|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | | **INFORME DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA ESTUDIANTES** | |
|  | | | | |
| **CARRERA**: COMPUTACIÓN/INGENIERÍA DE SISTEMAS | | | | **ASIGNATURA**: INTELIGENCIA ARTIFICIAL II |
| **NRO. PRÁCTICA**: | 2-1 | **TÍTULO PRÁCTICA**: Reconocimiento de formas en base a detección de bordes, binarización por umbral, aplicación de filtros y operaciones morfológicas. | | |
| **OBJETIVO ALCANZADO:** Reforzar los conocimientos adquiridos en clase sobre la aplicación de filtros para reducción de ruido y kernels para  la detección de bordes, operaciones morfológicas y normalización del histograma. | | | | |
| **ACTIVIDADES DESARROLLADAS** | | | | |
| **Desarrollar un programa que permita generar ruido de sal y pimienta y aplicar filtros para reducir dicho ruido** | | | | |
| 1. **Programar un método que genere un porcentaje de ruido de sal o pimienta en un video, considerando las dimensiones de este. Se deberá poder ingresar un porcentaje de ruido a través de dos trackbars (uno para sal y otro para pimienta).** | | | | |
| * 1. **Generemos una clase con la que trabajaremos esta primera parte**   /\*  \* @name: no necesaria  \* Metodos de procesamiento de imagne  \*/ class Procesamiento { public:  String name;   void tamanio\_imagen();   Mat sal\_o\_pimineta(Mat, int, bool);   Mat to\_gris(Mat);   Mat gaussian\_blur(Mat, int);   Mat median\_brur(Mat, int);   Mat gxgy(Mat);   Mat canny(Mat, int, int);   Procesamiento(string);  };   * 1. **Leemos variables e iniciamos variables para la parte uno**   int main(int *argc*, char \**argv*[]) {  VideoCapture video("../video.mp4");  if (video.isOpened()) {  Mat img;  Mat gris;  namedWindow("Original", **WINDOW\_AUTOSIZE**);  namedWindow("Gris", **WINDOW\_AUTOSIZE**);  namedWindow("Sal", **WINDOW\_AUTOSIZE**);  namedWindow("Pimiento", **WINDOW\_AUTOSIZE**);  namedWindow("MedianBlur", **WINDOW\_AUTOSIZE**);  namedWindow("Gaussian Blur", **WINDOW\_AUTOSIZE**);  namedWindow("Sovel GX - GY", **WINDOW\_AUTOSIZE**);  namedWindow("Canny", **WINDOW\_AUTOSIZE**);  while (3 == 3) {  video >> img;  resize(img, img, Size(), 0.8, 0.8);   * 1. **Método para para el cálculo de la método de sal o pimienta**   /\*  \* Metodo para calcular un porcentaje de sal o pimienta  \* @img: Imagen de entrada  \* @k: porcentaje de procesamienta  \* @type: true = sal | false = pimienta  \* pasamos @k de [0-100] a [0-1]  \* calculamos el porcentaje en base a columnas y filas  \* recorremos el numero de pixeles a cambia: true: pixel blanco | false: pixel negro  \*   \* @return: imagen procesado  \*/ Mat Procesamiento::sal\_o\_pimineta(Mat *img*, int *k*, bool *type*) {  double t = (*k* < 10) ? stod("0.0" + to\_string(*k*)) :  stod("0." + to\_string(*k*));  t = (*k* == 100) ? 1 : t;  Mat img\_trat = *img*.clone();  int total = (int) (t \* img\_trat.rows \* img\_trat.cols);  int cont = 0, fila = 0, col = 0;  while (cont < total) {  fila = rand() % img\_trat.rows;  col = rand() % img\_trat.cols;  cont++;  img\_trat.at<uchar>(fila, col) = (*type* == 0) ? 255 : 0;  }  return img\_trat; }   * 1. **Método para convertir a gris**   /\*  \* Convertir imagen de color a escala de grises  \* @imagen de entrdad  \*  \* @return imagen procesada en escala de grises  \*/ Mat Procesamiento::to\_gris(Mat *img*) {  Mat gris;  cvtColor(*img*, gris, **COLOR\_BGR2GRAY**);  return gris; }   * 1. **Inicializamos método y mostramos en pantalla**   //Iniciamos clase Procesamiento procesamiento(""); //Convertmos imaagen a gris gris = procesamiento.to\_gris(img); //metodo con false para sal Mat sal = procesamiento.sal\_o\_pimineta(gris, **val\_sal**, false); //metodo con true para pimienta  ...  .  .  ...  imshow("Original", img);  //agregamos las barras de estado createTrackbar("Sal", "Original", &**val\_sal**, 100, functionTrackbar, nullptr); createTrackbar("Pimiento", "Original", &**val\_pim**, 100, functionTrackbar, nullptr);  imshow("Gris", gris); imshow("Sal", sal); imshow("Pimiento", pim);   * 1. **Anexos** | | | | |
| 1. Programar una función para aplicar los siguientes filtros: | | | | |
| * 1. **Median Blur**      1. **Método para convertir una imagen a median blur**   /\*  \* Usando medianblur procesamos el filtro de una imagen  \* @img: imagen a procesar  \* @k: valor del kernel  \*/ Mat Procesamiento::median\_brur(Mat *im*, int *k*) {  *k* = (*k* % 2 != 1) ? *k* - 1 : *k*;  *k* = (*k* < 1) ? 1 : *k*;   Mat img;  medianBlur(*im*, img, *k*);  return img;  }   * + 1. **Método Gaussian blur**   /\*  \* usamos gausian blur  \* @img: Imagen a procesar  \* @k: valor del kerne  \*   \* validamos que k sea impar  \* validamos que k sea mayor a 0  \*/ Mat Procesamiento::gaussian\_blur(Mat *im*, int *k*) {  *k* = (*k* % 2 != 1) ? *k* - 1 : *k*;  *k* = (*k* < 1) ? 1 : *k*;  Mat img;  GaussianBlur(*im*, img, Size(*k*, *k*), 2, 2);  return img;  }   * + 1. **Inicializamos métodos y track bar**   //metodo de medain blue Mat media = procesamiento.median\_brur(gris, **val\_med**); //Metodo de gaunciablur Mat gausi = procesamiento.gaussian\_blur(gris, **val\_gau**);  ...  .  .  ...  createTrackbar("MedianBLur", "Original", &**val\_med**, 100, functionTrackbar, nullptr); createTrackbar("Gaussian Blur", "Original", &**val\_gau**, 100, functionTrackbar, nullptr);   * + 1. **Anexos** | | | | |
|  | | | | |
| 1. **Compare los resultados obtenidos por cada filtro, y reflexione cuál ha obtenido mejores resultados.**  * **Con el suavizado por median blur obtenemos una imagen que con un kernel pasando de 40 se pierden los bordes o llegan a ser generalizados como se muestra en la siguiente imagen.**     **Se necesita un kernel menor a 30 para que exista una distinción de objetos**     * **Con el suavizado por Gaussian blur se obtiene imágenes más suavizado más limpio y a criterio personal es mejor.**     **Se llega a distinguir los objetos de forma más visible con un kernel alto, ayudando esto a la distinción de bordes como se mostrará a continuación.** | | | | |
| 1. Aplicar al menos 2 algoritmos de detección de bordes y comparar los resultados de usar o no filtros de suavizado.    1. **Usaremos sobel en base a X y Y mostrando una imagen mostrada en base al procesamiento de las dos imágenes**   /\*  \* Img >> sobel uniendo X y Y  \* @img: Imagen a procesar  \* procesmos imagen con x  \* procesmos imagen con y  \* unimos las imagenes en x y y  \* @return: imagen procesado  \*/ Mat Procesamiento::gxgy(Mat *img*) {  Mat gX, gY;  Mat gXAbs, gYAbs;  Mat sobelBordes;  Sobel(*img*, gX, CV\_16S, 1, 0, 3);  Sobel(*img*, gY, CV\_16S, 0, 1, 3);  convertScaleAbs(gX, gXAbs);  convertScaleAbs(gY, gYAbs);  addWeighted(gXAbs, 0.5, gYAbs, 0.5, 0, sobelBordes);  return sobelBordes; }   * 1. **Usamos canny con un track bar para el umbral y radio**   /\*  \* Procesamiento con canny   \* @img: imagen a procesar   \* @return imagen procesada  \*/ Mat Procesamiento::canny(Mat *img*, int *umbral*, int *radio*) {  Mat can;  Canny(*img*, can, *umbral*, *umbral* \* (double) *radio*, 3);  return can; }   * 1. **Inicializamos métodos y track bar para mostrar en pantalla**   //Usamos suavizado medianblur para una mejor detecion de bordes //bordes Sobel de x y y Mat gxgy = procesamiento.gxgy(media); //bordes con canny Mat can = procesamiento.canny(media, **umbral**, **radio**);  ...  .  .  ...  createTrackbar("Umbral", "Original", &**umbral**, 100, functionTrackbar, nullptr); createTrackbar("Radio", "Original", &**radio**, 100, functionTrackbar, nullptr);  imshow("Sovel GX - GY", gxgy); imshow("Canny", can);   * 1. **Anexos** | | | | |
| **PARTE 2 Desarrollar un programa que permita aplicar operaciones morfológicas para mejorar la calidad de imágenes médicas, para ello deberá realizar las siguientes tareas:** | | | | |
| 1. Seleccionar 3 imágenes médicas a las que se les aplicarán las operaciones morfológicas. Las imágenes deben estar en escala de grises y deben corresponder a radiografías, angiografías, TACs, etc. | | | | |
| 1. **Erosión**   Cabe destacar que existen tres tipos de generar los puntos de erosión con **MORPH\_RECT** que se van generando cuadros por pixel o tamaño, **MORPH\_CROSS** genera (+) cruces, mientras **MORPH\_ELLIPSE** elipses o círculos.   * 1. Usamos el siguiente método el track bar de la imagen original.   void Erosion(int, void \*) {  Mat element = getStructuringElement(MORPH\_ELLIPSE, //MORPH\_RECT [] MORPH\_CROSS + MORPH\_ELLIPSE O  Size(2 \* erosion\_size + 1, 2 \* erosion\_size + 1),  Point(0, 0));  erode(imagen1, element, element);  imshow("Erosion", element); }   * 1. Cargamos imagen y agregamos el track bar   **imagen1** = imread("../radio1.jpg", **IMREAD\_COLOR**); resize(**imagen1**, **imagen1**, Size(), 0.4, 0.4); imshow("Radiografia 1", **imagen1**); createTrackbar("Eroicon", "Radiografia 1", &**erosion\_size**, 37, Erosion, nullptr); Erosion(0, 0);   * 1. Anexos * **MORPH\_RECT**      * **MORPH\_CROSS**      * **MORPH\_ELLIPSE** | | | | |
| 1. **Dilatación**    1. Método del track bar   void Dilation(int, void \*) {  Mat element = getStructuringElement(**MORPH\_CROSS**, //MORPH\_RECT [] MORPH\_CROSS + MORPH\_ELLIPSE O  Size(2 \* **dilation\_size** + 1, 2 \* **dilation\_size** + 1),  Point(0, 0));  dilate(**imagen2**, element, element);  imshow("Dilation", element); }   * 1. Inicialización de las imágenes y track bar   namedWindow("Radiografia 2", **WINDOW\_AUTOSIZE**); namedWindow("Dilation", **WINDOW\_AUTOSIZE**); **imagen2** = imread("../radio2.jpg", **IMREAD\_COLOR**); resize(**imagen2**, **imagen2**, Size(), 0.22, 0.22); imshow("Radiografia 2", **imagen2**); createTrackbar("Dilation", "Radiografia 2", &**dilation\_size**, 37, Dilation, nullptr); Dilation(0, nullptr);   * 1. Anexos * **MORPH\_RECT**      * **MORPH\_CROSS**      * **MORPH\_ELLIPSE** | | | | |
| 1. **Top Hat y Black Hat**   Para el tratamiento de imágenes, la documentación nos recomienda usar el método **morphologyEx** que si recibe como en **MORPH\_TOPHAT** como parámetro el 5 es para **Top Hat** y el 6 es para **Black Hat***.*   * 1. método llamado en el track bar con los parámetros para **TOP HAT.**   void Morphology\_Operations(int, void \*) {  /\*  \* Top Hat: operation: 5  \* Black Hat: operation: 6  \*/  int operation = 5;  Mat element = getStructuringElement(**MORPH\_ELLIPSE**, //MORPH\_RECT [] MORPH\_CROSS + MORPH\_ELLIPSE O  Size(2 \* **morph\_size** + 1, 2 \* **morph\_size** + 1), Point(**morph\_size**, **morph\_size**));  morphologyEx(**imagen3**, element, operation, element);  imshow("Top Hat", element); }   * 1. método llamado en el track bar con los parámetros para **Black HAT**.   void Morphology\_Operations(int, void \*) {  /\*  \* Top Hat: operation: 5  \* Black Hat: operation: 6  \*/  int operation = 6;  Mat element = getStructuringElement(**MORPH\_ELLIPSE**, //MORPH\_RECT [] MORPH\_CROSS + MORPH\_ELLIPSE O  Size(2 \* **morph\_size** + 1, 2 \* **morph\_size** + 1), Point(**morph\_size**, **morph\_size**));  morphologyEx(**imagen3**, element, operation, element);  imshow("Top Hat", element); }   * 1. **Anexos TOP HAT**        * 1. **Anexos BLACK HAT** | | | | |
| 1. Imagen Original + (Top Hat – Black Hat)    1. Usamos el siguiente método que será llamo el los metodos del track bar de Top Hat y Black Hat   /\*  \*Mato para la suma y resta de imagenes  \* @img1: primera imagen a restar  \* @img2: Segunda imagen a restar  \* @img3: imagen que va a ser sumada con el Residuo de la @img1 - @img2  \* @name: Nombre de la vantana a mostrar la operacion realizada  \*/ void suma\_rest(Mat *img1*, Mat *img2*, Mat *img3*, String *name* = "Suma | Resta") {  Mat resta;  Mat suma;  absdiff(*img1*, *img2*, resta);//Resta  addWeighted(*img3*, 0.5, resta, 0.5, 0, suma);//suma  imshow(*name*, suma); }   * 1. Anexos | | | | |
| **CONCLUSIONES**:   * El tratamiento de imágenes nos ayuda a identificar objetos que a simple vista no se puede identificar tal como lo podemos hacer con top hat y el Black hat ya que con estos métodos podemos jugar con los colores de los píxeles pasando les blanco o negro usado una foma de cruz, cuadro o circulo. * Por otra parte, para el tratamiento de imágenes con la detección de bordes es mejor si lo mejor hacemos con un método de suavizado como el Canny o el Gaussian Blur ayudándonos a detectar mejor los bordes de un objeto en imagen o en el frame de un video. | | | | |
| **RECOMENDACIONES:**   * Recomendamos usar imágenes a escala de grises ya que así se podemos identificar mejor el rango de colores y los metodos ejemplificados en este informe se son más eficientes con imagen en esta escala de colores. * Leer la documentación oficial de OpenCV ya que es muy completa y ofrece ejemplos prácticos y rápidos usando datos e imágenes de que se encuentran en la carpeta del OpenCV. | | | | |
| **URL GIT:**  [**https://github.com/RicardoVinicioJara/OpenCV/tree/main/Guia-Practicas-UNIDAD-3**](https://github.com/RicardoVinicioJara/OpenCV/tree/main/Guia-Practicas-UNIDAD-3) | | | | |

***Nombre de estudiante*: Ricardo Vinicio Jara Jara**

***Firma de estudiante*: **