, no siempre) en el eje X y el número correspondiente de píxeles en la imagen para cada x, representado en el eje Y.

Es sólo otra forma de entender la imagen. Al mirar el histograma de una imagen, obtiene intuición sobre el contraste, el brillo, la distribución de intensidad, etc. de esa imagen. Casi todas las herramientas de procesamiento de imágenes de hoy en día ofrecen funciones de histograma. A

continuación se muestra una imagen del sitio web de Cambridge en Color, y le recomiendamos que visite el sitio para obtener más detalles.



Puede ver la imagen y su histograma. (Recuerde, este histograma se dibuja para una imagen en escala de grises, no para una imagen en color). La región izquierda del histograma muestra la cantidad de píxeles más oscuros en la imagen y la región derecha muestra la cantidad de píxeles más brillantes. En el histograma, puede ver que la región oscura es mayor que la región más brillante, y la cantidad de tonos medios (valores de píxeles en el rango medio, digamos alrededor de 127) es muy inferior.

4.10.1.3.- Buscar el histograma

Utilizaremos la función:

cv.calcHist (image, channels, mask, hist, histSize, ranges, accumulate = false)

Parámetros:

image arreglo de imágenes fuente. Todos deben tener la misma profundidad, cv.CV_8U, cv.CV_16U o cv.CV_32F, y el mismo tamaño. Cada uno de ellos puede tener un número arbitrario de canales.

channels lista de los dims canales utilizados para calcular el histograma. Los canales de la primera matriz se numeran de 0 a images[0].channels()-1 , los canales de la

	segunda matriz se cuentan de images[0].channels() a images[0].channels() + images[1].channels ()-1, y así sucesivamente.
mask	máscara opcional. Si la matriz **no está vacía**, debe ser una matriz de 8 bits del mismo tamaño que images[i] . Los elementos de máscara distintos de cero marcan los elementos de matriz contados en el histograma.
hist	histograma de salida (tipo cv.CV_32F), que es una matriz densas o dispersas.
histSize	array de tamaños de histograma en cada dimensión. N.º elementos en el eje X.
ranges	array de los dims arrays de los límites de bin del histograma en cada dimensión. [valor min, valor max].
accumulate	bandera de acumulación. Si está configurada, el histograma no se borra al principio cuando se asigna. Esta función le permite calcular un único histograma a partir de varios conjuntos de matrices o actualizar el histograma a tiempo.

Para seguir con el código hay que recordar que en OpenCV el origen de coordenadas se sitúa en el extremos superior izquierdo de la imagen.

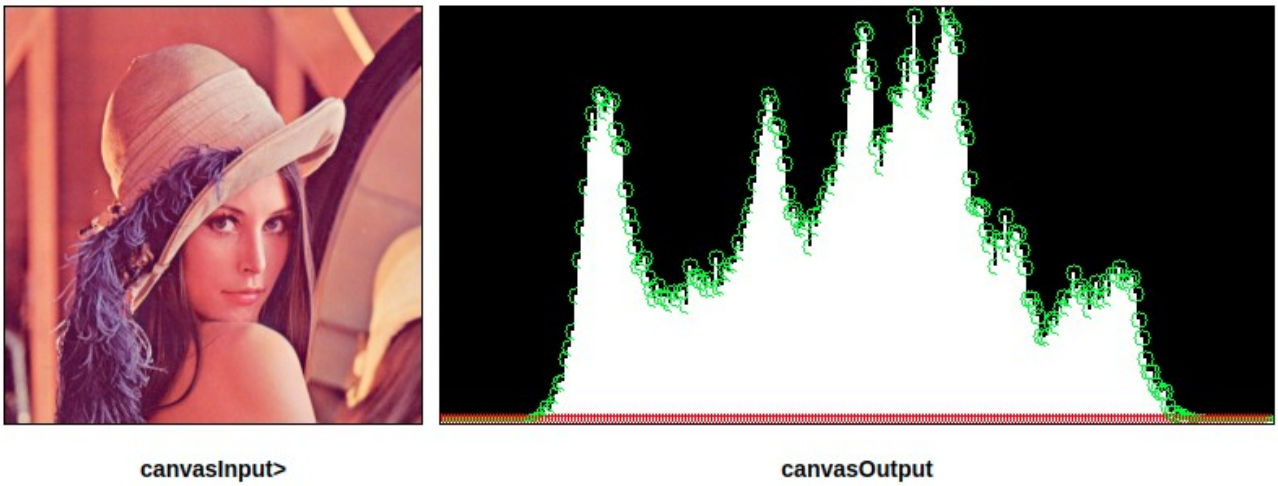
El liguiente programa calcula y muestra el histograma de una imagen.

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
let srcVec = new cv.MatVector();
srcVec.push_back(src);
let accumulate = false;
let channels = [0];
let histSize = [256];   // es un array luego, histSize[0] = 256
let ranges = [0, 255];
let hist = new cv.Mat();
let mask = new cv.Mat();
let color = new cv.Scalar(255, 255, 255);
let scale = 2;

// Puede probar con diferentes parámetros
cv.calcHist(srcVec, channels, mask, hist, histSize, ranges, accumulate);
let result = cv.minMaxLoc(hist, mask);
let max = result.maxVal;
let dst = new cv.Mat.zeros(src.rows, histSize[0] * scale, cv.CV_8UC3);
console.log(histSize[0])    // 256
// dibujar histograma
for (let i = 0; i < histSize[0]; i++) {
    // Para que la representación sea en altura como máximo el número  de
filas
    // de la matriz de entrada
    let binVal = hist.data32F[i] * src.rows / max;
    let point1 = new cv.Point(i * scale, src.rows - 1);
    let point2 = new cv.Point((i + 1) * scale - 1, src.rows - binVal);
    cv.rectangle(dst, point1, point2, color, cv.FILLED);
    // Pruebas mías (dibuja los putos rojos y verdes)
```

```
    cv.circle(dst, point1, 5, new cv.Scalar(255, 0, 0))
    cv.circle(dst, point2, 5, new cv.Scalar(000, 255, 0))
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); srcVec.delete(); mask.delete(); hist.delete();
```

4101_encontrar_dibujar_analizar.html

Resultados



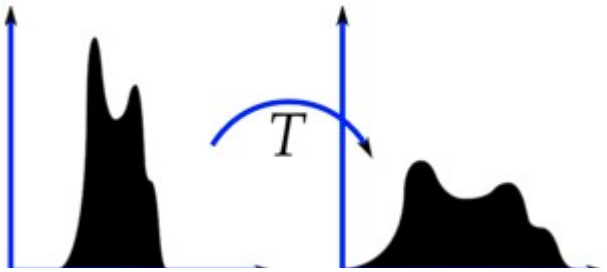
4.10.2.- Histogramas - 2 : Ecualización del histograma

4.10.2.1.- Objetivo

Aprenderemos los conceptos de ecualización de histogramas y lo utilizaremos para mejorar el contraste de nuestras imágenes.

4.10.2.2.- Teoría

Considere una imagen cuyos valores de píxel se limitan a un rango específico de valores únicamente. Por ejemplo, una imagen más brillante tendrá todos los píxeles confinados a valores altos. Pero una buena imagen tendrá píxeles de todas las regiones de la imagen. Por lo tanto, debe estirar este histograma hacia ambos extremos (como se muestra en la imagen a continuación, de wikipedia) y eso es lo que hace la ecualización de histograma (en palabras simples). Esto normalmente mejora el contraste de la imagen.



Le recomendamos que lea la página de wikipedia sobre ecualización de histogramas para obtener más detalles al respecto. Tiene una muy buena explicación con ejemplos elaborados, por lo que entenderá casi todo después de leer eso.

4.10.2.3.- Ecualización de histogramas en OpenCV

Utilizamos la siguiente función:

cv.equalizeHist (src, dst)

Parámetros:

src imagen fuente de un solo canal de 8 bits.

dst imagen de destino del mismo tamaño y tipo que src.

El siguiente código muestra como ecualizar un histograma.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.equalizeHist(src, dst);
cv.imshow('canvasInput', src);
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete();
```

4102_equalization.html

Resultados

canvasInput canvasOutput

4.10.2.4.- CLAHE (ecualización de histograma adaptativo limitado por contraste)

En la ecualización de histograma adaptable, la imagen se divide en pequeños bloques llamados "mosaicos" (el tamaño del mosaico es de 8x8 por defecto en OpenCV). Luego, para cada uno de estos bloques se evalúa el histograma como de costumbre. Entonces, en un área pequeña, el histograma se limitaría a una región pequeña (a menos que haya ruido). Si hay ruido, se amplificará. Para evitar esto, se aplica la limitación de contraste. Si algún bin de histograma está por encima del límite de contraste especificado (por defecto 40 en OpenCV), esos píxeles se recortan y distribuyen uniformemente a otros bins antes de aplicar la ecualización de histograma. Después de la ecualización, para eliminar los artefactos en los bordes de los mosaicos, se aplica la interpolación bilineal.

Utilizaremos la función:

cv.CLAHE (clipLimit = 40, tileGridSize = new cv.Size(8, 8))

Parámetros:

clipLimit umbral para la limitación de contraste.

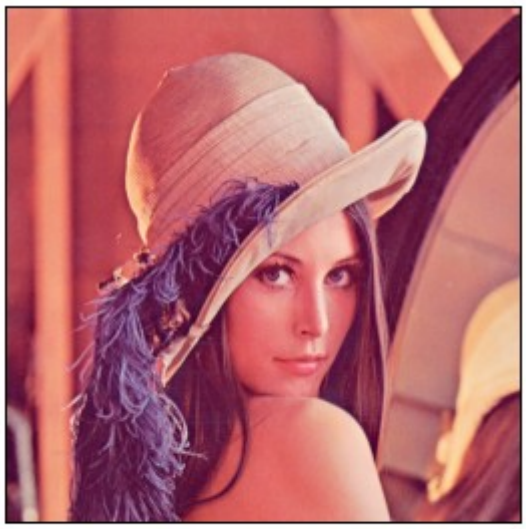
tileGridSize tamaño de la cuadrícula para la ecualización del histograma. La imagen de entrada se dividirá en mosaicos rectangulares de igual tamaño. tileGridSize define el tamaño de los mosaicos para dividir la imagen en filas y columnas.

El siguiente código muestra la utilización de CLAHE.

```
let src = cv.imread('canvasInput');
let equalDst = new cv.Mat();
let claheDst = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.equalizeHist(src, equalDst);
let tileGridSize = new cv.Size(8, 8);
// You can try more different parameters
let clahe = new cv.CLAHE(40, tileGridSize);
clahe.apply(src, claheDst);
cv.imshow('canvasOutput', equalDst);
cv.imshow('canvasOutput', claheDst);
src.delete(); equalDst.delete(); claheDst.delete(); clahe.delete();
```

4103_CLAHE.html

Resultados



canvasInput

canvasOutput

4.10.3.- Histogramas - 2 : Retroproyección de histograma

4.10.3.1.- Objetivos

- Aprenderemos la retroproyección de histogramas para segmentar objetos coloreados

4.10.3.2.- Teoría

Fue propuesto por Michael J. Swain, Dana H. Ballard en su artículo Indexación mediante histogramas de color.

¿Qué es realmente en palabras simples? Se utiliza para la segmentación de imágenes o para encontrar objetos de interés en una imagen. En palabras simples, crea una imagen del mismo tamaño (pero de un solo canal) que la de la imagen de entrada, donde cada píxel corresponde a la probabilidad de que ese píxel pertenezca a nuestro objeto. En mundos más simples, la imagen de salida tendrá nuestro objeto de interés más blanco en comparación con la parte restante. Bueno, esa es una explicación intuitiva. (No podemos hacerlo más simple). La retroproyección de histograma se utiliza con el algoritmo camshift, etc.

¿Cómo lo hacemos ? Creamos un histograma de una imagen que contenga nuestro objeto de interés. El objeto debe llenar la imagen tanto como sea posible para obtener mejores resultados. Y se prefiere un histograma de color sobre un histograma en escala de grises, porque el color del objeto es una mejor manera de definir el objeto que su intensidad en escala de grises. Luego, "reproyectamos" este histograma sobre nuestra imagen de entrada donde necesitamos encontrar el objeto, es decir, en otras palabras, calculamos la probabilidad de que cada píxel pertenezca al suelo y lo mostramos. La salida resultante en el umbral adecuado nos da el objeto de interés.

4.10.3.3.- Retroproyección en OpenCV

Usamos la función:

cv.calcBackProject (images, channels, hist, dst, ranges, scale)

Parámetros:

images	array de fuentes. Todos deben tener la misma profundidad, cv.CV_8U, cv.CV_16U o cv.CV_32F, y el mismo tamaño. Cada uno de ellos puede tener un número arbitrario de canales.
chanels	la lista de canales utilizados para calcular la retroproyección. El número de canales debe coincidir con la dimensionalidad del histograma.
hist	histograma de entrada que puede ser denso o disperso.
dst	matriz de retroproyección de destino que es una matriz de un solo canal del mismo tamaño y profundidad que images[0].
ranges	matriz de matrices de los límites de bin del histograma en cada dimensión (ver cv.calcHist).
scale	factor de escala opcional para la retroproyección de salida.

La otra función para normalizar el histograma:

cv.normalize (src, dst, alpha = 1, beta = 0,

norm_type = cv.NORM_L2, dtype = -1, mask = new cv.Mat())

Parámetros:

src	matriz de entrada
dst	matriz de salida del mismo tamaño que src
alpha	valor de norma para normalizar o el límite inferior del rango en el caso de la normalización del rango.
beta	límite superior del rango en el caso de la normalización del rango; no se utiliza para la normalización de normas.
norm_type	tipo de normalización (ver cv.NormTypes).
dtype	cuando es negativo, la matriz de salida tiene el mismo tipo que src; de lo contrario, tiene el mismo número de canales que src y la profundidad = CV_MAT_DEPTH (dtype)
mask	máscara de operación opcional.

El siguiente código muestra como calcular la retroproyección del histograma.

```
let src = cv.imread('srcCanvasInput');
let dst = cv.imread('dstCanvasInput');
cv.cvtColor(src, src, cv.COLOR_RGB2HSV, 0);
cv.cvtColor(dst, dst, cv.COLOR_RGB2HSV, 0);
let srcVec = new cv.MatVector();
let dstVec = new cv.MatVector();
srcVec.push_back(src);
dstVec.push_back(dst);
let backproj = new cv.Mat();
let none = new cv.Mat();
let mask = new cv.Mat();
let hist = new cv.Mat();
let channels = [0];
let histSize = [50];
let ranges = [0, 180];
let accumulate = false;
cv.calcHist(srcVec, channels, mask, hist, histSize, ranges, accumulate);
cv.normalize(hist, hist, 0, 255, cv.NORM_MINMAX, -1, none);
cv.calcBackProject(dstVec, channels, hist, backproj, ranges, 1);
cv.imshow('canvasOutput', backproj);
src.delete(); dst.delete(); srcVec.delete(); dstVec.delete();
backproj.delete(); mask.delete(); hist.delete(); none.delete();
```

4104_retroproyeccion.html

Resultados

srcCanvasInput dstCanvasInput canvasOutput

4.11.- Transformada de la imagen en OpenCV.js

No veremos nada en este apartado.

4.12.- Coincidencia de plantilla

4.12.1.- Objetivos

- Buscar objetos en una imagen mediante Coincidencia de plantillas
- Aprenderá estas funciones: **cv.matchTemplate()**, **cv.minMaxLoc()**

4.12.2.- Teoría

Coincidencia de plantillas es un método para buscar y encontrar la ubicación de una imagen de plantilla en una imagen más grande. OpenCV viene con una función cv.matchTemplate() para este propósito. Simplemente desliza la imagen de la plantilla sobre la imagen de entrada (como en la convolución 2D) y compara la plantilla y el parche de la imagen de entrada debajo de la imagen de la plantilla. Varios métodos de comparación están implementados en OpenCV. (Puede consultar los documentos para obtener más detalles). Devuelve una imagen en escala de grises, donde cada píxel indica cuánto coincide la vecindad de ese píxel con la plantilla.

Si la imagen de entrada tiene un tamaño (W x H) y la imagen de la plantilla tiene un tamaño (w x h), la imagen de salida tendrá un tamaño de (W-w + 1, H-h +1). Una vez que obtuvo el resultado, puede usar la función **cv.minMaxLoc()** para encontrar dónde está el máximo/mínimo valor. Tómelo como la esquina superior izquierda del rectángulo y tome (w,h) como ancho y alto del rectángulo. Ese rectángulo es su región de plantilla.

Si utiliza cv.TM_SQDIFF como método de comparación, el valor mínimo proporciona la mejor coincidencia.

4.12.3.- Coincidencia de plantillas en OpenCV

Utilizaremos la función:

cv.matchTemplate (image, templ, result, method,

mask = new cv.Mat())

Parámetros:

image imagen donde se ejecuta la búsqueda. Debe ser de 8 bits o de 32 bits en coma flotante.

templ plantilla buscada. No debe ser mayor que la imagen de origen y tener el mismo tipo de datos.

result mapa de resultados de comparación. Debe ser de punto flotante de 32 bits de un solo canal.

method parámetro que especifica el método de comparación (ver cv.TemplateMatchModes).

mask máscara de la plantilla buscada. Debe tener el mismo tipo de datos y tamaño que templ. No está configurada de forma predeterminada.

El siguiente código muestra una aplicación de búsqueda y coincidencia de plantilla.

```
let src = cv.imread('imageCanvasInput');
let templ = cv.imread('templateCanvasInput');
let dst = new cv.Mat();
let mask = new cv.Mat();
cv.matchTemplate(src, templ, dst, cv.TM_CCOEFF, mask);
let result = cv.minMaxLoc(dst, mask);
let maxPoint = result.maxLoc;
let color = new cv.Scalar(255, 0, 0, 255);
let point = new cv.Point(maxPoint.x + templ.cols, maxPoint.y + templ.rows);
cv.rectangle(src, maxPoint, point, color, 2, cv.LINE_8, 0);
cv.imshow('canvasOutput', src);
src.delete(); dst.delete(); mask.delete();
```

4121_coincidencia_plantilla.html

Resultados

imageCanvasInput

canvasOutput

templateCanvasInput

4.13.- Transormada de linea Hough

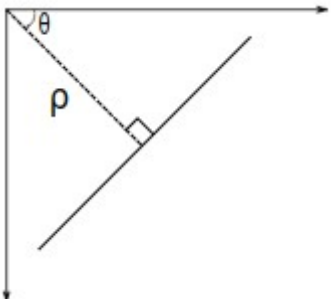
4.13.1.- Objetivos

- Entender el concepto de la Transformada de Hough.
- Aprenderemos a usarlo para detectar líneas en una imagen.
- Aprenderemos las siguientes funciones: **cv.HoughLines()**, **cv.HoughLinesP()**

4.13.2.- Teoría

La transformada de Hough es una técnica popular para detectar cualquier forma, si puede representar esa forma en una ecuación matemática. Puede detectar la forma incluso si está rota o distorsionada un poco. Veremos cómo funciona para una línea.

Una línea se puede representar como y=mx + c o en forma paramétrica, como ρ=x*cosθ + y*sinθ donde ρ es la distancia perpendicular desde el origen a la línea, y θ es el ángulo formado por esta línea perpendicular y el eje horizontal medido en en sentido contrario a las agujas del reloj (esa dirección varía según cómo represente el sistema de coordenadas. Esta representación se usa en OpenCV). Revise la imagen a continuación:

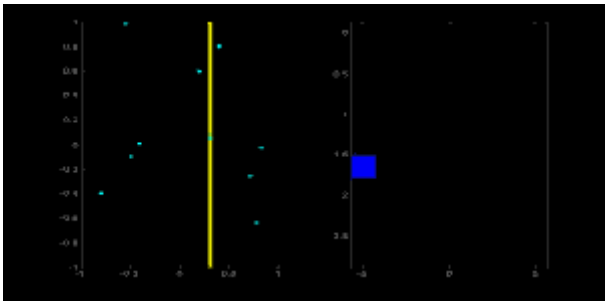


Entonces, si la recta pasa por debajo del origen, tendrá rho positivo y un ángulo menor de 180. Si pasa por encima del origen, en lugar de tomar un ángulo mayor de 180, se toma el ángulo menor de 180, y rho se toma negativo. Cualquier línea vertical tendrá 0 grados y las líneas horizontales tendrán 90 grados.

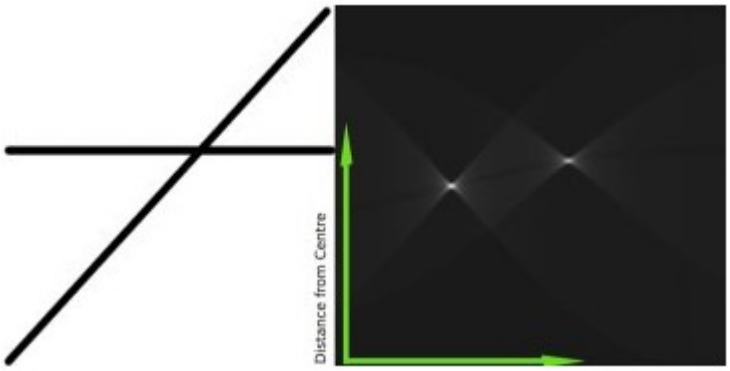
Ahora veamos cómo funciona la Transformada de Hough para las líneas. Cualquier línea se puede representar en estos dos términos, (ρ,θ). Entonces, primero crea una matriz o acumulador 2D (para contener los valores de los dos parámetros) y se establece en 0 inicialmente. Deje que las filas denoten ρ y las columnas denoten θ. El tamaño de la matriz depende de la precisión que necesite. Suponga que desea que la precisión de los ángulos sea de 1 grado, necesitará 180 columnas. Para ρ, la distancia máxima posible es la longitud diagonal de la imagen. Entonces, tomando una precisión de un píxel, el número de filas puede ser la longitud diagonal de la imagen.

Considere una imagen de 100x100 con una línea horizontal en el medio. Toma el primer punto de la línea. Conoces sus valores (x, y). Ahora en la ecuación de línea, pon los valores θ=0,1,2,....,180 y comprueba el ρ que obtienes. Para cada par (ρ, θ), incrementa el valor en uno en nuestro acumulador en sus celdas correspondientes (ρ, θ). Así que ahora en el acumulador, la celda (50,90) = 1 junto con algunas otras celdas.

Ahora toma el segundo punto de la línea. Haz lo mismo que arriba. Incrementa los valores en las celdas correspondientes a (ρ,θ) que obtuviste. Esta vez, la celda (50,90) = 2. Lo que realmente haces es votar los valores (ρ, θ). Continúa este proceso para cada punto de la línea. En cada punto, la celda (50,90) se incrementará o votará a favor, mientras que otras celdas pueden o no votarse a favor. Así, al final, la celda (50,90) tendrá el máximo de votos. Entonces, si busca en el acumulador el máximo de votos, obtiene el valor (50,90) que dice que hay una línea en esta imagen a una distancia de 50 desde el origen y en un ángulo de 90 grados. Se muestra bien en la siguiente animación (Imagen cortesía: Amos Storkey)



Así es como funciona la transformación Hough para las líneas. Es simple. A continuación se muestra una imagen que muestra el acumulador. Los puntos brillantes en algunos lugares indican que son los parámetros de posibles líneas en la imagen. (Imagen cortesía: Wikipedia)



4.13.3.- Hough Transform in OpenCV

Todo lo explicado anteriormente está encapsulado en la función OpenCV, **cv.HoughLines().** Simplemente devuelve una matriz de valores ((ρ,θ). ρ se mide en píxeles y θ se mide en radianes. El primer parámetro, la imagen de entrada debe ser una imagen binaria, por lo tanto, aplique el umbral o use la detección de bordes Canny antes de aplicar la transformación Hough.

Utilizamos la función:

cv.HoughLines (image, lines, rho, theta, threshold, srn = 0,

stn = 0, min_theta = 0, max_theta = Math.PI)

image	Imagen de fuente binaria de un solo canal de 8 bits. La imagen puede ser modificada por la función.
lines	vector de salida de líneas (tipo cv.32FC2). Cada línea está representada por un vector de dos elementos (ρ,θ) . ρ es la distancia desde el origen de coordenadas (0,0). θ es el ángulo de rotación de la línea en radianes.
rho	resolución de distancia del acumulador en píxeles.
theta	resolución angular del acumulador en radianes
threshold	parámetro de umbral del acumulador. Solo se devuelven aquellas líneas que obtienen suficientes votos.
srn	para la transformada de Hough multiescala, es un divisor de la resolución de distancia rho. La resolución aproximada de la distancia del acumulador es rho y la resolución exacta del acumulador es rho/srm. Si tanto srn=0 como stn=0 , se usa la transformada clásica de Hough. De lo contrario, ambos parámetros deberían ser

positivos.

stn para la transformada de Hough multiescala, es un divisor de la resolución de distancia theta.

min_theta para transformada de Hough estándar y multiescala, ángulo mínimo para comprobar si hay líneas. Debe estar entre 0 y max_theta.

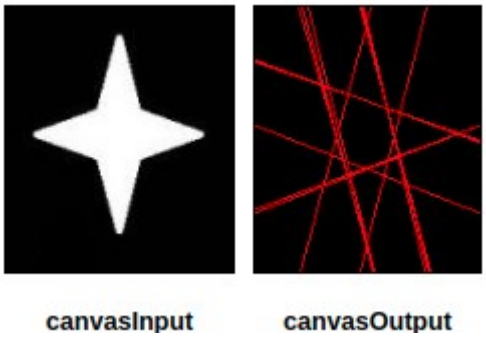
max_theta para transformada de Hough estándar y multiescala, ángulo máximo para comprobar si hay líneas. Debe estar entre min_theta y CV_PI.

El siguiente código muestra una aplicación de la función **cv.HoughLines**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
let lines = new cv.Mat();
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.Canny(src, src, 50, 200, 3);
// Puede probar con fiferentes parámetrso
cv.HoughLines(src, lines, 1, Math.PI / 180,
            30, 0, 0, 0, Math.PI);
// draw lines
for (let i = 0; i < lines.rows; ++i) {
    let rho = lines.data32F[i * 2];
    let theta = lines.data32F[i * 2 + 1];
    let a = Math.cos(theta);
    let b = Math.sin(theta);
    let x0 = a * rho;
    let y0 = b * rho;
    let startPoint = {x: x0 - 1000 * b, y: y0 + 1000 * a};
    let endPoint = {x: x0 + 1000 * b, y: y0 - 1000 * a};
    cv.line(dst, startPoint, endPoint, [255, 0, 0, 255]);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); lines.delete();
```

4131_hough_lines.html

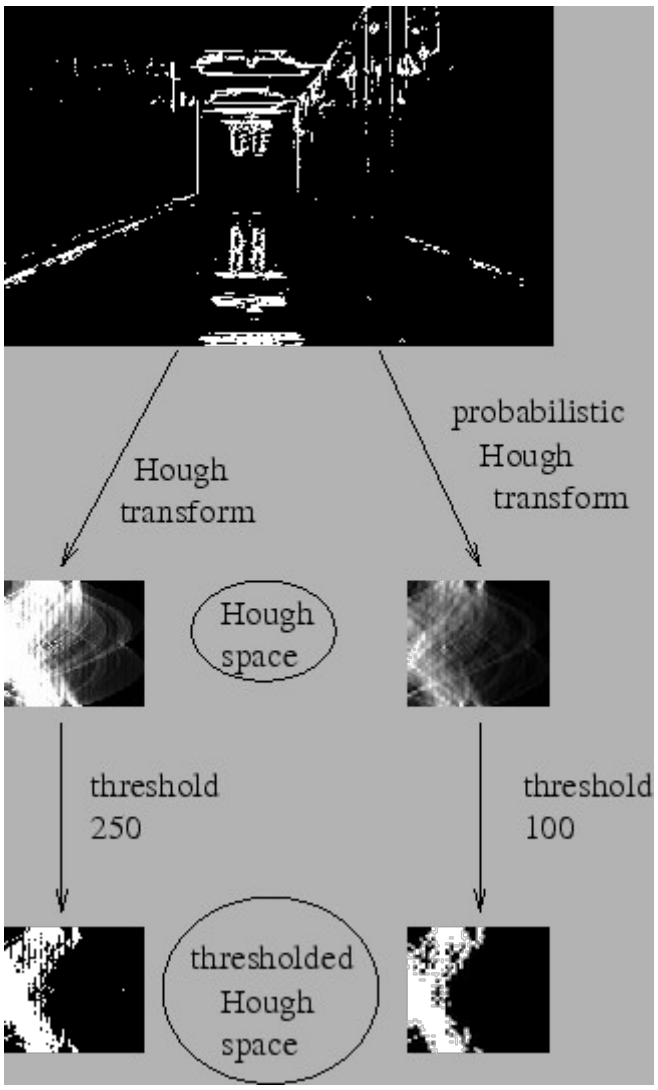
Resultados



4.13.4.- Transformada probabilística de Hough

En la transformada de Hough, puede ver que incluso para una línea con dos argumentos, se necesitan muchos cálculos. Probabilistic Hough Transform es una optimización de la Hough Transform que vimos. No tiene en cuenta todos los puntos. En cambio, solo toma un subconjunto aleatorio de puntos que es suficiente para la detección de líneas. Solo tenemos que disminuir el umbral. Vea la imagen a continuación que compara la Transformada de Hough y la Transformada de Hough probabilística en el espacio de Hough.

(Image Courtesy :Franck Bettinger's home page)



La implementación de OpenCV se basa en la detección robusta de líneas usando la transformada de Hough probabilística progresiva de Matas, J. y Galambos, C. y Kittler, J.V. [149].

Utilizamos la función:

cv.HoughLinesP (image, lines, rho, theta, threshold,

minLineLength = 0, maxLineGap = 0)

Parametros:

image Imagen de fuente binaria de un solo canal de 8 bits. La imagen puede ser modificada por la función.

lines vector de salida de líneas (tipo cv.32SC4). Cada línea está representada por un vector de 4 elementos (x1,y1,x2,y2), donde (x1,y1) y (x2,y2) son los

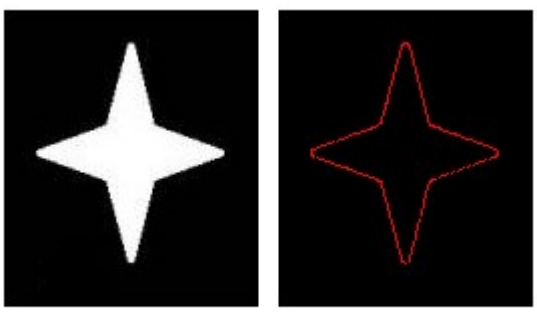
	puntos finales de cada segmento de línea detectado.
rho	resolución de distancia del acumulador en píxeles.
theta	resolución angular del acumulador en radianes.
thresold	parámetro de umbral del acumulador. Solo se devuelven aquellas líneas que obtienen suficientes votos.
minLineLength	longitud mínima de la línea. Los segmentos de línea más cortos que eso son rechazados.
maxLineGap	espacio máximo permitido entre puntos en la misma línea para unirlos.

El siguiente código muestra una a plicación de la función **cv.HoughLinesP**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8UC3);
let lines = new cv.Mat();
let color = new cv.Scalar(255, 0, 0);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
cv.Canny(src, src, 50, 200, 3);
// Puede probar con diferentes parámetros
cv.HoughLinesP(src, lines, 1, Math.PI / 180, 2, 0, 0);
// draw lines
for (let i = 0; i < lines.rows; ++i) {
    let startPoint = new cv.Point(lines.data32S[i * 4], lines.data32S[i * 4 +
1]);
    let endPoint = new cv.Point(lines.data32S[i * 4 + 2], lines.data32S[i * 4
+ 3]);
    cv.line(dst, startPoint, endPoint, color);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); lines.delete();
```

4132_houghLinesP.html

Resultados



4.14.- Transformada de Cŕculo Hough

4.14.1.- Objetivos

- Aprenderemos a usar la Transformada de Hough para encontrar círculos en una imagen.
- Aprenderemos esta funcion: **cv.HoughCircles()**

4.14.2.- Teoría

Un círculo se representa matemáticamente como (x−xcenter)2 + (y−ycenter)2=r2 donde (xcenter,ycenter) es el centro del círculo, y r es el radio del círculo. A partir de la ecuación, podemos ver que tenemos 3 parámetros, por lo que necesitamos un acumulador 3D para la transformación Hough, que sería muy ineficaz. Entonces, OpenCV usa un método más complicado, el Método de gradiente Hough, que usa la información del gradiente de los bordes.

Utilizamos la función:

cv.HoughCircles (image, circles, method, dp, minDist, param1 = 100, param2 = 100, minRadius = 0, maxRadius = 0)

Parámetros:

image	Imagen de entrada en escala de grises de un solo canal de 8 bits.
circles	vector de salida de círculos encontrados (tipo cv.CV_32FC3). Cada vector está odificado como un vector de coma flotante de 3 elementos (x, y, radio).
method	método de detección (ver cv.HoughModes). Actualmente, el único método implementado es HOUGH_GRADIENT
dp	razón inversa de la resolución del acumulador a la resolución de la imagen. Por ejemplo, si dp = 1, el acumulador tiene la misma resolución que la imagen de entrada. Si dp = 2, el acumulador tiene la mitad de ancho y alto.
minDist	distancia mínima entre los centros de los círculos detectados. Si el parámetro es demasiado pequeño, se pueden detectar falsamente múltiples círculos vecinos además de uno verdadero. Si es demasiado grande, es posible que se pierdan algunos círculos.
param1	primer parámetro específico del método. En el caso de HOUGH_GRADIENT , es

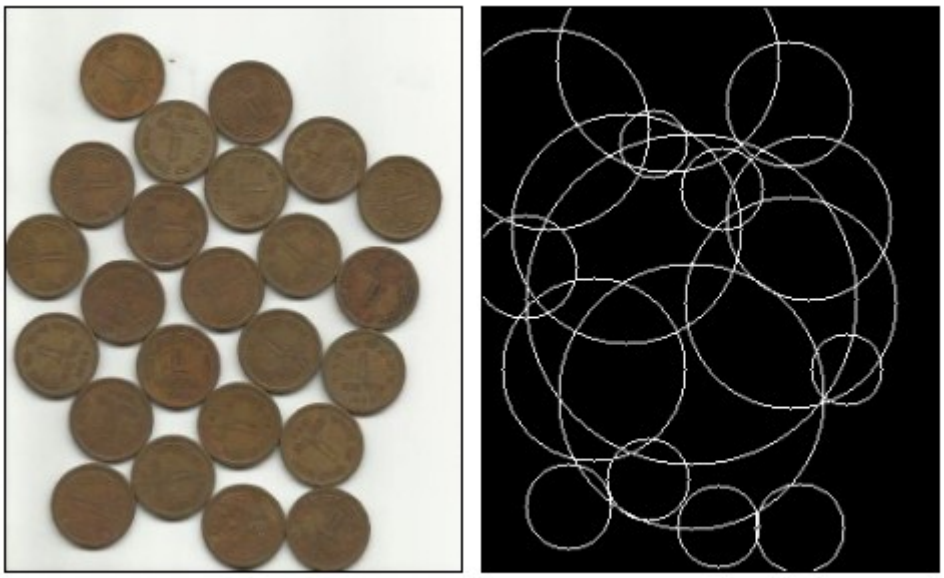
	el umbral más alto de los dos pasados al detector de bordes Canny (el más bajo es dos veces más pequeño).
param2	segundo parámetro específico del método. En el caso de HOUGH_GRADIENT , es el umbral del acumulador para los centros del círculo en la etapa de detección. Cuanto más pequeño es, más círculos falsos se pueden detectar. Los círculos, correspondientes a los valores acumulados más grandes, se devolverán primero.
minRadius	radio mínimo del círculo.
maxRadius	radio máximo del círculo.

El siguiente código muestra una aplicación de la función **cv.HoughCircles**

```
let src = cv.imread('canvasInput');
let dst = cv.Mat.zeros(src.rows, src.cols, cv.CV_8U);
let circles = new cv.Mat();
let color = new cv.Scalar(255, 0, 0);
cv.cvtColor(src, src, cv.COLOR_RGBA2GRAY, 0);
// Puede probar con diferentes parámetros
cv.HoughCircles(src, circles, cv.HOUGH_GRADIENT,
                1, 45, 75, 40, 0, 0);
// draw circles
for (let i = 0; i < circles.cols; ++i) {
    let x = circles.data32F[i * 3];
    let y = circles.data32F[i * 3 + 1];
    let radius = circles.data32F[i * 3 + 2];
    let center = new cv.Point(x, y);
    cv.circle(dst, center, radius, color);
}
cv.imshow('canvasOutput', dst);
src.delete(); dst.delete(); circles.delete();
```

4141_hough_circulos.html

Resultados



4.15.- Segmentación de imagen con el algoritmo de Watershed

4.15.1.- Objetivos

- Aprenderemos cómo usar la segmentación de imágenes basada en marcadores usando el algoritmo de cuencas hidrográficas.
- Aprenderemos: **cv.watershed()**

4.15.2.- Teoría

Cualquier imagen en escala de grises se puede ver como una superficie topográfica donde la alta intensidad denota picos y colinas, mientras que la baja intensidad denota valles. Empiezas llenando cada valle aislado (mínimo local) con agua de diferentes colores (etiquetas). A medida que el agua sube, dependiendo de los picos (gradientes) cercanos, el agua de diferentes valles, obviamente con diferentes colores, comenzará a fusionarse. Para evitar eso, construye barreras en los lugares donde el agua se fusiona. Continúas el trabajo de llenar de agua y construir barreras hasta que todos los picos estén bajo el agua. Luego, las barreras que creaste te dan el resultado de la segmentación. Esta es la "filosofía" detrás de la línea divisoria de aguas. Puede visitar la página web de CMM sobre la cuenca hidrográfica para entenderlo con la ayuda de algunas animaciones.

Pero este enfoque le brinda un resultado sobresegmentado debido al ruido o cualquier otra irregularidad en la imagen. Entonces, OpenCV implementó un algoritmo de cuenca hidrográfica basado en marcadores en el que especifica cuáles son todos los puntos del valle que se fusionarán y cuáles no. Es una segmentación de imágenes interactiva. Lo que hacemos es dar diferentes etiquetas a nuestro objeto que conocemos. Etiquetar la región que estamos seguros de ser el primer plano u objeto con un color (o intensidad), etiquetar la región que estamos seguros de ser el fondo o no objeto con otro color y finalmente la región de la que no estamos seguros de nada, etiquételo con 0. Ese es nuestro marcador. Luego aplique el algoritmo de cuenca hidrográfica. Luego, nuestro

marcador se actualizará con las etiquetas que le dimos, y los límites de los objetos tendrán un valor de -1.

4.15.3.- Código

A continuación, veremos un ejemplo de cómo usar la Transformación de distancia junto con la línea divisoria de aguas para segmentar objetos que se tocan entre sí.

Considere la imagen de las monedas a continuación, las monedas se tocan entre sí. Incluso si lo limita, se tocarán entre sí. Empezamos por encontrar una estimación aproximada de las monedas. Para eso, podemos usar la binarización de Otsu.

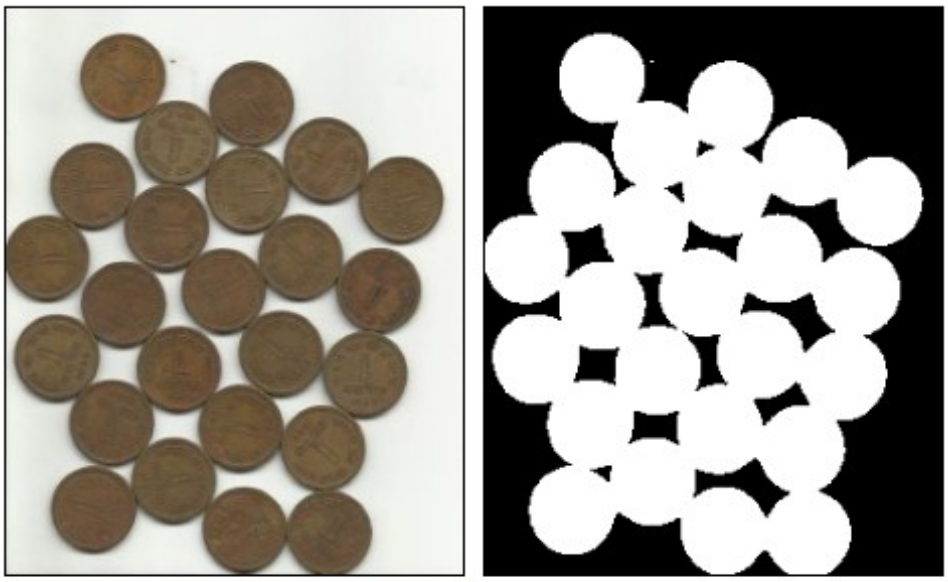
4.15.4.- Binarización de Otsu (Image thresold)

El siguiente código muestra como se identifican unas monedas y se representan en color blanco sobre un fondo negro.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let gray = new cv.Mat();

// convertir a grises y umbralizar la imagen
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
// El flag THRESH_OTSU, utiliza el algoritmos de Otsu para elegir el valor
óptimo de umbral
cv.threshold(gray, gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU);

cv.imshow('canvasOutput', gray);
src.delete(); dst.delete(); gray.delete();
```

4151_imagen_thresold.html

Resultados



4.15.5.- Fondo de imagen

Ahora necesitamos eliminar cualquier pequeño ruido blanco en la imagen. Para eso podemos usar la apertura morfológica. Para eliminar pequeños agujeros en el objeto, podemos usar el cierre morfológico. Entonces, ahora sabemos con certeza que la región cercana al centro de los objetos está en primer plano y la región muy alejada del objeto está en segundo plano. La única región de la que no estamos seguros es la región límite de las monedas.

Entonces necesitamos extraer el área que estamos seguros de que son monedas. La erosión elimina los píxeles del límite. Entonces, lo que quede, podemos estar seguros de que es moneda. Eso funcionaría si los objetos no se tocaran entre sí. Pero dado que se tocan entre sí, otra buena opción sería encontrar la transformación de distancia y aplicar un umbral adecuado. A continuación, debemos encontrar el área en la que estamos seguros de que no son monedas. Para eso, dilatamos el resultado. La dilatación aumenta el límite del objeto al fondo. De esta manera, podemos asegurarnos de que cualquier región en el fondo que resulte sea realmente un fondo, ya que se elimina la región límite. Vea la imagen a continuación.

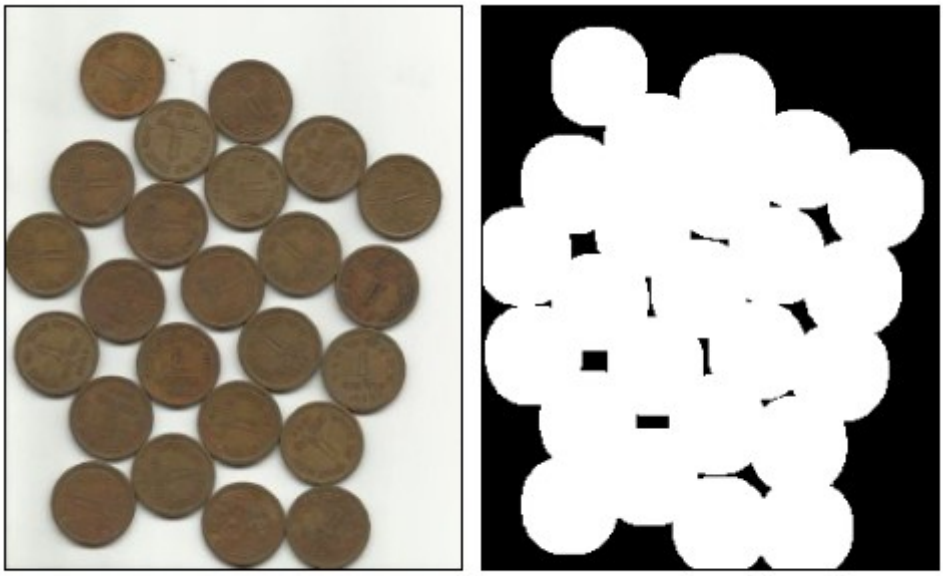
Las siguientes líneas de código muestran la imagen a la que nos referimos anteriormente.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let gray = new cv.Mat();
let opening = new cv.Mat();
let coinsBg = new cv.Mat();
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(gray, gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU);

// get background
let M = cv.Mat.ones(3, 3, cv.CV_8U);
cv.erode(gray, gray, M);
```

```
cv.dilate(gray, opening, M);
cv.dilate(opening, coinsBg, M, new cv.Point(-1, -1), 3);

cv.imshow('canvasOutput', coinsBg);
src.delete(); dst.delete(); gray.delete(); opening.delete();
coinsBg.delete(); M.delete();
```

4152_imagen_fondo.html

Resultados



canvasInput canvasOutput

4.15.6.- Transformación de distancia

Las regiones restantes son aquellas de las que no tenemos idea, ya sean monedas o fondo. El algoritmo Watershed debería encontrarlo. Estas áreas normalmente se encuentran alrededor de los límites de las monedas donde se encuentran el primer plano y el fondo (o incluso se encuentran dos monedas diferentes). Lo llamamos frontera. Se puede obtener restando el área **sure_fg** del área **sure_bg**.

Utilizamos la función:

cv.distanceTransform (src, dst, distanceType, maskSize, labelType = cv.CV_32F)

src	Imagen de origen de un solo canal (binario) de 8 bits.
dst	imagen de salida con distancias calculadas. Es una imagen de un solo canal de punto flotante de 8 o 32 bits del mismo tamaño que src.
distanceType	tipo de distancia (ver cv.DistanceTypes)

maskSize tamaño de la máscara de transformación de distancia, consulte (cv.DistanceTransformMasks).

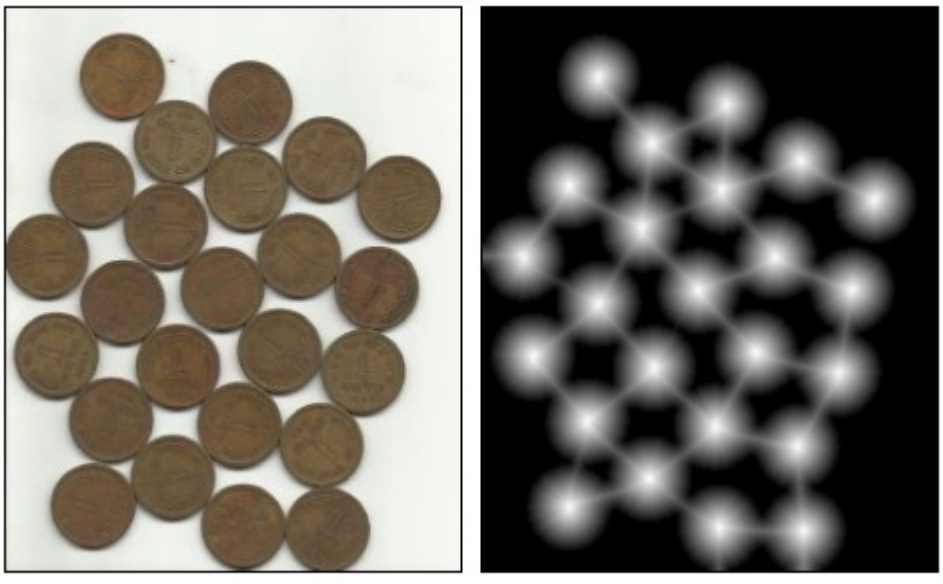
labelType tipo de imagen de salida. Puede ser cv.CV_8U o cv.CV_32F. El tipo cv.CV_8U solo se puede usar para la primera variante de la función y el tipo de distanceType == DIST_L1.

El siguiente código muestra la utilización de la función **cv.distanceTransform**

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let gray = new cv.Mat();
let opening = new cv.Mat();
let coinsBg = new cv.Mat();
let coinsFg = new cv.Mat();
let distTrans = new cv.Mat();
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(gray, gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU);
let M = cv.Mat.ones(3, 3, cv.CV_8U);
cv.erode(gray, gray, M);
cv.dilate(gray, opening, M);
cv.dilate(opening, coinsBg, M, new cv.Point(-1, -1), 3);

// transformación de distancia
cv.distanceTransform(opening, distTrans, cv.DIST_L2, 5);
cv.normalize(distTrans, distTrans, 1, 0, cv.NORM_INF);

cv.imshow('canvasOutput', distTrans);
src.delete(); dst.delete(); gray.delete(); opening.delete();
coinsBg.delete(); coinsFg.delete(); distTrans.delete(); M.delete();
```

4153_transformacion_distancia.html

Resultados



4.15.7.- Primer plano de la imagen

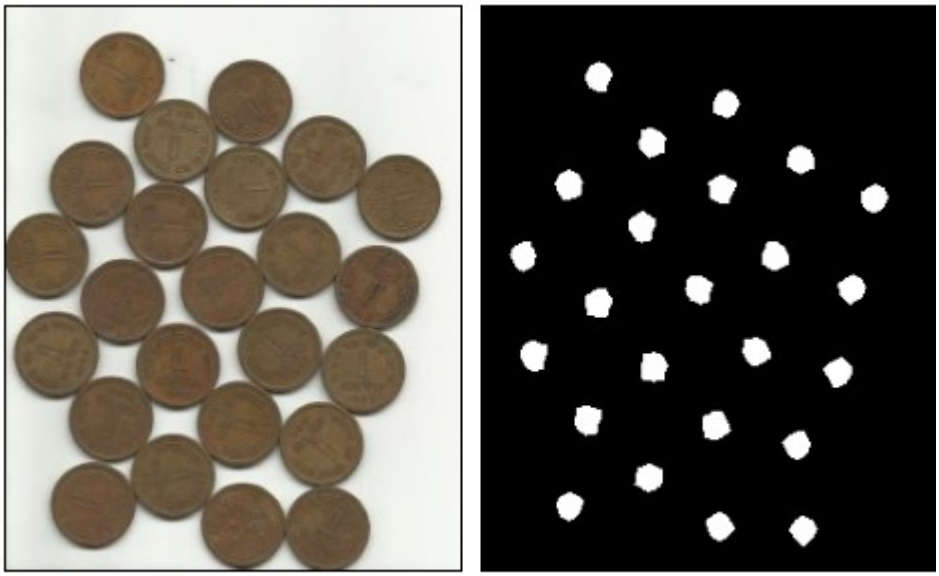
En la imagen umbralizada, tenemos algunas regiones de monedas de las que estamos seguros y ahora están separadas. (En algunos casos, puede estar interesado solo en la segmentación del primer plano, no en separar los objetos que se tocan entre sí. En ese caso, no necesita usar la transformación de distancia, solo la erosión es suficiente. La erosión es solo otro método para extraer un área de primer plano segura, eso es todo.)

El siguiente código muestra como encontrar el primer plano de la imagen.

```
let src = cv.imread('canvasInput');
let dst = new cv.Mat();
let gray = new cv.Mat();
let opening = new cv.Mat();
let coinsBg = new cv.Mat();
let coinsFg = new cv.Mat();
let distTrans = new cv.Mat();
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(gray, gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU);
let M = cv.Mat.ones(3, 3, cv.CV_8U);
cv.erode(gray, gray, M);
cv.dilate(gray, opening, M);
cv.dilate(opening, coinsBg, M, new cv.Point(-1, -1), 3);
cv.distanceTransform(opening, distTrans, cv.DIST_L2, 5);
cv.normalize(distTrans, distTrans, 1, 0, cv.NORM_INF);

// get foreground
cv.threshold(distTrans, coinsFg, 0.7 * 1, 255, cv.THRESH_BINARY);

cv.imshow('canvasOutput', coinsFg);
src.delete(); dst.delete(); gray.delete(); opening.delete();
coinsBg.delete(); coinsFg.delete(); distTrans.delete(); M.delete();
```

4154_primer_plano_image.html

Resultados



4.15.8.- Algoritmo de Watershed

Ahora sabemos con certeza cuáles son la región de las monedas, cuáles son el fondo y todo. Entonces creamos un marcador (es una matriz del mismo tamaño que la imagen original, pero con tipo de datos int32) y etiquetamos las regiones dentro de él. Las regiones que sabemos con certeza (ya sea en primer plano o en segundo plano) se etiquetan con números enteros positivos, pero con números enteros diferentes, y el área que no sabemos con certeza se deja como cero. Para esto usamos **cv.connectedComponents()**. Etiqueta el fondo de la imagen con 0, luego otros objetos se etiquetan con números enteros a partir de 1.

Pero sabemos que si el fondo está marcado con 0, la cuenca lo considerará como un área desconocida. Entonces queremos marcarlo con un entero diferente. En su lugar, marcaremos la región desconocida, definida por desconocido, con 0.

Ahora nuestro marcador está listo. Es hora del paso final, aplicar la cuenca hidrográfica. Luego se modificará la imagen del marcador. La región límite se marcará con -1.

Utilizaremos la función:

cv.connectedComponents (image, labels, connectivity = 8, ltype = cv.CV_32S)

Parámetros:

image imagen a etiquetar monocanal de 8 bits.

labels imagen etiquetada de destino (tipo cv.CV_32SC1).

connectivity 8 o 4 para conectividad de 8 o 4 vías respectivamente.

ltype tipo de imagen de etiqueta de salida. Actualmente se admiten cv.CV_32S y cv.CV_16U.

La otra función que utilizamos:

cv.watershed (image, markers)

Parámetros:

image imagen de entrada de 3 canales de 8 bits.

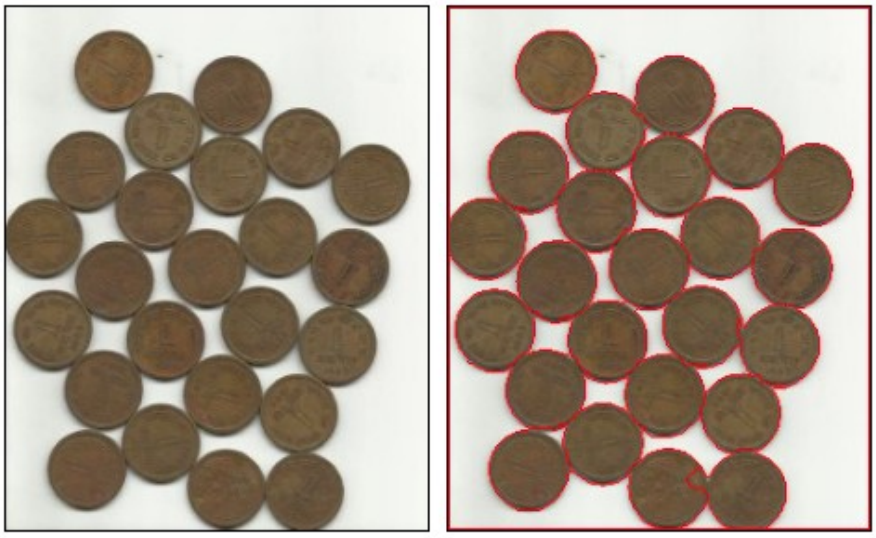
markers entrada/salida de imagen de un solo canal de 32 bits (mapa) de marcadores. Debe tener el mismo tamaño que la imagen.

El siguiente código permite obtener tanto el fondo como las monedas en primer plano en sus colores formas y colores originales.

```
let dst = new cv.Mat();
let gray = new cv.Mat();
let opening = new cv.Mat();
let coinsBg = new cv.Mat();
let coinsFg = new cv.Mat();
let distTrans = new cv.Mat();
let unknown = new cv.Mat();
let markers = new cv.Mat();
// gray and threshold image
cv.cvtColor(src, gray, cv.COLOR_RGBA2GRAY, 0);
cv.threshold(gray, gray, 0, 255, cv.THRESH_BINARY_INV + cv.THRESH_OTSU);
// get background
let M = cv.Mat.ones(3, 3, cv.CV_8U);
cv.erode(gray, gray, M);
cv.dilate(gray, opening, M);
cv.dilate(opening, coinsBg, M, new cv.Point(-1, -1), 3);
// distance transform
cv.distanceTransform(opening, distTrans, cv.DIST_L2, 5);
cv.normalize(distTrans, distTrans, 1, 0, cv.NORM_INF);
// get foreground
cv.threshold(distTrans, coinsFg, 0.7 * 1, 255, cv.THRESH_BINARY);
coinsFg.convertTo(coinsFg, cv.CV_8U, 1, 0);
cv.subtract(coinsBg, coinsFg, unknown);
// get connected components markers
cv.connectedComponents(coinsFg, markers);
for (let i = 0; i < markers.rows; i++) {
    for (let j = 0; j < markers.cols; j++) {
        markers.intPtr(i, j)[0] = markers.ucharPtr(i, j)[0] + 1;
        if (unknown.ucharPtr(i, j)[0] == 255) {
            markers.intPtr(i, j)[0] = 0;
        }
```

```
    }
}
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB, 0);
cv.watershed(src, markers);
// draw barriers
for (let i = 0; i < markers.rows; i++) {
    for (let j = 0; j < markers.cols; j++) {
        if (markers.intPtr(i, j)[0] == -1) {
            src.ucharPtr(i, j)[0] = 255; // R
            src.ucharPtr(i, j)[1] = 0; // G
            src.ucharPtr(i, j)[2] = 0; // B
        }
    }
}
cv.imshow('canvasOutput', src);
src.delete(); dst.delete(); gray.delete(); opening.delete();
coinsBg.delete();
coinsFg.delete(); distTrans.delete(); unknown.delete(); markers.delete();
M.delete();
```

Resultados



canvasInput canvasOutput

4.16.- Extraccion del primer plano con el algoritmo GrabCut

4.16.1.- Objetivos

- Aprenderemos el algoritmo GrabCut para extraer el primer plano en las imágenes.

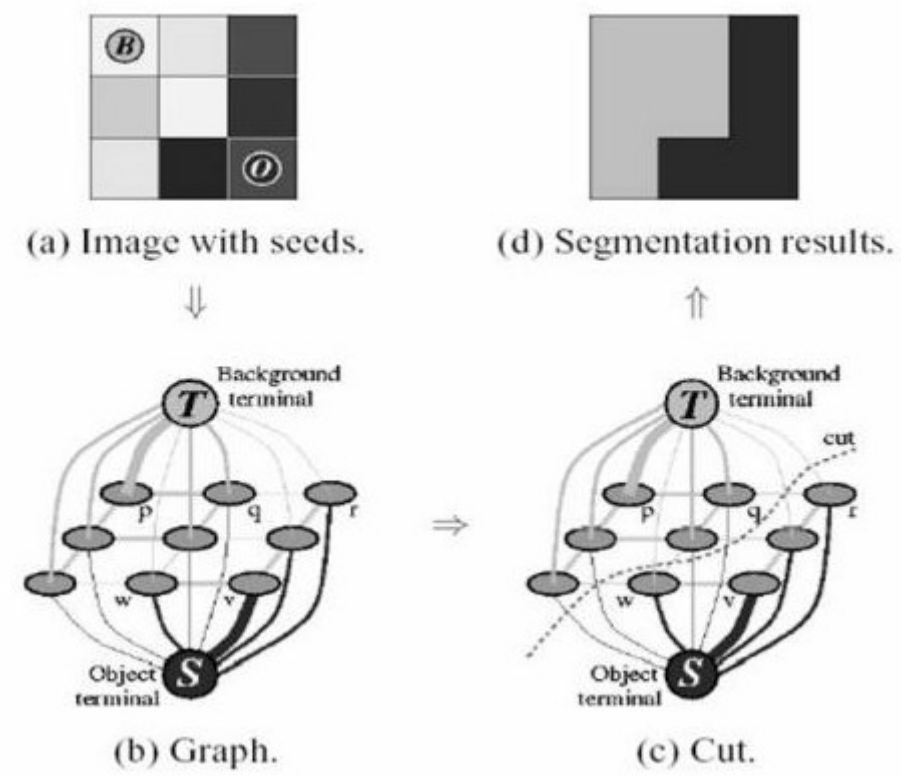
El algoritmo GrabCut fue diseñado por Carsten Rother, Vladimir Kolmogorov y Andrew Blake de Microsoft Research Cambridge, Reino Unido, en su artículo, "GrabCut": extracción interactiva de primer plano mediante cortes de gráfico iterados. Se necesitaba un algoritmo para la extracción de primer plano con una interacción mínima del usuario, y el resultado fue GrabCut.

¿Cómo funciona desde el punto de vista del usuario? Inicialmente, el usuario dibuja un rectángulo alrededor de la región de primer plano (la región de primer plano debe estar

completamente dentro del rectángulo). Luego, el algoritmo lo segmenta iterativamente para obtener el mejor resultado. Hecho. Pero en algunos casos, la segmentación no estará bien, por ejemplo, puede haber marcado alguna región de primer plano como fondo y viceversa. En ese caso, el usuario necesita hacer retoques finos. Simplemente dé algunos trazos en las imágenes donde haya algunos resultados defectuosos. Strokes básicamente dice *"Oye, esta región debería estar en primer plano, la marcaste como fondo, corrígela en la próxima iteración"* o su opuesto para el fondo. Luego, en la siguiente iteración, obtendrá mejores resultados.

¿Qué sucede en segundo plano?

- El usuario ingresa el rectángulo. Todo lo que esté fuera de este rectángulo se tomará como fondo seguro (esa es la razón por la que se mencionó antes que su rectángulo debe incluir todos los objetos). Todo lo que está dentro del rectángulo es desconocido. Del mismo modo, cualquier entrada del usuario que especifique el primer plano y el fondo se considera de etiquetado estricto, lo que significa que no cambiará en el proceso.
- El ordenador hace un etiquetado inicial en función de los datos que le hemos dado. Etiqueta los píxeles de primer plano y de fondo (o etiquetas duras).
- Ahora se utiliza un modelo de mezcla gaussiana (GMM) para modelar el primer plano y el fondo.
- Dependiendo de los datos que proporcionamos, GMM aprende y crea una nueva distribución de píxeles. Es decir, los píxeles desconocidos se etiquetan como primer plano probable o fondo probable según su relación con los otros píxeles etiquetados de forma rígida en términos de estadísticas de color (es como agrupar).
- Se construye un gráfico a partir de esta distribución de píxeles. Los nodos en los gráficos son píxeles. Se agregan dos nodos adicionales, el nodo de origen y el nodo de receptor. Cada píxel de primer plano está conectado al nodo Fuente y cada píxel de fondo está conectado al nodo Sumidero.
- Los pesos de los bordes que conectan los píxeles con el nodo de origen y el nodo final se definen por la probabilidad de que un píxel esté en primer plano/fondo. Los pesos entre los píxeles se definen por la información del borde o la similitud de píxeles. Si hay una gran diferencia en el color de los píxeles, el borde entre ellos tendrá un peso bajo.
- Luego se usa un algoritmo mincut para segmentar el gráfico. Corta el gráfico en dos nodos fuente y nodo sumidero separados con una función de costo mínimo. La función de costo es la suma de todos los pesos de los bordes que se cortan. Después del corte, todos los píxeles conectados al nodo Fuente pasan a primer plano y los conectados al nodo Sumidero pasan a segundo plano.
- El proceso continúa hasta que la clasificación converge.



Utilizaremos la función:

cv.grabCut (image, mask, rect, bgdModel, fgdModel, iterCount,
mode = cv.GC_EVAL)

Parámetros:

image entrada de imagen de 3 canales de 8 bits

mask Máscara monocanal de entrada/salida de 8 bits. La función inicializa la máscara cuando el modo se establece en GC_INIT_WITH_RECT. Sus elementos pueden tener uno de los cv.rabCutClasses.

rect ROI que contiene un objeto segmentado. Los píxeles fuera del ROI están marcados como "fondo obvio". El parámetro solo se usa cuando mode==GC_INIT_WITH_RECT.

bgdModel matriz temporal para el modelo de fondo. No lo modifique mientras esté procesando la misma imagen.

fgdModel matriz temporales para el modelo de primer plano. No lo modifique mientras esté procesando la misma imagen.

iterCount	número de iteraciones que debe hacer el algoritmo antes de devolver el resultado. Tenga en cuenta que el resultado se puede refinar con más llamadas con mode==GC_INIT_WITH_MASK o mode==GC_EVAL .
mode	modo de operación que podría ser uno de los cv::GrabCutModes

El siguiente código muestra una aplicación de la función **cv.grabCut**

```
let src = cv.imread('canvasInput');
cv.cvtColor(src, src, cv.COLOR_RGBA2RGB, 0);
let mask = new cv.Mat();
let bgdModel = new cv.Mat();
let fgdModel = new cv.Mat();
let rect = new cv.Rect(50, 50, 260, 280);
cv.grabCut(src, mask, rect, bgdModel, fgdModel, 1, cv.GC_INIT_WITH_RECT);
// draw foreground
for (let i = 0; i < src.rows; i++) {
    for (let j = 0; j < src.cols; j++) {
        if (mask.ucharPtr(i, j)[0] == 0 || mask.ucharPtr(i, j)[0] == 2) {
            src.ucharPtr(i, j)[0] = 0;
            src.ucharPtr(i, j)[1] = 0;
            src.ucharPtr(i, j)[2] = 0;
        }
    }
}
// draw grab rect
let color = new cv.Scalar(0, 0, 255);
let point1 = new cv.Point(rect.x, rect.y);
let point2 = new cv.Point(rect.x + rect.width, rect.y + rect.height);
cv.rectangle(src, point1, point2, color);
cv.imshow('canvasOutput', src);
src.delete(); mask.delete(); bgdModel.delete(); fgdModel.delete();
```

4161_algoritmo_grabCut.html

canvasInput

canvasOutput

Resultados

4.17.- Procesamiento de imagen para captura de video

El siguiente programa captura el vídeo de la webcam instalada en el equipo y ofrece un menú para aplicar todos los filtros estudiados fotograma por fotograma a dicho vídeo obtenido de la webcam.

El listado es extenso y no tiene sentido reproducirlo aquí así que recomendamos analizarlo sobre un editor de texto. El programa es:

4171_captura_video_procesamiento.html

5.- Análisis de Video

En esta sección aprenderá diferentes técnicas para trabajar con videos como seguimiento de objetos, etc.

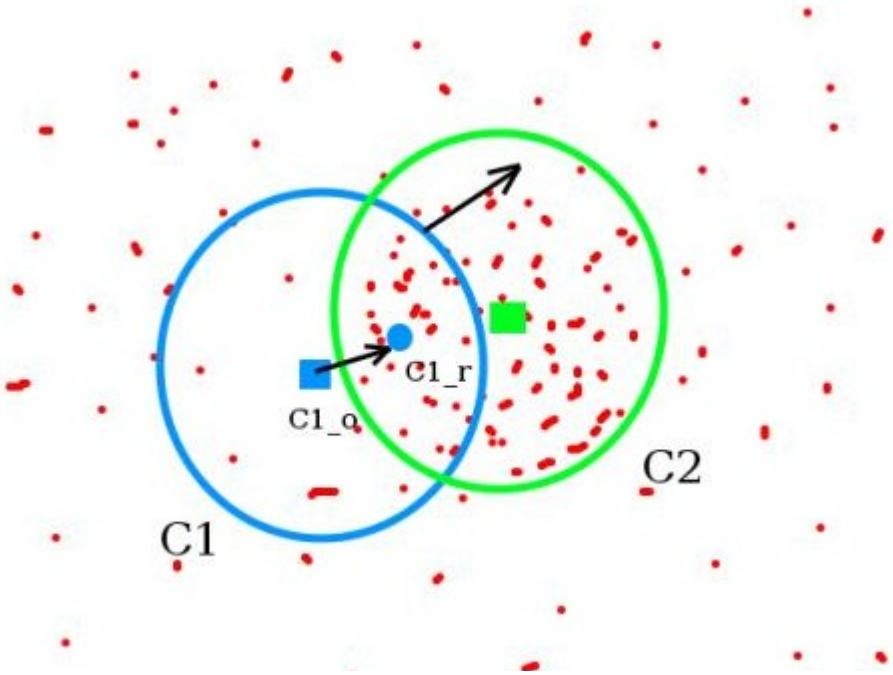
5.1.- Meanshift y Camshift

Aquí, aprenderemos sobre algoritmos de seguimiento como "Meanshift" y su versión mejorada, "Camshift" para encontrar y rastrear objetos en videos.

5.1.1.- Meanshift

Meanshift (desplazamiento medio)

La intuición detrás del cambio de media es simple. Considere que tiene un conjunto de puntos. (Puede ser una distribución de píxeles como una retroproyección de histograma). Aparece una pequeña ventana (puede ser un círculo) y debe mover esa ventana al área de máxima densidad de píxeles (o número máximo de puntos). Se ilustra en la imagen simple que se muestra a continuación:



La ventana inicial se muestra en un círculo azul con el nombre "C1". Su centro original está marcado en un rectángulo azul, llamado "C1_o". Pero si encuentra el centroide de los puntos dentro de esa ventana, obtendrá el punto "C1_r" (marcado en un pequeño círculo azul) que es

el centroide real de la ventana. Seguro que no coinciden. Así que mueva su ventana de manera que el círculo de la nueva ventana coincida con el centroide anterior. Nuevamente encuentre el nuevo centroide. Lo más probable es que no coincida. Así que muévalo nuevamente y continúe las iteraciones de manera que el centro de la ventana y su centroide caigan en la misma ubicación (o con un pequeño error deseado). Entonces, finalmente, lo que obtienes es una ventana con la máxima distribución de píxeles. Está marcado con un círculo verde, llamado "C2". Como puede ver en la imagen, tiene un número máximo de puntos.
Todo el proceso se demuestra en una imagen estática a continuación:

Por lo tanto, normalmente pasamos la imagen retroproyectada del histograma y la ubicación inicial del objetivo. Cuando el objeto se mueve, obviamente el movimiento se refleja en la imagen retroproyectada del histograma. Como resultado, el algoritmo de desplazamiento medio mueve nuestra ventana a la nueva ubicación con la máxima densidad.

5.1.1.1.- Meanshift en OpenCV

Para usar el desplazamiento medio en OpenCV.js, primero debemos configurar el objetivo, encontrar su histograma para que podamos retroproyectar el objetivo en cada cuadro para el cálculo del desplazamiento medio. También necesitamos proporcionar la ubicación inicial de la ventana. Para el histograma, aquí solo se considera **Hue**. Además, para evitar valores falsos debido a la poca luz, los valores de poca luz se descartan mediante la función **cv.inRange()**.

Utilizamos la función:

cv.meanShift (probImage, window, criteria)

Parámetros:

probImage	Retroproyección del histograma del objeto. Ver cv.calcBackProject para más detalles.
window	Ventana de búsqueda inicial.
criteria	Criterios de parada para el algoritmo de búsqueda iterativa.

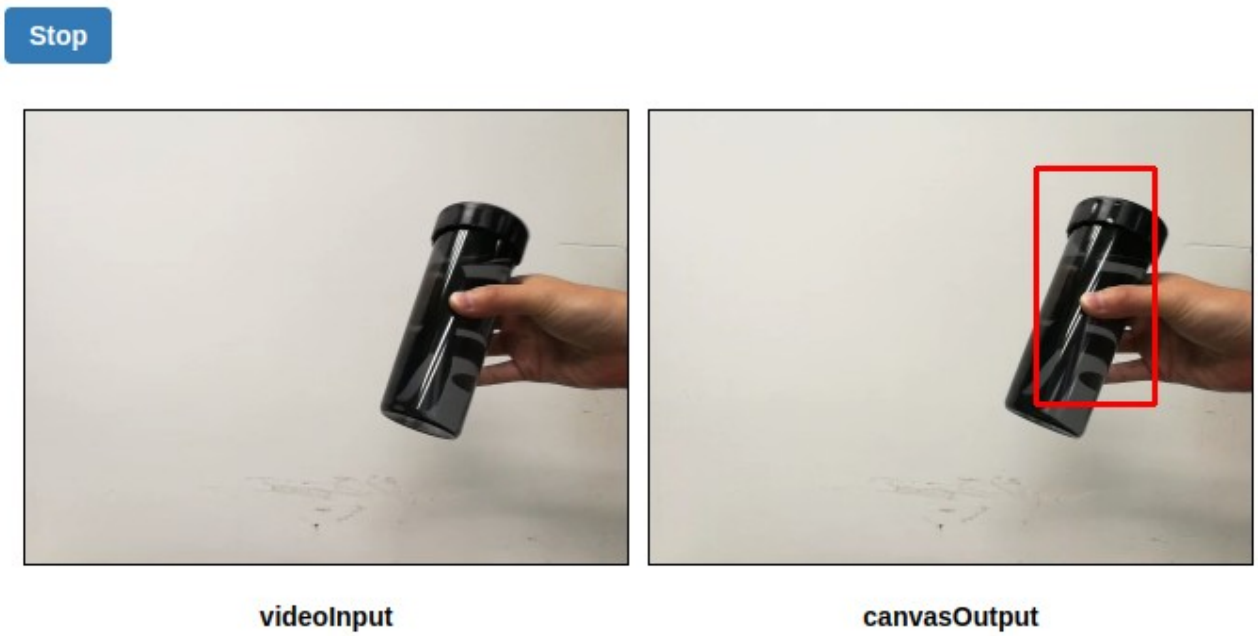
Retorna:

número de iteraciones que tomó meanShift para converger y la nueva ubicación

El siguiente listado muestra un aplicación del proceso **cv.meanShift**

5011_meanshift.html

Resultados



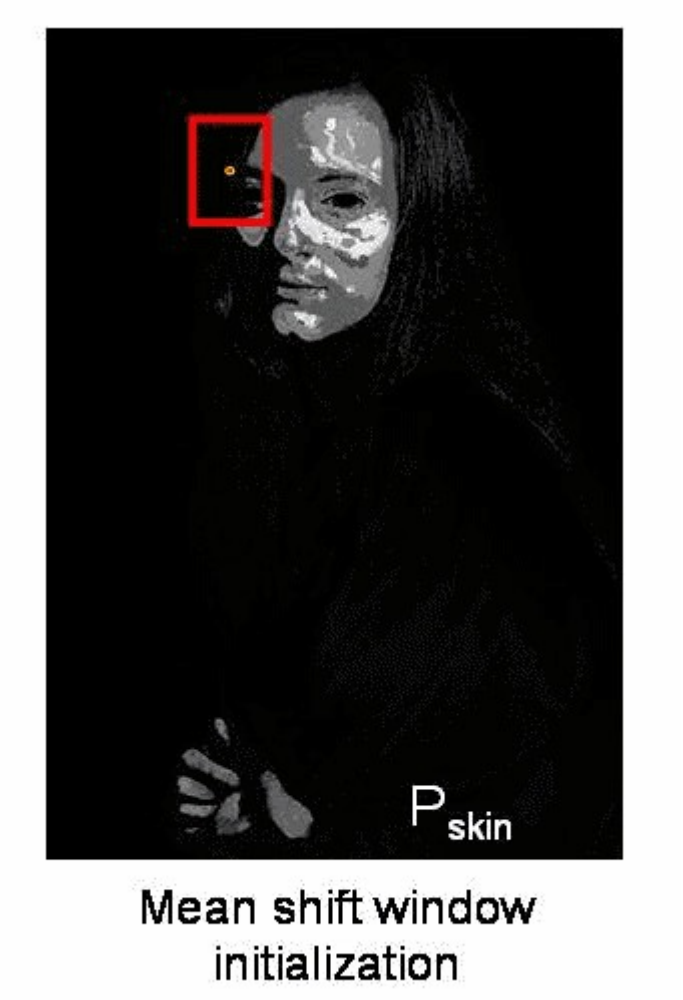
5.1.2.- Camshift

Camshift (Desplazamiento de levas)

¿Seguiste de cerca el último resultado? Hay un problema. Nuestra ventana siempre tiene el mismo tamaño cuando el objeto está más lejos y está muy cerca de la cámara. Eso no es bueno.

Necesitamos adaptar el tamaño de la ventana con el tamaño y la rotación del objetivo. Una vez más, la solución provino de "OpenCV Labs" y se llama CAMshift (Continuously Adaptive Meanshift) publicado por Gary Bradsky en su artículo "Seguimiento facial de visión por computadora para uso en una interfaz de usuario perceptual" en 1988.

Se aplica el desplazamiento medio primero. Una vez que el cambio medio converge, actualiza el tamaño de la ventana como, s = 2 × raiz(M00/256). También calcula la orientación de la elipse que mejor se ajusta a ella. De nuevo, aplica el desplazamiento medio con la nueva ventana de búsqueda escalada y la ubicación de la ventana anterior. El proceso continúa hasta que se alcanza la precisión requerida.



5.1.2.1.- Camshift in OpenCV.js

Es casi lo mismo que el cambio de medio, pero devuelve un rectángulo rotado (que es nuestro resultado) y parámetros de cuadro (solía pasar como ventana de búsqueda en la siguiente iteración).

Utilizamos la función:

cv.CamShift (probImage, window, criteria)

Parámetros:

probImage	retroproyección del histograma del objeto. Ver cv.calcBackProject para más detalles.
window	ventana de búsqueda inicial.
criteria	criterios de parada para el algoritmo de búsqueda iterativa.

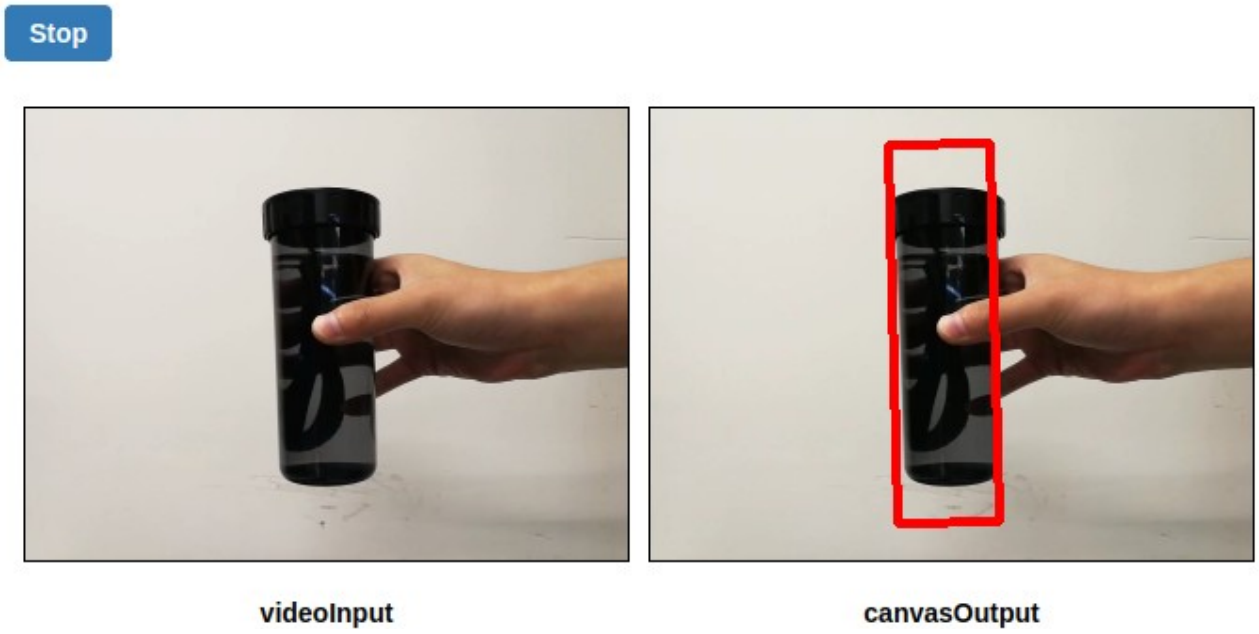
Retorna

rectángulo girado y la nueva ventana de búsqueda

Puede encontrar una aplicación de la función **cv.CamShift** en el siguiente programa.

5012_camshift.html

Resultados



5.2.- Flujo óptico

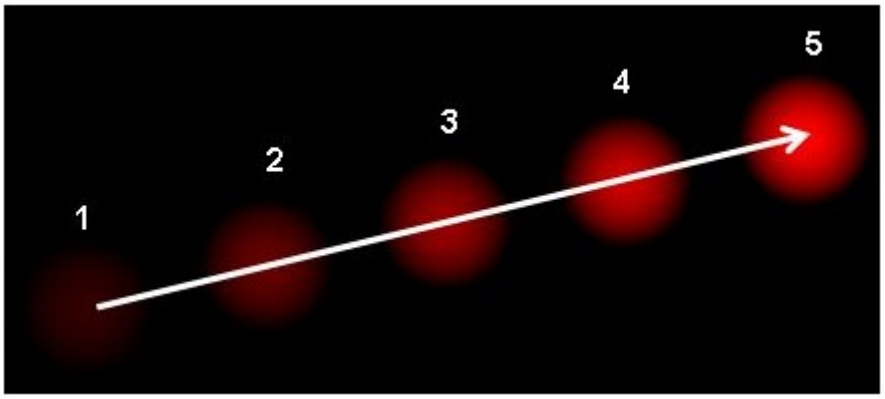
Ahora analicemos un concepto importante, "Flujo óptico", que está relacionado con videos y tiene muchas aplicaciones.

5.2.1.- Objetivos

- Comprenderemos los conceptos de flujo óptico y su estimación mediante el método de Lucas-Kanade.
- Usaremos funciones como **cv.calcOpticalFlowPyrLK()** para rastrear puntos característicos en un video.

5.2.2.- Flujo óptico

El flujo óptico es el patrón de movimiento aparente de los objetos de la imagen entre dos fotogramas consecutivos causados por el movimiento del objeto o la cámara. Es un campo vectorial 2D donde cada vector es un vector de desplazamiento que muestra el movimiento de los puntos desde el primer cuadro al segundo. Considere la imagen a continuación (Imagen cortesía: artículo de Wikipedia sobre flujo óptico).



Muestra una pelota moviéndose en 5 fotogramas consecutivos. La flecha muestra su vector de desplazamiento. El flujo óptico tiene muchas aplicaciones en áreas como:

- Estructura de movimiento
- Compresión de video
- Estabilización de vídeo …

El flujo óptico funciona con varios supuestos:

1. Las intensidades de píxeles de un objeto no cambian entre fotogramas consecutivos.

2. Los píxeles vecinos tienen un movimiento similar.

Considere un píxel I (x, y, t) en el primer cuadro (compruebe que se agrega una nueva dimensión, el tiempo, aquí. Anteriormente estábamos trabajando solo con imágenes, por lo que no necesitamos tiempo). Se mueve por ina distancia (dx,dy) en el siguiente cuadro tomado después del tiempo dt. Entonces, dado que esos píxeles son los mismos y la intensidad no cambia, podemos decir:

$$I(x,y,t)=I(x+dx,y+dy,t+dt)$$

Luego tome la aproximación de la serie de Taylor del lado derecho, elimine los términos comunes y divida por dt para obtener la siguiente ecuación:

$$f_x u + f_y v + f_t = 0$$

donde:

$$f_x = \frac{\partial f}{\partial x} \; ; \; f_y = \frac{\partial f}{\partial y}$$

$$u = \frac{dx}{dt} \; ; \; v = \frac{dy}{dt}$$

La ecuación anterior se llama ecuación de flujo óptico. En él podemos encontrar fx y fy, son gradientes de imagen. Del mismo modo, ft es el gradiente a lo largo del tiempo. Pero (u,v) es desconocido. No podemos resolver esta ecuación con dos variables desconocidas. Así que se proporcionan varios métodos para resolver este problema y uno de ellos es Lucas-Kanade.

5.2.3.- Método Lucas-Kanade

Hemos visto una suposición antes, que todos los píxeles vecinos tendrán un movimiento similar. El método de Lucas-Kanade toma un parche de 3x3 alrededor del punto. Entonces todos los 9 puntos tienen el mismo movimiento. Podemos encontrar (fx,fy,ft) para estos 9 puntos. Así que ahora nuestro problema se convierte en resolver 9 ecuaciones con dos variables desconocidas que están sobredeterminadas. Se obtiene una mejor solución con el método de ajuste de mínimos cuadrados. A continuación se muestra la solución final, que es un problema desconocido de dos ecuaciones y dos, y se resuelve para obtener la solución.

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i {f_{x_i}}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i {f_{y_i}}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix}$$

(Verifique la similitud de la matriz inversa con el detector de esquinas de Harris. Denota que las esquinas son mejores puntos para rastrear).

Entonces, desde el punto de vista del usuario, la idea es simple, damos algunos puntos para rastrear, recibimos los vectores de flujo óptico de esos puntos. Pero de nuevo hay algunos problemas. Hasta ahora, tratábamos de pequeños movimientos. Entonces falla cuando hay un gran movimiento. Así que de nuevo vamos por las pirámides. Cuando subimos en la pirámide, los pequeños movimientos se eliminan y los grandes movimientos se convierten en pequeños movimientos. Entonces, aplicando Lucas-Kanade allí, obtenemos flujo óptico junto con la escala.

5.2.4.- Flujo óptico de Lucas-Kanade en OpenCV.js

Utilizamos la función:

cv.calcOpticalFlowPyrLK (prevImg, nextImg, prevPts, nextPts,
status, err, winSize = new cv.Size(21, 21),
maxLevel = 3, criteria = new
cv.TermCriteria(cv.TermCriteria_COUNT+ cv.TermCriteria_EPS,
30, 0.01), flags = 0, minEigThreshold = 1e-4).

previmg primera imagen o pirámide de entrada de 8 bits construida por buildOpticalFlowPyramid.

nextimg segunda imagen de entrada o pirámide del mismo tamaño y del mismo tipo que prevImg.

prevPts vector de puntos 2D para los que se necesita encontrar el flujo; las coordenadas de los puntos deben ser números de punto flotante de precisión simple.

nextPts vector de salida de puntos 2D (con coordenadas de punto flotante de precisión simple) que contienen las nuevas posiciones calculadas de las entidades de entrada en la segunda imagen; cuando se pasa el indicador cv.OPTFLOW_USE_ INITIAL_FLOW, el vector debe tener el mismo tamaño que en la entrada.

status vector de estado de salida (de caracteres sin firmar); cada elemento del vector se

establece en 1 si se ha encontrado el flujo para las características correspondientes; de lo contrario, se establece en 0.

err vector de salida de errores; cada elemento del vector se establece en un error para la característica correspondiente, el tipo de medida de error se puede establecer en el parámetro de banderas; si no se encontró el flujo, entonces el error no está definido (use el parámetro de estado para encontrar tales casos)

winSize tamaño de la ventana de búsqueda en cada nivel de la pirámide

maxLevel número de nivel de pirámide máximo basado en 0; si se establece en 0, no se utilizan pirámides (un solo nivel), si se establece en 1, se utilizan dos niveles, y así sucesivamente; si las pirámides se pasan a la entrada, el algoritmo usará tantos niveles como las pirámides tengan, pero no más que maxLevel.

criteria parámetro, que especifica los criterios de finalización del algoritmo de búsqueda iterativa (después del número máximo especificado de iteraciones criterio.maxCount o cuando la ventana de búsqueda se mueve menos de criterio.epsilon.

flags banderas de operación:

cv.OPTFLOW_USE_INITIAL_FLOW usa estimaciones iniciales, almacenadas en nextPts; si el indicador no está establecido, entonces prevPts se copia en nextPts y se considera la estimación inicial.

cv.OPTFLOW_LK_GET_MIN_EIGENVALS utiliza valores propios mínimos como medida de error (consulte la descripción de minEigThreshold); si la bandera no está configurada, entonces la distancia L1 entre los parches alrededor del original y un punto movido, dividida por el número de píxeles en una ventana, se usa como medida de error.

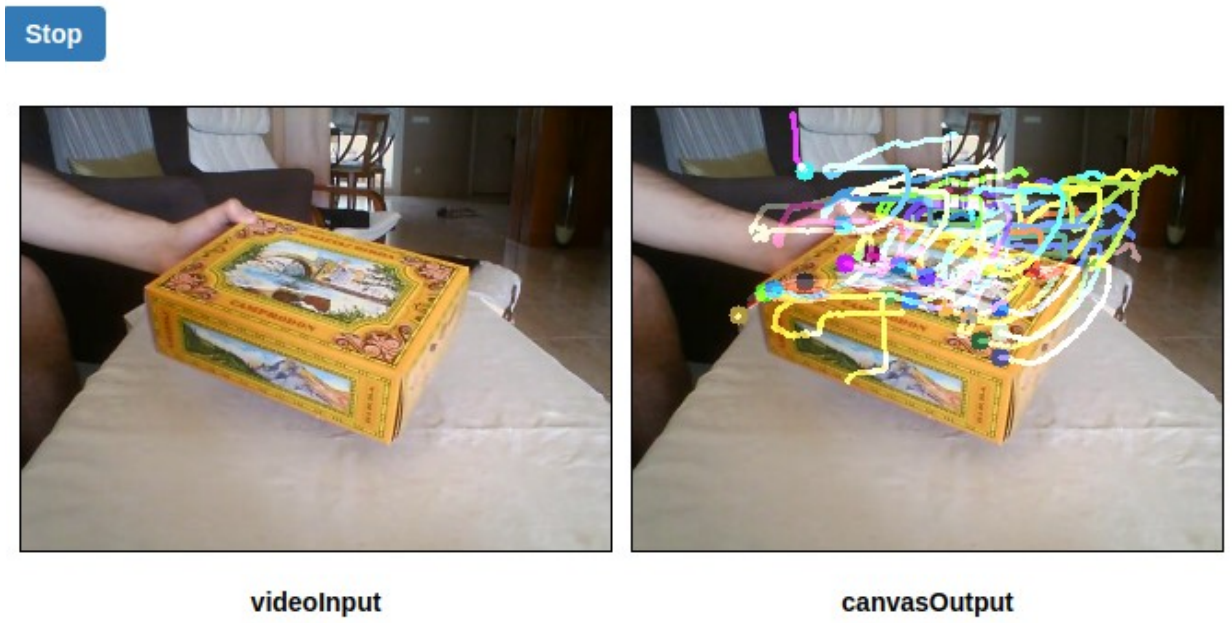
minEigThreshold el algoritmo calcula el valor propio mínimo de una matriz normal de 2x2 de ecuaciones de flujo óptico, dividido por el número de píxeles en una ventana; si este valor es menor que minEigThreshold, entonces se filtra la característica correspondiente y su flujo no se procesa, por lo que permite eliminar puntos defectuosos y obtener un aumento de rendimiento.

(Este código no verifica qué tan correctos son los siguientes puntos clave. Entonces, incluso si algún punto característico desaparece en la imagen, existe la posibilidad de que el flujo óptico encuentre el siguiente punto que puede parecer cercano a él. Entonces, en realidad, para un seguimiento sólido, los puntos de esquina deben detectarse en intervalos particulares).

El siguiente programa muestra la utilización de la función **cv.calcOpticalFlowPyrLK**

5021_Flujo_Optico_Lucas_Kanade copy.html

Resultados



5.2.5.- Flujo óptico denso en OpenCV.js

El método de Lucas-Kanade calcula el flujo óptico para un conjunto de características dispersas (en nuestro ejemplo, las esquinas se detectan con el algoritmo de Shi-Tomasi). OpenCV.js proporciona otro algoritmo para encontrar el flujo óptico denso. Calcula el flujo óptico para todos los puntos en el cuadro. Se basa en el algoritmo de Gunnar Farneback que se explica en "Estimación de movimiento de dos fotogramas basada en expansión polinomial" por Gunnar Farneback en 2003. ("Two-Frame Motion Estimation Based on Polynomial Expansion" by Gunnar Farneback in 2003.)

Utilizaremos la función:

cv.calcOpticalFlowFarneback (prev, next, flow, pyrScale, levels, winsize, iterations, polyN, polySigma, flags)

Parámetros:

prev primera imagen de entrada de un solo canal de 8 bits.

next segunda imagen de entrada del mismo tamaño y del mismo tipo que la anterior.

flow imagen de flujo calculada que tiene el mismo tamaño que la anterior y el mismo tipo CV_32FC2.

pyrScale parámetro, especificando la escala de la imagen (<1) para construir pirámides para cada imagen; pyrScale=0.5 significa una pirámide clásica, donde cada siguiente capa es dos veces más pequeña que la anterior.

levels número de capas de la pirámide incluyendo la imagen inicial; levels = 1 significa que no se crean capas adicionales y solo se utilizan las imágenes originales.

winsize promediar el tamaño de la ventana; Los valores más grandes aumentan la solidez del algoritmo frente al ruido de la imagen y brindan más posibilidades de detección de movimiento rápido, pero producen un campo de movimiento más borroso.

iterations número de iteraciones que hace el algoritmo en cada nivel de la pirámide.

polyN tamaño de la vecindad de píxeles utilizada para encontrar la expansión polinomial en cada píxel; los valores más grandes significan que la imagen se aproximará con superficies más suaves, lo que generará un algoritmo más sólido y un campo de movimiento más borroso, normalmente polyN = 5 o 7.

polySigma desviación estándar de la Gaussiana que se usa para suavizar las derivadas que se usan como base para la expansión del polinomio; para polyN=5, puede configurar polySigma=1.1, para poliN=7, un buen valor sería polySigma=1.5.

flags Indicadores de operación que pueden ser una combinación de los siguientes:

cv.OPTFLOW_USE_INITIAL_FLOW usa el flujo de entrada como una aproximación de flujo inicial.

cv.OPTFLOW_FARNEBACK_GAUSSIAN utiliza el filtro Gaussiano `winsize×winsize` en lugar de un filtro de caja del mismo tamaño para la estimación del flujo óptico; por lo general, esta opción brinda un flujo z más preciso que con un filtro de caja, a costa de una velocidad más baja; normalmente, winsize para una ventana gaussiana debe establecerse en un valor mayor para lograr el mismo nivel de robustez.

El siguiente programa muestra una aplicación de la función **cv.calcOpticalFlowFarneback**

5022_flujo_optico_denso.html

5.3.- Sustracción del fondo

5.3.1.- Obetivos

Nos familiarizaremos con los métodos de sustracción de fondo disponibles en OpenCV.js.

5.3.2.- Conceptos básicos

La sustracción de fondo es uno de los principales pasos de preprocesamiento en muchas aplicaciones basadas en visión. Por ejemplo, considere los casos como el contador de visitantes donde una cámara estática registra el número de visitantes que ingresan o salen de la habitación, o una cámara de tránsito que extrae información sobre los vehículos, etc. En todos estos casos, primero debe extraer solo a la persona o los vehículos. Técnicamente, debe extraer el primer plano en movimiento del fondo estático.

Si solo tiene una imagen de fondo, como la imagen de la habitación sin visitantes, la imagen de la carretera sin vehículos, etc., es un trabajo fácil. Simplemente reste el fond de la nueva imagen. Obtienes los objetos de primer plano solos. Pero en la mayoría de los casos, es posible que no tenga esa imagen, por lo que debemos extraer el fondo de cualquier imagen que tengamos. Se vuelve más complicado cuando hay sombra de los vehículos. Dado que la sombra también se está moviendo, la simple resta marcará eso también como primer plano. Complica las cosas.

OpenCV.js ha implementado un algoritmo para este propósito, que es muy fácil de usar.

5.3.3.- BackgroundSubtractorMOG2

Es una mezcla gaussiana basada en el algoritmo de segmentación Background/Foreground. Se basa en dos artículos de Z.Zivkovic, "Improved adaptive Gaussian mixture model for background subtraction" en 2004 y "Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction" en 2006. Una característica importante de este algoritmo es que selecciona el número apropiado de distribución gaussiana para cada píxel. Proporciona una mejor adaptabilidad a diferentes escenas debido a cambios de iluminación, etc.

Utilizaremos la función:

cv.BackgroundSubtractorMOG2 (history = 500, varThreshold = 16,
detectShadows = true)

Parámetros:

history	Longitud de la historia.
varThresold	Umbral de la distancia al cuadrado entre el píxel y la muestra para decidir si un píxel está cerca de esa muestra. Este parámetro no afecta la actualización en segundo plano.
detectShadows	Si es verdadero, el algoritmo detectará las sombras y las marcará. Disminuye un poco la velocidad, por lo que si no necesita esta función, establezca el parámetro en falso.

Return

instancia de cv.BackgroundSubtractorMOG2

La otra función que necesitamos para obtener el primer plano:

apply (image, fgmask, learningRate = -1)

Parámetros:

image	Siguiente fotograma de vídeo. El marco de punto flotante se utilizará sin escala y debe estar en el rango [0,255].

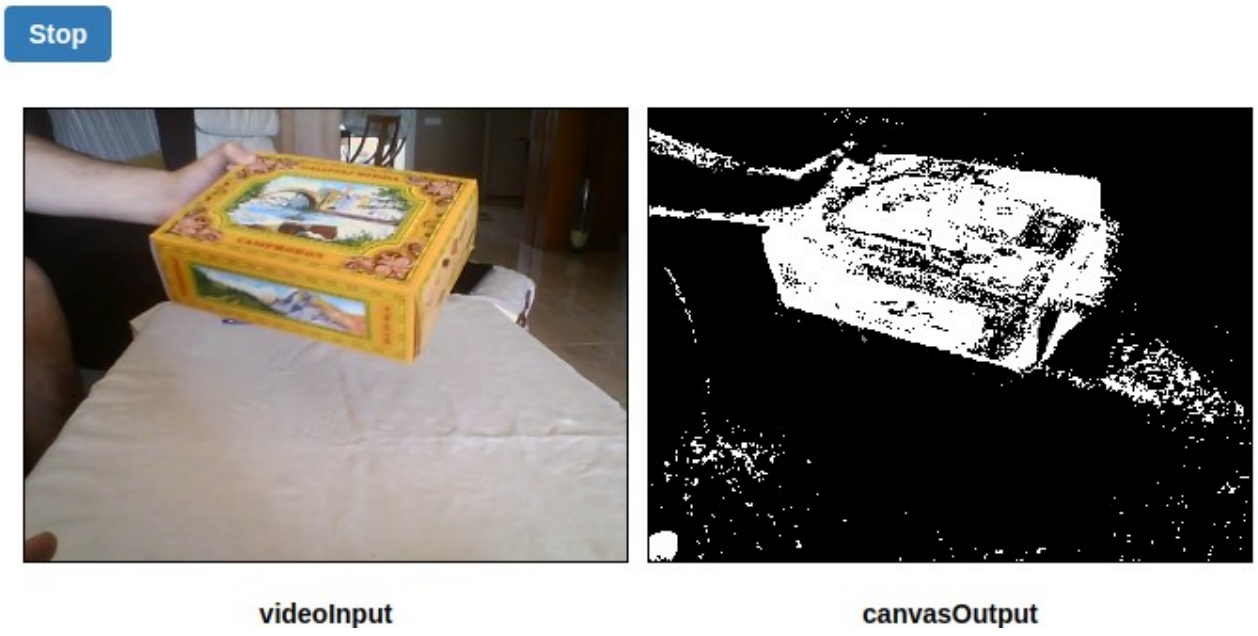
fgmask La máscara de primer plano de salida como una imagen binaria de 8 bits.

learningRate El valor entre 0 y 1 que indica qué tan rápido se aprende el modelo de fondo. El valor de parámetro negativo hace que el algoritmo utilice alguna tasa de aprendizaje elegida automáticamente. 0 significa que el modelo de fondo no se actualiza en absoluto, 1 significa que el modelo de fondo se reinicializa por completo desde el último fotograma.

El siguiente programa muestra la utilización de la función **cv.BackgroundSubtractorMOG2**

5031_substraer_fondo.html

Resultados



Nota

La instancia de cv.BackgroundSubtractorMOG2 debe eliminarse manualmente.

6.- Detección de Objetos

En esta sección, aprenderemos técnicas de detección de objetos como la detección de rostros, etc.

6.1.- Detección de rostros utilizando Haar Cascades

6.1.1.- Objetivos

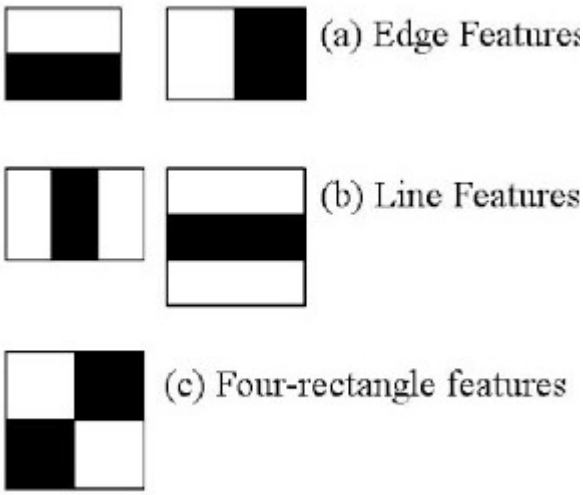
Aprenda los conceptos básicos de la detección de rostros utilizando los clasificadores en cascada basados en características de Haar.

Eextienda lo mismo para la detección de ojos, etc.

6.1.2.- Conceptos básicos

La detección de objetos mediante clasificadores en cascada basados en características de Haar es un método eficaz propuesto por Paul Viola y Michael Jones en el artículo de 2001, "Rapid Object Detection using a Boosted Cascade of Simple Features". Es un enfoque basado en el aprendizaje automático en el que se entrena una función en cascada a partir de muchas imágenes positivas y negativas. Luego se usa para detectar objetos en otras imágenes.

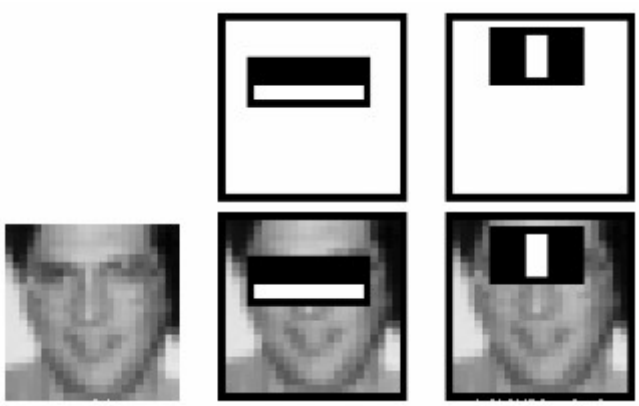
Aquí trabajaremos con la detección de rostros. Inicialmente, el algoritmo necesita muchas imágenes positivas (imágenes de rostros) e imágenes negativas (imágenes sin rostros) para entrenar al clasificador. Luego necesitamos extraer características de él. Para esto, se utilizan las características de Haar que se muestran en la imagen a continuación. Son como nuestro kernel convolucional. Cada característica es un valor único obtenido al restar la suma de píxeles debajo del rectángulo blanco de la suma de píxeles debajo del rectángulo negro. (BLANCO – NEGRO)



Ahora todos los tamaños y ubicaciones posibles de cada kernel se utilizan para calcular muchas características. Para cada cálculo de características, necesitamos encontrar la suma de los píxeles debajo de los rectángulos blanco y negro. Para solucionar esto, introdujeron las imágenes integrales. Simplifica el cálculo de la suma de los píxeles. Por grande que puede ser el número de píxeles, la la operación utilizando imágenes integrales involucra solo cuatro píxeles.

Pero entre todas estas características que calculamos, la mayoría de ellas son irrelevantes. Por ejemplo, considere la imagen de abajo. La fila superior muestra dos buenas características. La primera característica seleccionada parece centrarse en la propiedad de que la región de los ojos

suele ser más oscura que la región de la nariz y las mejillas. La segunda característica seleccionada se basa en la propiedad de que los ojos son más oscuros que el puente de la nariz. Pero las mismas ventanas aplicadas en las mejillas o en cualquier otro lugar son irrelevantes. Entonces, ¿cómo seleccionamos las mejores características de 160000 características? Lo consigue Adaboost.



Para ello, aplicamos todas y cada una de las características en todas las imágenes de entrenamiento. Para cada característica, encuentramos el mejor umbral que clasificará las caras en positivas y negativas. Pero obviamente, habrá errores o clasificaciones erróneas. Seleccionamos las características con la tasa de error mínima, lo que significa que son las características que mejor clasifican las imágenes de rostros y no rostros. (El proceso no es tan simple como esto. A cada imagen se le da el mismo peso al principio. Después de cada clasificación, se aumentan los pesos de las imágenes mal clasificadas. Luego, nuevamente se realiza el mismo proceso. Se calculan nuevas tasas de error. También nuevos pesos. El el proceso continúa hasta que se logra la precisión requerida o la tasa de error o se encuentra el número requerido de características).

El clasificador final es una suma ponderada de estos clasificadores débiles. Se llama débil porque por sí solo no puede clasificar la imagen, pero junto con otros forma un clasificador fuerte. El documento dice que incluso 200 características brindan una detección con una precisión del 95%. Su configuración final tenía alrededor de 6000 características. (Imagínese una reducción de más de 160000 características a 6000 características. Esa es una gran ganancia).

Así que ahora tomas una imagen. Tome cada ventana de 24x24. Aplicarle 6000 características. Comprueba si es cara o no. Wow... Wow... ¿No es un poco ineficiente y consume mucho tiempo? Sí, lo es. Los autores tienen una buena solución para eso.

En una imagen, la mayor parte de la región de la imagen es una región sin rostro. Por lo tanto, es una mejor idea tener un método simple para verificar si una ventana no es una región de la cara. Si no es así, deséchalo de una sola vez. No vuelva a procesarlo. En su lugar, concéntrese en la región donde puede haber una cara. De esta manera, podemos tener más tiempo para comprobar una posible región de la cara.

Para ello introdujeron el concepto de Cascada de Clasificadores. En lugar de aplicar todas las 6000 funciones en una ventana, se agrupan las funciones en diferentes etapas de clasificadores y se aplíquelan una por una. (Normalmente, las primeras etapas contendrán una cantidad muy inferior de características). Si una ventana falla en la primera etapa, se desecha. No consideramos las

características restantes en él. Si pasa, aplique la segunda etapa de características y continúe el proceso. La ventana que pasa por todas las etapas es un rostro. ¡¡¡Que tal el plan!!!

El detector de los autores tenía 6000 características con 38 etapas con 1, 10, 25, 25 y 50 características en las primeras cinco etapas. (Dos características en la imagen de arriba se obtienen en realidad como las dos mejores características de Adaboost). Según los autores, en promedio, se evalúan 10 características de 6000 por subventana. Así que esta es una explicación simple e intuitiva de cómo funciona la detección de rostros de Viola-Jones. Lea el documento para obtener más detalles.

6.1.3.- Detección de cascada Haar en OpenCV

Aquí nos ocuparemos de la detección. OpenCV ya contiene muchos clasificadores preentrenados para cara, ojos, sonrisa, etc. Esos archivos XML se almacenan en la carpeta **opencv/data/haarcascades/**. Vamos a crear un detector de cara y ojos con OpenCV.

Utilizamos la función:

```
detectMultiScale (image, objects, scaleFactor = 1.1,
                  minNeighbors = 3, flags = 0,
                  minSize = new cv.Size(0, 0),
                  maxSize = new cv.Size(0, 0))
```

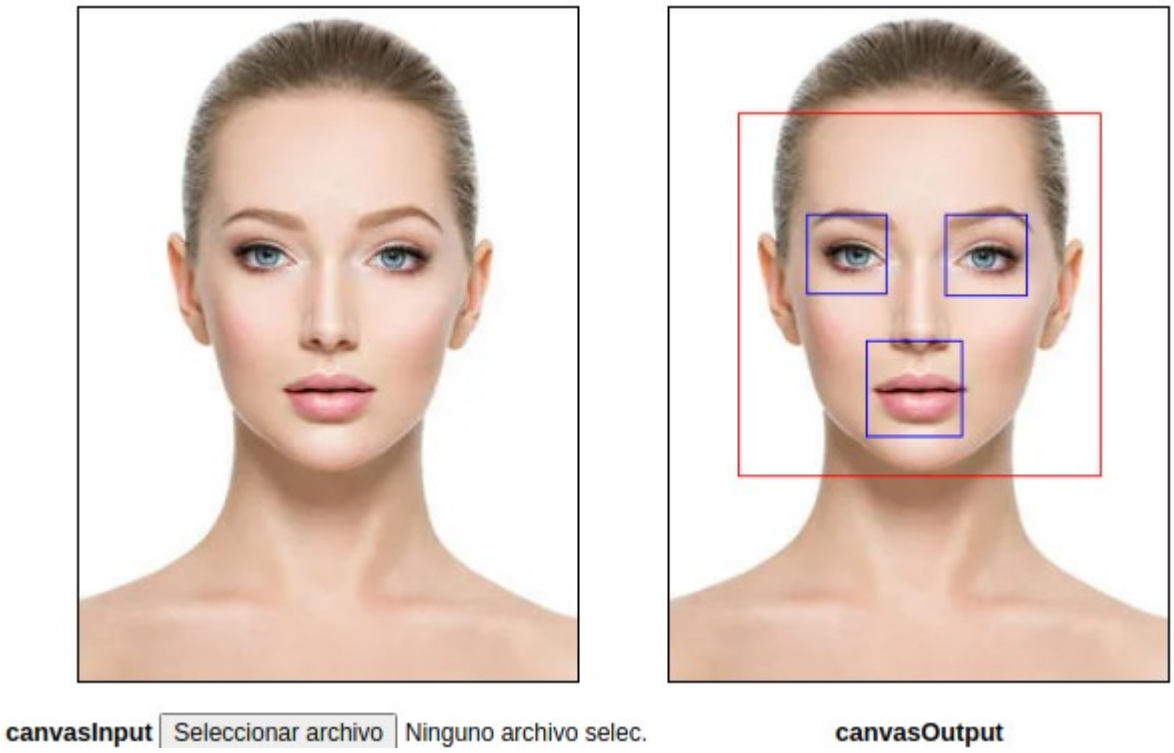
Parámetros:

image	matriz del tipo CV_8U que contiene una imagen donde se detectan objetos.
objects	vector de rectángulos donde cada rectángulo contiene el objeto detectado. Los rectángulos pueden estar parcialmente fuera de la imagen original.
scaleFactor	parámetro que especifica cuánto se reduce el tamaño de la imagen en cada escala de imagen.
minNeighbors	parámetro que especifica cuántos vecinos debe tener cada rectángulo candidato para conservarlo.
flags	parámetro con el mismo significado para una cascada antigua que en la función cvHaarDetectObjects. No se utiliza para una nueva cascada.
minSize	tamaño mínimo posible del objeto. Los objetos más pequeños que esto se ignoran.
maxSize	tamaño máximo posible del objeto. Los objetos más grandes que esto se ignoran. Si maxSize == minSize, el modelo se evalúa en una sola escala

El siguiente programa permite detectar rostros, ojos y boca en imágenes de caras.

6011_deteccion_rostro_haar_cascades.html

Resultados



6.2.- Detección de rostros en captura de vídeos

La parte teórica en este caso es la misma que en el apartado anterior salvo que se aplica fotograma a fotograma en el streming de la cámara web incorporada al ordenador.

El siguiente programa de OpenCV.js detecta el rostro sobre el vídeo capturado por la cámara.

6021_deteccion_rostro_camara.html

7.- Redes Neuronales Profundas (módulo dnn)

Estos tutoriales muestran cómo usar el módulo dnn en JavaScript.

El tratamiento de este tema por parte OpenCV es muy superficial a nivel teórico y se basa en mostrar resultados de su librería OpenCV con modelos ya entrenados en otros frameworks. No estoy totalmente seguro de esto, puede que si lo analizamos bien igual nos damos una sorpresa.

No obstante no vamos a abordar ahoar la tarducción porque el contenido de inglés no es muy grande y los interesados en esta parte del tutorial y que además no conozcan el inglés, cosa rara, pueden encontrar una ayuda definitivamente acertada y práctica en "**Google Translator**."

8.- Anexo I: Listado de la librería utils.js

```
function Utils(errorOutputId) { // eslint-disable-line no-unused-vars
    let self = this;
    this.errorOutput = document.getElementById(errorOutputId);

    const OPENCV_URL = 'assets/opencv.js';
    this.loadOpenCv = function(onloadCallback) {
        let script = document.createElement('script');
        script.setAttribute('async', '');
        script.setAttribute('type', 'text/javascript');
        script.addEventListener('load', async () => {
            if (cv.getBuildInformation)
            {
                console.log(cv.getBuildInformation());
                onloadCallback();
            }
            else
            {
                // WASM
                if (cv instanceof Promise) {
                    cv = await cv;
                    console.log(cv.getBuildInformation());
                    onloadCallback();
                } else {
                    cv['onRuntimeInitialized']=()=>{
                        console.log(cv.getBuildInformation());
                        onloadCallback();
                    }
                }
            }
        });
        script.addEventListener('error', () => {
            self.printError('Failed to load ' + OPENCV_URL);
        });
        script.src = OPENCV_URL;
        let node = document.getElementsByTagName('script')[0];
        node.parentNode.insertBefore(script, node);
    };

    this.createFileFromUrl = function(path, url, callback) {
        let request = new XMLHttpRequest();
        request.open('GET', url, true);
        request.responseType = 'arraybuffer';
        request.onload = function(ev) {
            if (request.readyState === 4) {
                if (request.status === 200) {
                    let data = new Uint8Array(request.response);
                    cv.FS_createDataFile('/', path, data, true, false,
false);
                    callback();
```

```
                } else {
                    self.printError('Failed to load ' + url + ' status: ' +
request.status);
                }
            }
        };
        request.send();
    };

    this.loadImageToCanvas = function(url, cavansId) {
        let canvas = document.getElementById(cavansId);
        let ctx = canvas.getContext('2d');
        let img = new Image();
        img.crossOrigin = 'anonymous';
        img.onload = function() {
            canvas.width = img.width;
            canvas.height = img.height;
            ctx.drawImage(img, 0, 0, img.width, img.height);
        };
        img.src = url;
    };

    this.executeCode = function(textAreaId) {
        try {
            this.clearError();
            let code = document.getElementById(textAreaId).value;
            eval(code);
        } catch (err) {
            this.printError(err);
        }
    };

    this.clearError = function() {
        this.errorOutput.innerHTML = '';
    };

    this.printError = function(err) {
        if (typeof err === 'undefined') {
            err = '';
        } else if (typeof err === 'number') {
            if (!isNaN(err)) {
                if (typeof cv !== 'undefined') {
                    err = 'Exception: ' + cv.exceptionFromPtr(err).msg;
                }
            }
        } else if (typeof err === 'string') {
            let ptr = Number(err.split(' ')[0]);
            if (!isNaN(ptr)) {
                if (typeof cv !== 'undefined') {
                    err = 'Exception: ' + cv.exceptionFromPtr(ptr).msg;
                }
            }
        } else if (err instanceof Error) {
            err = err.stack.replace(/\n/g, '<br>');
        }
        this.errorOutput.innerHTML = err;
    };

    this.loadCode = function(scriptId, textAreaId) {
```

```
        let scriptNode = document.getElementById(scriptId);
        let textArea = document.getElementById(textAreaId);
        if (scriptNode.type !== 'text/code-snippet') {
            throw Error('Unknown code snippet type');
        }
        textArea.value = scriptNode.text.replace(/^\n/, '');
    };

    this.addFileInputHandler = function(fileInputId, canvasId) {
        let inputElement = document.getElementById(fileInputId);
        inputElement.addEventListener('change', (e) => {
            let files = e.target.files;
            if (files.length > 0) {
                let imgUrl = URL.createObjectURL(files[0]);
                self.loadImageToCanvas(imgUrl, canvasId);
            }
        }, false);
    };

    function onVideoCanPlay() {
        if (self.onCameraStartedCallback) {
            self.onCameraStartedCallback(self.stream, self.video);
        }
    };

    this.startCamera = function(resolution, callback, videoId) {
        const constraints = {
            'qvga': {width: {exact: 320}, height: {exact: 240}},
            'vga': {width: {exact: 640}, height: {exact: 480}}};
        let video = document.getElementById(videoId);
        if (!video) {
            video = document.createElement('video');
        }

        let videoConstraint = constraints[resolution];
        if (!videoConstraint) {
            videoConstraint = true;
        }

        navigator.mediaDevices.getUserMedia({video: videoConstraint, audio:
false})
            .then(function(stream) {
                video.srcObject = stream;
                video.play();
                self.video = video;
                self.stream = stream;
                self.onCameraStartedCallback = callback;
                video.addEventListener('canplay', onVideoCanPlay, false);
            })
            .catch(function(err) {
                self.printError('Camera Error: ' + err.name + ' ' +
err.message);
            });
    };

    this.stopCamera = function() {
        if (this.video) {
            this.video.pause();
            this.video.srcObject = null;
```

```
            this.video.removeEventListener('canplay', onVideoCanPlay);
        }
        if (this.stream) {
            this.stream.getVideoTracks()[0].stop();
        }
    };
};
```

utils.js