

# Programación Móvil

Dr. Marco Aurelio Nuño Maganda

Universidad Politecnica de Victoria  
Ingeniería en Tecnologías de la Información  
Cuatrimestre Septiembre - Diciembre 2024

*[mnunom@upv.edu.mx](mailto:mnunom@upv.edu.mx)*

August 29, 2024

# Breve CV del Facilitador

- Doctor en Ciencias Computacionales por parte del INAOE (2009).
- Profesor de Tiempo Completo de la UPV desde 2009.
- Miembro del Sistema Nacional de Investigadores - Nivel Candidado (2014-2016), Nivel I (2020-2022), Nivel I (2023-2027)
- 17 tesis dirigidas a nivel maestría.
- Asignaturas impartidas en el pasado
  - Licenciatura: Cómputo en Dispositivos Mviles, Graficación por Computadora Avanzada, Lenguajes y Automátas, Programación Orientada a Objetos
  - Maestría: Visión por computadora, Tópicos Selectos de Imagenología, Fundamentos de Sistemas de Información
- Miembro del Núcleo Académico Básico (NAB) de la maestria en Ingeniería de la UPV.

# Plataforma Virtual para el Curso

- Nombre de la clase: **Programación Móvil-Septiembre - Diciembre 2024**
- Código de clase en Classroom: **xv543pb**
- Enlace Meet para sesiones no presenciales:  
**<https://meet.google.com/jsb-nmxy-dht>**

# Horario de la Clase

## ■ Días y horas de clase

Clave de Grupo	Lunes	Martes	Miércoles	Jueves	Viernes
ITI-271311	14:00- 14:54	8:50- 10:40	14:00-14:54	9:45 - 10:40	12:05 - 13:00
ITI-271304		11:10 -13:00	9:45-12:05	11:10 -13:00	

## ■ Fechas Importantes:

- Inicio de Cursos: 2/Septiembre
- Fin de Cursos: 13/Diciembre
- Dias no hábiles oficiales: 16 de septiembre (lunes), 1 de octubre (martes) y 18 de noviembre (lunes).

# Plataforma Virtual para el Curso

- Nombre de la clase: **Programación Móvil- Septiembre - Diciembre 2024**
- Código de clase en Classroom: **xv543pb**
- Enlace Meet para sesiones no presenciales:  
**<https://meet.google.com/jsb-nmxy-dht>**

# Reglas básicas

- Se recomienda puntualidad y asistencia a las sesiones.
- Respeto hacia el profesor y hacia sus compañeros y compañeras.
- No se permite el ingreso y/o ingestión de **Alimentos** ni **Bebidas** de ningún tipo a la clase.
- Solo se puede usar **AUDIFONOS O DISPOSITIVOS MANO-LIBRES EN CLASE** previa autorización por parte del profesor. Cualquier uso no AUTORIZADO es motivo de amonestación al estudiante, y expulsión en caso de reincidir sin derecho a réplica.

# Uso del Teléfono Inteligente

- Se recomienda no utilizarlo durante el transcurso de la clase. Depende del comportamiento del grupo que esto no sea aplicado...

## Resguardo del teléfono inteligente

De ser necesario, se solicitará al INICIO de la CLASE a todos los asistentes a la clase (incluyendo al profesor) guardar su telefono en una caja, la cual será cerrada, regresando su telefono al finalizar la SESION.



# Pase de Lista

- Se pasa lista al inicio de la clase. En caso de reincorporación tardía, se pone un retardo.
- DOS RETARDOS equivalen a una INASISTENCIA, que no es JUSTIFICABLE.



# Justificacion de Inasistencia (1)

Para justificar una inasistencia, es necesario cumplir con los siguientes pasos:

- Ingresar al apartado de Google classroom creado para dicho fin y cargar un archivo PDF para cad dia (o lapso) de inasistencia que desee reportar.
- Formato del nombre del PDF (todo en minusculas):

iti-<clavegrupo>-<apat>-<amat>-<nombre>-<dd1>-<mm1>-<yyyy1>-<dd2>-<mm2>-<yyyy2>.pdf

- Donde (Si pone otros datos, dicha justificacion queda anulada):
  - <clavegrupo> es la clave que viene en su horario.
  - <apat>-<amat>-<nombre> son sus nombres. Debe utilizar guiones bajo para sustituir los espacios en su datos (ver ejemplo).
  - <dd1>-<mm1>-<yyyy1> es la fecha de inicio de inasistencia
  - <dd2>-<mm2>-<yyyy2> es la fecha de fin de inasistencia (si es faltó solo día, use la misma 02-09-2024-02-09-2024)

- Ejemplos iti-552244-nuño-maganda-marco\_aurelio-02-09-2024-02-09-2024.pdf

iti-001233-del\_sagrado\_corazon-hernandez-michael\_jackson-02-09-2024-02-10-2024.pdf

## Justificacion de Inasistencia (2)

- El interior de ese PDF debe contener documento que justifique su inasistencia.  
**Solo se reciben inasistencias por motivos medicos y de trabajo (SAT, Pasaporte, VISA, entrevista de trabajo)).** Los motivos meramente personales quedan cubiertos por el 20% de faltas que se conviene para el estudiante.

# Alumnos con Empleo (1)

- Al NO alcanzar un 80% de asistencia, el estudiante pierde su derecho de ser EVALUADO

## Alumnos VIPs

En caso de tener un empleo formal dentro o fuera de la ciudad, es necesario entregar una **constancia laboral** que acredite el horario que se esta cubriendo (en el caso de locales, este horario se debe empalmar con el de la materia). En esa constancia debe acreditar que se esta haciendo labores de manera presencial en tal ubicacion. Esto lo dispensa solo del requisito de las asistencias, mas no de los proyectos que deban entregarse. Incluso pudiera solicitarle presentar avance de manera “remota” durante alguna de las clases. Enviar esa constancia con copia para el director de carrera.

## Alumnos con Empleo (2)

- La justificación de inasistencias por *actividad laboral* se considerará a partir del momento de la recepción de dicha constancia en el correo del instructor (y no a partir de la fecha indicada en la constancia), por lo que si se recibe de manera tardía (con mas de una semana de retardo), dichas inasistencias NO SERAN justificadas.
- La justificación será válida si el estudiante programa **POR LO MENOS** dos asesorías por semana. De no hacerlo, pierde el beneficio de la justificación y se aplican las reglas anteriormente establecidas.

- 1 Introducción a la Inteligencia Artificial
  - 1 Inteligencia Artificial y Sistemas Inteligentes
  - 2 Razonamiento y representación del conocimiento
  - 3 Estrategias de Búsqueda
- 2 Técnicas Básicas de Aprendizaje
  - 1 Aprendizaje Supervisado y No Supervisado
  - 2 Aprendizaje basado en Redes Neuronales
  - 3 Algoritmos Genéticos
- 3 Introducción a la percepción automática
  - 1 Percepción Visual
  - 2 Percepción Visual Estereoscópica
  - 3 Procesamiento de Lenguaje Natural

# Evaluación (1)

- Para cada unidad del curso, se consideran 3 aspectos:
  - Ejercicios o investigaciones especiales (1)- 25%
  - Proyecto Individual - 35%
  - Proyecto en Equipo - 40%
- Para aprobar el curso, es obligatorio:
  - Tener calificación aprobatoria en todas las unidades (100-100-40 no da calificación aprobatoria).
  - Tener por lo menos dos asesorías por semana (Registrarlas por semana, no 30 asesorías al final del cuatrimestre)
  - Cumplir con el 80% de asistencia mínimo, incluyendo aquellas inasistencias justificadas debidamente mediante el SIITA

## Evaluación (2)

Para cada unidad, habra sesiones de “teoria”, sesiones de seguimiento de proyectos y sesiones de esparcimiento

- En las sesiones de teoria, el profesor presentara uno o varios temas
- En las sesiones de seguimiento de proyectos, de manera aleatoria se nombrara al integrante de equipo individual o en equipo. En el caso de que un integrante individual no responda, se le bajarán 5 puntos a su calificación del proyecto
- En las sesiones de esparcimiento, se permitirá a los estudiantes trabajar en proyectos pendientes, pero se contabilizará la asistencia.

# Evaluación (3)

## Sesiones de Seguimiento de proyectos

- En el caso de que el integrante del equipo seleccionado aleatoriamente no responda satisfactoriamente lo cuestionado, se le bajaran 5 puntos a su calificación del proyecto a todos los integrantes del equipo
- En el caso de los proyectos en equipo, el integrante seleccionado es aleatorio. Si en una primera ronda le toco al integrante A, en una segunda ronda posiblemente le toque al integrante B



# Evaluación (4)

Lo que se debe presentar en una sesion de seguimiento de proyectos

- En un trabajo individual
  - Compartir pantalla de la ejecucion del avance del proyecto
  - Explicar con recursos multimedia los pasos para la resolucion del proyecto
  - Establecer el avance desde la ultima entrega
- En un trabajo grupal
  - Compartir pantalla de la ejecucion del avance del proyecto
  - Explicar con recursos multimedia los pasos para la resolucion del proyecto
  - Desglosar como se repartio el trabajo entre los integrantes del equipo
  - Establecer el avance desde la ultima entrega

# Evaluación (5)

Acerca de los proyectos

- Aleatorios y DIFERENTES para la mayoría (preferentemente para cada integrante)
- Equipos: Proyectos diferentes para cada equipo, e Integrantes de los mismos formados de manera ALEATORIA!!

# Fragmentación de equipos

- Si llegar a ocurrir que en un proyecto en equipo no hay un acuerdo para trabajar en equipo (Hay dos o mas entregas del proyecto asignado por partes diferentes dentro del mismo equipo)

## Penalización

Cada “fragmento” de equipo recibe una penalizacion de 25 puntos mas las penalizaciones acumuladas por otros rubros.

- esta regla **NO APLICA** cuando hay uno o varios “desertores” del equipo (y hay una sola entrega del proyectos en equipo)

# Acerca de Exención

- Cuando el profesor realiza alguna mecánica para exentar un proyecto (Individual/Equipo/Asignación especial) y uno o varios estudiantes completan lo solicitado, existen dos posibilidades:
  - El estudiante acepta exentar la elaboración de dicho proyecto o actividad, pero al hacer esto asume que la calificación asignada es 70.
  - El estudiante decide hacer el proyecto a pesar de haber exentado. En este caso el estudiante se hace acreedor a 20 puntos que puede aplicar sobre la calificación de dicho proyecto.

# Cartucho de Recuperación (REC)

- Estudiante tiene derecho a solicitar un ÚNICO proyecto de recuperación aplicable a un solo proyecto o actividad.
- Esta solicitud debe HACERLA es estudiante - El profesor NO ES RESPONSABLE de informar al estudiante cuando tiene un ADEUDO.
- Si el proyecto no entregado es individual, se asigna otro proyecto diferente.
- Si el proyecto es en equipo, de común acuerdo con los integrantes pueden trabajar en otro proyecto diferente en equipo, o recibir una asignación individual de un proyecto diferente.
- La calificación recuperada será asignada siempre y cuando cumpla con el porcentaje de falta mínimo necesario para aprobar. Además, debe haber agendado el % de asesorías proporcional al tiempo de cuatrimestre transcurrido.
- El nuevo proyecto asignado esta diseñado para que el estudiante invierta en él por lo menos 1 SEMANA. Si lo solicita un día antes de terminar el cuatrimestre, posiblemente no tendrá tiempo de llevarlo a cabo.

# Reporte Técnico de Desarrollo de Práctica

- Para cada práctica realizada, entregar un documento (**únicamente en formato PDF\***) con las siguientes secciones:
  - Introducción
  - Desarrollo Experimental
  - Resultados
  - Conclusiones
  - **Referencias**
- Para GENERAR este reporte es necesario utilizar la plantilla en LATEX (**únicamente usando LATEX\***) localizada en el siguiente enlace:  
<https://www.overleaf.com/read/dgkhvfwnygvc>

# Reporte Técnico de Desarrollo de Práctica

- Bajo ninguna circunstancia deben incluir **CÓDIGO FUENTE**. Si pueden incluir diagrama de flujo, Pseudocódigo, Diagrama E-R, Diagrama de Clases, de Casos de USO, etc. De incluir código fuente, solo tendrá un 50% del valor en la calificación.
- En caso de trabajos individuales o en EQUIPO, deben emplear la plantilla LaTeX que se provee. En caso de utilizar algo diferente a LaTeX u otra plantilla de LaTeX, la calificación proporcional del informe será **DESESTIMADA**.
- En caso de trabajos en equipo, se debe agregar los integrantes al inicio del INFORME. **El trabajo solo cuenta para aquellos integrantes mencionados en el informe (y que dicho nombre se encuentre registrado tal cual en la lista). Una vez ENTREGADO, si hay OMISIONES de los integrantes, no se realizará CORRECCION alguna, se debe asumir la consecuencias que esto conlleva.**

# Ponderación del Informe en la Calificación del Proyecto

- Informe: 34 Puntos
  - Uso adecuado de Latex: 5 Puntos
  - Organización y Redacción: 6 Puntos
  - Referencias en formato adecuado: 8 Puntos
  - Evidencia del trabajo realizado: 8 Puntos
  - Sin faltas de ortografía ni errores de dedo: 7 Puntos
- Proyecto: 66 Puntos
  - Ejecución y Funcionalidad: 45 Puntos
  - Modularidad: 13 Puntos
  - Documentación: 8 Puntos



# Entregables de proyecto individual (1)

- Crear un archivo ZIP con el siguiente formato de nombre:
  - **iti-271229\_uX\_nuno\_maganda\_marco\_aurelio**
- Dentro, debe contener lo siguiente:
  - **iti-271229\_uX\_nuno\_maganda\_marco\_aurelio\_source** (Carpeta con código fuente de la aplicación)
  - **iti-271229\_uX\_nuno\_maganda\_marco\_aurelio\_latex** (Carpeta con código fuente del informe)
  - **iti-271229\_uX\_nuno\_maganda\_marco\_aurelio.apk** (Instalable (solo si se trata de una aplicación móvil) )
  - **iti-271229\_uX\_nuno\_maganda\_marco\_aurelio.pdf** (Informe)
- Donde:
  - **X** es el número de unidad a un dígito (1, 2, etc)
  - **Sustituir con sus apellidos y nombres de manera apropiada**
- NO DEBE HABER otros archivos .ZIP dentro del ZIP PRINCIPAL

## Entregables de proyecto individual (2)

- En el caso que un proyecto individual sea asignado en equipo a varios estudiantes, el archivo entregable DEBE MANEJARSE como la de un proyecto individual
  - Solo un integrante del equipo carga en la plataforma el entregable individual.
  - El informe debe llevar los nombres de los integrantes del equipo que trabajaron (Si se omite a alguien, se asume que no trabajo en el proyecto).
  - NO ES NECESARIO que los otros integrantes marquen en el sistema la tarea como entregada, ya que se conoce su situación desde que se asigna el proyecto. El profesor ya sabe que ustedes van en equipo con el estudiante que hizo la entrega, y por eso deben asegurarse que en el informe entregado, vayan anotados sus nombres.

# Entregables de proyectos en equipo

- Crear un archivo ZIP con el siguiente formato de nombre:
  - **iti-271229\_eq\_NN\_uX**
- Dentro, debe contener lo siguiente:
  - **iti-271229\_eq\_NN\_uX\_source** (Carpeta con código fuente de la aplicación)
  - **iti-271229\_eq\_NN\_uX\_latex** (Carpeta con código fuente del informe)
  - **iti-271229\_eq\_NN\_uX.apk** (Instalable - Solo aplicaciones móviles)
  - **iti-271229\_eq\_NN\_uX.pdf** (Informe)

Donde:

- **NN** es el número de equipo a dos dígitos (01, 02, etc)
  - **X** es el número de unidad a un dígito (1, 2, etc)
- En cada entrega, **UN SOLO INTEGRANTE DEL EQUIPO** deberá cargar los archivos en el classroom.
- NO DEBE HABER otros archivos .ZIP dentro del ZIP PRINCIPAL

# Entregables de asignaciones especiales

- Crear un archivo ZIP con el siguiente formato de nombre:
  - **iti-271229\_aeX\_uY\_nuno\_maganda\_marco\_aurelio**
- Dentro, debe contener lo siguiente:
  - **iti-271229\_aeX\_uY\_nuno\_maganda\_marco\_aurelio\_source** (Carpeta con código fuente de la aplicación - Cuando aplique)
  - **iti-271229\_aeX\_uY\_nuno\_maganda\_marco\_aurelio\_latex** (Carpeta con código fuente del informe o diapositivas)
  - **iti-271229\_aeX\_uY\_nuno\_maganda\_marco\_aurelio.apk** (Instalable - Solo aplicaciones móviles, Cuando aplique)
  - **iti-271229\_aeX\_uY\_nuno\_maganda\_marco\_aurelio.pdf** (Informe)
- Donde:
  - **X** es el número de asignación dentro de la unidad a un dígito (1, 2, etc)
  - **Y** es el número de unidad a un dígito (1, 2, etc)
  - **Sustituir con sus apellidos y nombres de manera apropiada**
- NO DEBE HABER otros archivos .ZIP dentro del ZIP PRINCIPAL

# Nombres de Archivos Entregables

En el caso de nombres y apellidos acentuados, con diéresis o con virgulilla (~), sustituir de acuerdo con las siguientes reglas:

- Sustituir N/n por Ñ/ñ
- Sustituir A/a por Á/á
- Sustituir E/e por É/é
- Sustituir I/i por Í/í
- Sustituir O/o por Ó/ó
- Sustituir U/u por Ú/ú
- Sustituir U/u por Ü/ü

# Penalizaciones por Entregas Incompletas

- Proyecto que no este entregado de acuerdo con las especificaciones, será penalizado. Dos escenarios posibles:
  - El proyecto puede revisarse (completo o con faltas al formato).
  - El proyecto NO puede revisarse (falta codigo fuente, informe, APK, no se compila por alguna falla, etc). En automático el proyecto queda descartado.

Se recomienda LEER con cuidado la sección de entregables de esta presentación. Las penalizaciones son acumulables.

Detalle	Puntos de penalización sobre calificación final
Nombre Archivo	8
Tipo de Archivo	8
Estructura de Directorios	8
Falta o Error en Script	8
Poner ZIPs dentro del ZIP	8
Incluir ejecutables (aplica solo cuando el lenguaje es C++)	8

# Premio a la Compresión Lectora 2024

A pesar de las instrucciones en esta presentación, alguien va a hacer las cosas mal



# Salon de la Fama de Entregas Completas e Incompletas

Nombre	
	iti-271229_u1_trevino_gandarilla_jesus_david_source
	iti-271229_u1_trevino_gandarilla_jesus_david_latex
	iti-271229_u1_trevino_gandarilla_jesus_david.pdf

Nombre	
	iti-271229_u1_coyoy_lopez_mario_source
	iti-271229_u1_coyoy_lopez_mario_latex
	iti_271229_u1_coyoy_lopez_mario.pdf

Nombre	
	iti-271229_u1_olivares_rodriguez_brayan_source
	iti-271229_u1_olivares_rodriguez_brayan_latex
	iti-271229_u1_olivares_rodriguez_brayan.pdf

Nombre	
	iti-271229_u1_martinez_herrera_jose_guadalupe_source
	REPORTE INDIVIDUAL U2.zip
	REPORTE_INDIVIDUAL_U2.pdf

Nombre	
	iti-271154_u1_rodriguez_porras_alejandra_carolina_latex.zip
	iti-271154_u1_rodriguez_porras_alejandra_carolina_source.zip
	iti_271154_u1_rodriguez_porras_alejandra_carolina_latex .pdf

Nombre	
	iti-271229_u1_parras_pecina_maria_fernanda_latex
	iti-271229_u1_parras_pecina_maria_fernanda_source



# Respecto a las Asignaciones Especiales

## Importante

Si en algún momento del curso, la asignación especial consiste en desarrollar una presentación de un tema explicado en un BLOG, capítulo de libro, tutorial, etc., el LENGUAJE en el que deben hacerse las DIAPOSITIVAS es el MISMO que en el que está explicado dicho BLOG (a menos que se establezca un lenguaje diferente de manera explícita en el momento de asignar dicha tarea).

# Fechas importantes de entrega de proyectos (1)

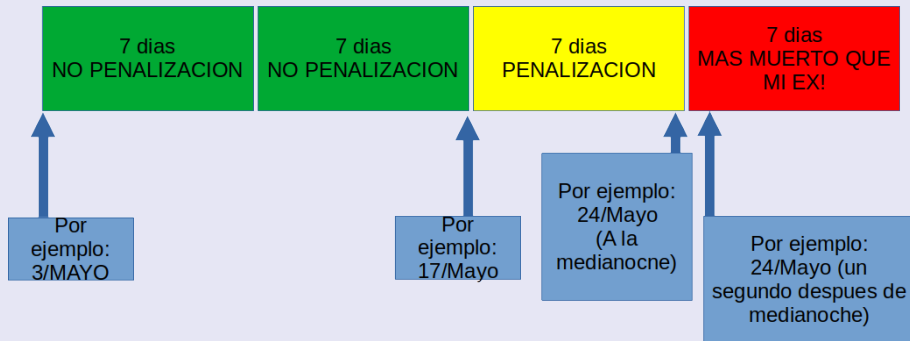
- Fecha de asignación: fecha en que se da a conocer al grupo el trabajo a elaborar
- Fecha de entrega sin penalización: 14 días naturales después de la fecha de asignación
- Proyecto entregado después de la fecha de penalización se le aplica una penalización de 25 PUNTOS
- Fecha de cierre: 21 días naturales después de la fecha de asignación.

## Regla “CANTU”

- Ningún proyecto será revisado después de la fecha de cierre. Se programarán las entregas para cerrar y no permitir entregas tardías.

# Fechas importantes de entrega de proyectos (2)

## Grafo "DAFNE"



- En el momento de publicar la tarea, se incluirá la fecha para no penalización y fecha de cierre.

# ¿Es posible obtener una calificación negativa?

SI

Calificación asignada después de revisión	70
(-) Debería entregarse 17/Mayo, lo entregó 24/Mayo (1 minuto antes de la medianoche)	25
(-) Nombre Archivo MAL	8
(-) Tipo de Archivo MAL	8
(-) Estructura de Directorios INCORRECTA	8
(-) Faltas o Errores en Script	8
(-) ZIPs dentro del ZIP	8
(-) Incluir ejecutables (aplica solo cuando el lenguaje es C++)	8
(-) Penalización por Fragmentación de Equipo	25
<b>Calificación Final</b>	<b>-28</b>

## LINUX

### En orden de dificultad

- Linux instalado de manera emulada usando VirtualBox o VMWare.
- Crear una USB o HD booteable (con persistencia) y bootear desde su laptop solo para las clases y los proyectos.
- Linux instalado de manera nativa. Distribuciones recomendadas: **Mint, Ubuntu, Lubuntu, Xubuntu, Debian**

**\*\* Tienen la opción de no INSTALAR LINUX, pero la evaluación será realiza en una PC con Linux instalado**

# Software Utilizado

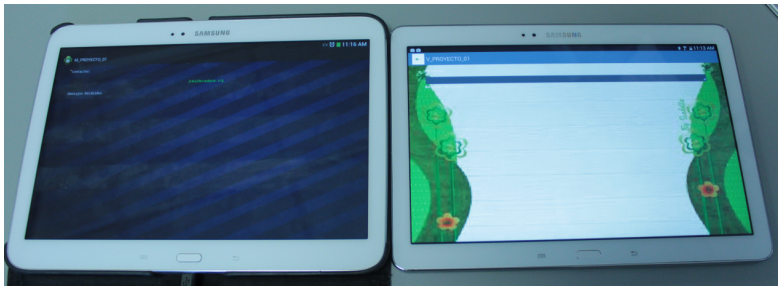
Sobre una instalación de Linux, se debe instalar lo siguiente:

- Navegador Chrome/Firefox actualizado
- LaTeX para edición de reportes
- Python3
- Otras librerías (se especificarán conforme se vayan utilizando)

Se buscan integrantes para ingresar al  
Salon de la fama del PLAGIO

# Plagio

- Reprobación automática a quien reproduzca códigos de otros compañeros y los reporte como suyos, además de una nota en su expediente con copia para el consejo de calidad





- Reprobación automática a quien copie códigos de Internet y los reporte como suyos, además de una nota en su expediente con copia para el consejo de calidad



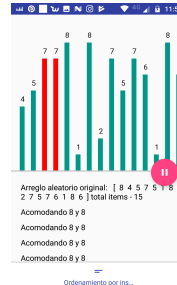
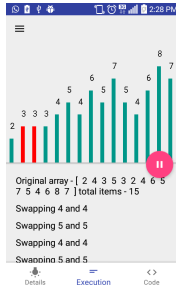
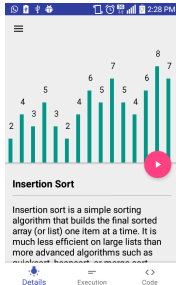
```

1 //Histogram equalization using C++ Image Processing
2 #include <iostream>
3 #include <opencv2/opencv.hpp>
4 #include <opencv2/imgproc/imgproc.hpp>
5 using namespace cv;
6 using namespace std;
7 using namespace cv;
8 using namespace cv;
9
10 //Funcion que inicializa todos los valores a 0
11 void initializeImage (int histogram[])
12 {
13     for(int i = 0; i < 256; i++)
14     {
15         histogram[i] = 0;
16     }
17 }
18
19 //Calculando el numero de pixeles de cada valor de intensidad
20 for(int i = 0; i < image.rows; i++)
21 {
22     for(int j = 0; j < image.cols; j++)
23     {
24         histogram[histImage.at<uchar>(i,j)]++;
25     }
26 }
27
28 //Funcion acumulativa del histograma
29 void calculateCumulativeHistogram (int histogram[])
30 {
31     cumulativeHistogram[0] = histogram[0];
32     for(int i = 1; i < 256; i++)
33     {
34         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
35     }
36 }
37
38 //Funcion que devuelve los histogramas
39 void normalizeHistogram (int histogram[], const char* name)
40 {
41     int hist[256];
42     for(int i = 0; i < 256; i++)
43     {
44         hist[i] = histogram[i];
45     }
46
47     //Calculando los histogramas
48     int hist_w = 512; int hist_h = 4096;
49     int bin_w = cvRound((double) 256/hist_w);
50     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
51
52     //Find the maximum intensity element from histogram
53     int max = hist[0];
54     for(int i = 1; i < 256; i++)
55     {
56         if(hist[i] > max)
57             max = hist[i];
58     }
59
60 //normalize the histogram between 0 and 255
61 for(int i = 0; i < 256; i++)
62 {
63     hist[i] = (float) hist[i] / max;
64 }
65
66 //Histogram equalization using C++ Image Processing
67 #include <iostream>
68 #include <opencv2/opencv.hpp>
69 #include <opencv2/imgproc/imgproc.hpp>
70 using namespace cv;
71 using namespace std;
72 using namespace cv;
73 using namespace cv;
74
75 void initializeImage (int histogram[])
76 {
77     for(int i = 0; i < 256; i++)
78     {
79         histogram[i] = 0;
80     }
81 }
82
83 //Calculando el numero de pixeles de cada valor de intensidad
84 for(int i = 0; i < image.rows; i++)
85 {
86     for(int j = 0; j < image.cols; j++)
87     {
88         histogram[histImage.at<uchar>(i,j)]++;
89     }
90 }
91
92 //Funcion acumulativa del histograma
93 void calculateCumulativeHistogram (int histogram[])
94 {
95     cumulativeHistogram[0] = histogram[0];
96     for(int i = 1; i < 256; i++)
97     {
98         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
99     }
100 }
101
102 //Funcion que devuelve los histogramas
103 void normalizeHistogram (int histogram[], const char* name)
104 {
105     int hist[256];
106     for(int i = 0; i < 256; i++)
107     {
108         hist[i] = histogram[i];
109     }
110
111     //Calculando los histogramas
112     int hist_w = 512; int hist_h = 4096;
113     int bin_w = cvRound((double) 256/hist_w);
114     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
115
116     //Find the maximum intensity element from histogram
117     int max = hist[0];
118     for(int i = 1; i < 256; i++)
119     {
120         if(hist[i] > max)
121             max = hist[i];
122     }
123
124 //normalize the histogram between 0 and 255
125 for(int i = 0; i < 256; i++)
126 {
127     hist[i] = (float) hist[i] / max;
128 }
129
130 //Histogram equalization using C++ Image Processing
131 #include <iostream>
132 #include <opencv2/opencv.hpp>
133 #include <opencv2/imgproc/imgproc.hpp>
134 using namespace cv;
135 using namespace std;
136 using namespace cv;
137 using namespace cv;
138
139 void initializeImage (int histogram[])
140 {
141     for(int i = 0; i < 256; i++)
142     {
143         histogram[i] = 0;
144     }
145 }
146
147 //Calculando el numero de pixeles de cada valor de intensidad
148 for(int i = 0; i < image.rows; i++)
149 {
150     for(int j = 0; j < image.cols; j++)
151     {
152         histogram[histImage.at<uchar>(i,j)]++;
153     }
154 }
155
156 //Funcion acumulativa del histograma
157 void calculateCumulativeHistogram (int histogram[])
158 {
159     cumulativeHistogram[0] = histogram[0];
160     for(int i = 1; i < 256; i++)
161     {
162         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
163     }
164 }
165
166 //Funcion que devuelve los histogramas
167 void normalizeHistogram (int histogram[], const char* name)
168 {
169     int hist[256];
170     for(int i = 0; i < 256; i++)
171     {
172         hist[i] = histogram[i];
173     }
174
175     //Calculando los histogramas
176     int hist_w = 512; int hist_h = 4096;
177     int bin_w = cvRound((double) 256/hist_w);
178     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
179
180     //Find the maximum intensity element from histogram
181     int max = hist[0];
182     for(int i = 1; i < 256; i++)
183     {
184         if(hist[i] > max)
185             max = hist[i];
186     }
187
188 //normalize the histogram between 0 and 255
189 for(int i = 0; i < 256; i++)
190 {
191     hist[i] = (float) hist[i] / max;
192 }
193
194 //Histogram equalization using C++ Image Processing
195 #include <iostream>
196 #include <opencv2/opencv.hpp>
197 #include <opencv2/imgproc/imgproc.hpp>
198 using namespace cv;
199 using namespace std;
200 using namespace cv;
201 using namespace cv;
202
203 void initializeImage (int histogram[])
204 {
205     for(int i = 0; i < 256; i++)
206     {
207         histogram[i] = 0;
208     }
209 }
210
211 //Calculando el numero de pixeles de cada valor de intensidad
212 for(int i = 0; i < image.rows; i++)
213 {
214     for(int j = 0; j < image.cols; j++)
215     {
216         histogram[histImage.at<uchar>(i,j)]++;
217     }
218 }
219
220 //Funcion acumulativa del histograma
221 void calculateCumulativeHistogram (int histogram[])
222 {
223     cumulativeHistogram[0] = histogram[0];
224     for(int i = 1; i < 256; i++)
225     {
226         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
227     }
228 }
229
230 //Funcion que devuelve los histogramas
231 void normalizeHistogram (int histogram[], const char* name)
232 {
233     int hist[256];
234     for(int i = 0; i < 256; i++)
235     {
236         hist[i] = histogram[i];
237     }
238
239     //Calculando los histogramas
240     int hist_w = 512; int hist_h = 4096;
241     int bin_w = cvRound((double) 256/hist_w);
242     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
243
244     //Find the maximum intensity element from histogram
245     int max = hist[0];
246     for(int i = 1; i < 256; i++)
247     {
248         if(hist[i] > max)
249             max = hist[i];
250     }
251
252 //normalize the histogram between 0 and 255
253 for(int i = 0; i < 256; i++)
254 {
255     hist[i] = (float) hist[i] / max;
256 }
257
258 //Histogram equalization using C++ Image Processing
259 #include <iostream>
260 #include <opencv2/opencv.hpp>
261 #include <opencv2/imgproc/imgproc.hpp>
262 using namespace cv;
263 using namespace std;
264 using namespace cv;
265 using namespace cv;
266
267 void initializeImage (int histogram[])
268 {
269     for(int i = 0; i < 256; i++)
270     {
271         histogram[i] = 0;
272     }
273 }
274
275 //Calculando el numero de pixeles de cada valor de intensidad
276 for(int i = 0; i < image.rows; i++)
277 {
278     for(int j = 0; j < image.cols; j++)
279     {
280         histogram[histImage.at<uchar>(i,j)]++;
281     }
282 }
283
284 //Funcion acumulativa del histograma
285 void calculateCumulativeHistogram (int histogram[])
286 {
287     cumulativeHistogram[0] = histogram[0];
288     for(int i = 1; i < 256; i++)
289     {
290         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
291     }
292 }
293
294 //Funcion que devuelve los histogramas
295 void normalizeHistogram (int histogram[], const char* name)
296 {
297     int hist[256];
298     for(int i = 0; i < 256; i++)
299     {
300         hist[i] = histogram[i];
301     }
302
303     //Calculando los histogramas
304     int hist_w = 512; int hist_h = 4096;
305     int bin_w = cvRound((double) 256/hist_w);
306     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
307
308     //Find the maximum intensity element from histogram
309     int max = hist[0];
310     for(int i = 1; i < 256; i++)
311     {
312         if(hist[i] > max)
313             max = hist[i];
314     }
315
316 //normalize the histogram between 0 and 255
317 for(int i = 0; i < 256; i++)
318 {
319     hist[i] = (float) hist[i] / max;
320 }
321
322 //Histogram equalization using C++ Image Processing
323 #include <iostream>
324 #include <opencv2/opencv.hpp>
325 #include <opencv2/imgproc/imgproc.hpp>
326 using namespace cv;
327 using namespace std;
328 using namespace cv;
329 using namespace cv;
330
331 void initializeImage (int histogram[])
332 {
333     for(int i = 0; i < 256; i++)
334     {
335         histogram[i] = 0;
336     }
337 }
338
339 //Calculando el numero de pixeles de cada valor de intensidad
340 for(int i = 0; i < image.rows; i++)
341 {
342     for(int j = 0; j < image.cols; j++)
343     {
344         histogram[histImage.at<uchar>(i,j)]++;
345     }
346 }
347
348 //Funcion acumulativa del histograma
349 void calculateCumulativeHistogram (int histogram[])
350 {
351     cumulativeHistogram[0] = histogram[0];
352     for(int i = 1; i < 256; i++)
353     {
354         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
355     }
356 }
357
358 //Funcion que devuelve los histogramas
359 void normalizeHistogram (int histogram[], const char* name)
360 {
361     int hist[256];
362     for(int i = 0; i < 256; i++)
363     {
364         hist[i] = histogram[i];
365     }
366
367     //Calculando los histogramas
368     int hist_w = 512; int hist_h = 4096;
369     int bin_w = cvRound((double) 256/hist_w);
370     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
371
372     //Find the maximum intensity element from histogram
373     int max = hist[0];
374     for(int i = 1; i < 256; i++)
375     {
376         if(hist[i] > max)
377             max = hist[i];
378     }
379
380 //normalize the histogram between 0 and 255
381 for(int i = 0; i < 256; i++)
382 {
383     hist[i] = (float) hist[i] / max;
384 }
385
386 //Histogram equalization using C++ Image Processing
387 #include <iostream>
388 #include <opencv2/opencv.hpp>
389 #include <opencv2/imgproc/imgproc.hpp>
390 using namespace cv;
391 using namespace std;
392 using namespace cv;
393 using namespace cv;
394
395 void initializeImage (int histogram[])
396 {
397     for(int i = 0; i < 256; i++)
398     {
399         histogram[i] = 0;
400     }
401 }
402
403 //Calculando el numero de pixeles de cada valor de intensidad
404 for(int i = 0; i < image.rows; i++)
405 {
406     for(int j = 0; j < image.cols; j++)
407     {
408         histogram[histImage.at<uchar>(i,j)]++;
409     }
410 }
411
412 //Funcion acumulativa del histograma
413 void calculateCumulativeHistogram (int histogram[])
414 {
415     cumulativeHistogram[0] = histogram[0];
416     for(int i = 1; i < 256; i++)
417     {
418         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
419     }
420 }
421
422 //Funcion que devuelve los histogramas
423 void normalizeHistogram (int histogram[], const char* name)
424 {
425     int hist[256];
426     for(int i = 0; i < 256; i++)
427     {
428         hist[i] = histogram[i];
429     }
430
431     //Calculando los histogramas
432     int hist_w = 512; int hist_h = 4096;
433     int bin_w = cvRound((double) 256/hist_w);
434     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
435
436     //Find the maximum intensity element from histogram
437     int max = hist[0];
438     for(int i = 1; i < 256; i++)
439     {
440         if(hist[i] > max)
441             max = hist[i];
442     }
443
444 //normalize the histogram between 0 and 255
445 for(int i = 0; i < 256; i++)
446 {
447     hist[i] = (float) hist[i] / max;
448 }
449
450 //Histogram equalization using C++ Image Processing
451 #include <iostream>
452 #include <opencv2/opencv.hpp>
453 #include <opencv2/imgproc/imgproc.hpp>
454 using namespace cv;
455 using namespace std;
456 using namespace cv;
457 using namespace cv;
458
459 void initializeImage (int histogram[])
460 {
461     for(int i = 0; i < 256; i++)
462     {
463         histogram[i] = 0;
464     }
465 }
466
467 //Calculando el numero de pixeles de cada valor de intensidad
468 for(int i = 0; i < image.rows; i++)
469 {
470     for(int j = 0; j < image.cols; j++)
471     {
472         histogram[histImage.at<uchar>(i,j)]++;
473     }
474 }
475
476 //Funcion acumulativa del histograma
477 void calculateCumulativeHistogram (int histogram[])
478 {
479     cumulativeHistogram[0] = histogram[0];
480     for(int i = 1; i < 256; i++)
481     {
482         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
483     }
484 }
485
486 //Funcion que devuelve los histogramas
487 void normalizeHistogram (int histogram[], const char* name)
488 {
489     int hist[256];
490     for(int i = 0; i < 256; i++)
491     {
492         hist[i] = histogram[i];
493     }
494
495     //Calculando los histogramas
496     int hist_w = 512; int hist_h = 4096;
497     int bin_w = cvRound((double) 256/hist_w);
498     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
499
500     //Find the maximum intensity element from histogram
501     int max = hist[0];
502     for(int i = 1; i < 256; i++)
503     {
504         if(hist[i] > max)
505             max = hist[i];
506     }
507
508 //normalize the histogram between 0 and 255
509 for(int i = 0; i < 256; i++)
510 {
511     hist[i] = (float) hist[i] / max;
512 }
513
514 //Histogram equalization using C++ Image Processing
515 #include <iostream>
516 #include <opencv2/opencv.hpp>
517 #include <opencv2/imgproc/imgproc.hpp>
518 using namespace cv;
519 using namespace std;
520 using namespace cv;
521 using namespace cv;
522
523 void initializeImage (int histogram[])
524 {
525     for(int i = 0; i < 256; i++)
526     {
527         histogram[i] = 0;
528     }
529 }
530
531 //Calculando el numero de pixeles de cada valor de intensidad
532 for(int i = 0; i < image.rows; i++)
533 {
534     for(int j = 0; j < image.cols; j++)
535     {
536         histogram[histImage.at<uchar>(i,j)]++;
537     }
538 }
539
540 //Funcion acumulativa del histograma
541 void calculateCumulativeHistogram (int histogram[])
542 {
543     cumulativeHistogram[0] = histogram[0];
544     for(int i = 1; i < 256; i++)
545     {
546         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
547     }
548 }
549
550 //Funcion que devuelve los histogramas
551 void normalizeHistogram (int histogram[], const char* name)
552 {
553     int hist[256];
554     for(int i = 0; i < 256; i++)
555     {
556         hist[i] = histogram[i];
557     }
558
559     //Calculando los histogramas
560     int hist_w = 512; int hist_h = 4096;
561     int bin_w = cvRound((double) 256/hist_w);
562     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
563
564     //Find the maximum intensity element from histogram
565     int max = hist[0];
566     for(int i = 1; i < 256; i++)
567     {
568         if(hist[i] > max)
569             max = hist[i];
570     }
571
572 //normalize the histogram between 0 and 255
573 for(int i = 0; i < 256; i++)
574 {
575     hist[i] = (float) hist[i] / max;
576 }
577
578 //Histogram equalization using C++ Image Processing
579 #include <iostream>
580 #include <opencv2/opencv.hpp>
581 #include <opencv2/imgproc/imgproc.hpp>
582 using namespace cv;
583 using namespace std;
584 using namespace cv;
585 using namespace cv;
586
587 void initializeImage (int histogram[])
588 {
589     for(int i = 0; i < 256; i++)
590     {
591         histogram[i] = 0;
592     }
593 }
594
595 //Calculando el numero de pixeles de cada valor de intensidad
596 for(int i = 0; i < image.rows; i++)
597 {
598     for(int j = 0; j < image.cols; j++)
599     {
600         histogram[histImage.at<uchar>(i,j)]++;
601     }
602 }
603
604 //Funcion acumulativa del histograma
605 void calculateCumulativeHistogram (int histogram[])
606 {
607     cumulativeHistogram[0] = histogram[0];
608     for(int i = 1; i < 256; i++)
609     {
610         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
611     }
612 }
613
614 //Funcion que devuelve los histogramas
615 void normalizeHistogram (int histogram[], const char* name)
616 {
617     int hist[256];
618     for(int i = 0; i < 256; i++)
619     {
620         hist[i] = histogram[i];
621     }
622
623     //Calculando los histogramas
624     int hist_w = 512; int hist_h = 4096;
625     int bin_w = cvRound((double) 256/hist_w);
626     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
627
628     //Find the maximum intensity element from histogram
629     int max = hist[0];
630     for(int i = 1; i < 256; i++)
631     {
632         if(hist[i] > max)
633             max = hist[i];
634     }
635
636 //normalize the histogram between 0 and 255
637 for(int i = 0; i < 256; i++)
638 {
639     hist[i] = (float) hist[i] / max;
640 }
641
642 //Histogram equalization using C++ Image Processing
643 #include <iostream>
644 #include <opencv2/opencv.hpp>
645 #include <opencv2/imgproc/imgproc.hpp>
646 using namespace cv;
647 using namespace std;
648 using namespace cv;
649 using namespace cv;
650
651 void initializeImage (int histogram[])
652 {
653     for(int i = 0; i < 256; i++)
654     {
655         histogram[i] = 0;
656     }
657 }
658
659 //Calculando el numero de pixeles de cada valor de intensidad
660 for(int i = 0; i < image.rows; i++)
661 {
662     for(int j = 0; j < image.cols; j++)
663     {
664         histogram[histImage.at<uchar>(i,j)]++;
665     }
666 }
667
668 //Funcion acumulativa del histograma
669 void calculateCumulativeHistogram (int histogram[])
670 {
671     cumulativeHistogram[0] = histogram[0];
672     for(int i = 1; i < 256; i++)
673     {
674         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
675     }
676 }
677
678 //Funcion que devuelve los histogramas
679 void normalizeHistogram (int histogram[], const char* name)
680 {
681     int hist[256];
682     for(int i = 0; i < 256; i++)
683     {
684         hist[i] = histogram[i];
685     }
686
687     //Calculando los histogramas
688     int hist_w = 512; int hist_h = 4096;
689     int bin_w = cvRound((double) 256/hist_w);
690     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
691
692     //Find the maximum intensity element from histogram
693     int max = hist[0];
694     for(int i = 1; i < 256; i++)
695     {
696         if(hist[i] > max)
697             max = hist[i];
698     }
699
700 //normalize the histogram between 0 and 255
701 for(int i = 0; i < 256; i++)
702 {
703     hist[i] = (float) hist[i] / max;
704 }
705
706 //Histogram equalization using C++ Image Processing
707 #include <iostream>
708 #include <opencv2/opencv.hpp>
709 #include <opencv2/imgproc/imgproc.hpp>
710 using namespace cv;
711 using namespace std;
712 using namespace cv;
713 using namespace cv;
714
715 void initializeImage (int histogram[])
716 {
717     for(int i = 0; i < 256; i++)
718     {
719         histogram[i] = 0;
720     }
721 }
722
723 //Calculando el numero de pixeles de cada valor de intensidad
724 for(int i = 0; i < image.rows; i++)
725 {
726     for(int j = 0; j < image.cols; j++)
727     {
728         histogram[histImage.at<uchar>(i,j)]++;
729     }
730 }
731
732 //Funcion acumulativa del histograma
733 void calculateCumulativeHistogram (int histogram[])
734 {
735     cumulativeHistogram[0] = histogram[0];
736     for(int i = 1; i < 256; i++)
737     {
738         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
739     }
740 }
741
742 //Funcion que devuelve los histogramas
743 void normalizeHistogram (int histogram[], const char* name)
744 {
745     int hist[256];
746     for(int i = 0; i < 256; i++)
747     {
748         hist[i] = histogram[i];
749     }
750
751     //Calculando los histogramas
752     int hist_w = 512; int hist_h = 4096;
753     int bin_w = cvRound((double) 256/hist_w);
754     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
755
756     //Find the maximum intensity element from histogram
757     int max = hist[0];
758     for(int i = 1; i < 256; i++)
759     {
760         if(hist[i] > max)
761             max = hist[i];
762     }
763
764 //normalize the histogram between 0 and 255
765 for(int i = 0; i < 256; i++)
766 {
767     hist[i] = (float) hist[i] / max;
768 }
769
770 //Histogram equalization using C++ Image Processing
771 #include <iostream>
772 #include <opencv2/opencv.hpp>
773 #include <opencv2/imgproc/imgproc.hpp>
774 using namespace cv;
775 using namespace std;
776 using namespace cv;
777 using namespace cv;
778
779 void initializeImage (int histogram[])
780 {
781     for(int i = 0; i < 256; i++)
782     {
783         histogram[i] = 0;
784     }
785 }
786
787 //Calculando el numero de pixeles de cada valor de intensidad
788 for(int i = 0; i < image.rows; i++)
789 {
790     for(int j = 0; j < image.cols; j++)
791     {
792         histogram[histImage.at<uchar>(i,j)]++;
793     }
794 }
795
796 //Funcion acumulativa del histograma
797 void calculateCumulativeHistogram (int histogram[])
798 {
799     cumulativeHistogram[0] = histogram[0];
800     for(int i = 1; i < 256; i++)
801     {
802         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
803     }
804 }
805
806 //Funcion que devuelve los histogramas
807 void normalizeHistogram (int histogram[], const char* name)
808 {
809     int hist[256];
810     for(int i = 0; i < 256; i++)
811     {
812         hist[i] = histogram[i];
813     }
814
815     //Calculando los histogramas
816     int hist_w = 512; int hist_h = 4096;
817     int bin_w = cvRound((double) 256/hist_w);
818     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
819
820     //Find the maximum intensity element from histogram
821     int max = hist[0];
822     for(int i = 1; i < 256; i++)
823     {
824         if(hist[i] > max)
825             max = hist[i];
826     }
827
828 //normalize the histogram between 0 and 255
829 for(int i = 0; i < 256; i++)
830 {
831     hist[i] = (float) hist[i] / max;
832 }
833
834 //Histogram equalization using C++ Image Processing
835 #include <iostream>
836 #include <opencv2/opencv.hpp>
837 #include <opencv2/imgproc/imgproc.hpp>
838 using namespace cv;
839 using namespace std;
840 using namespace cv;
841 using namespace cv;
842
843 void initializeImage (int histogram[])
844 {
845     for(int i = 0; i < 256; i++)
846     {
847         histogram[i] = 0;
848     }
849 }
850
851 //Calculando el numero de pixeles de cada valor de intensidad
852 for(int i = 0; i < image.rows; i++)
853 {
854     for(int j = 0; j < image.cols; j++)
855     {
856         histogram[histImage.at<uchar>(i,j)]++;
857     }
858 }
859
860 //Funcion acumulativa del histograma
861 void calculateCumulativeHistogram (int histogram[])
862 {
863     cumulativeHistogram[0] = histogram[0];
864     for(int i = 1; i < 256; i++)
865     {
866         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
867     }
868 }
869
870 //Funcion que devuelve los histogramas
871 void normalizeHistogram (int histogram[], const char* name)
872 {
873     int hist[256];
874     for(int i = 0; i < 256; i++)
875     {
876         hist[i] = histogram[i];
877     }
878
879     //Calculando los histogramas
880     int hist_w = 512; int hist_h = 4096;
881     int bin_w = cvRound((double) 256/hist_w);
882     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
883
884     //Find the maximum intensity element from histogram
885     int max = hist[0];
886     for(int i = 1; i < 256; i++)
887     {
888         if(hist[i] > max)
889             max = hist[i];
890     }
891
892 //normalize the histogram between 0 and 255
893 for(int i = 0; i < 256; i++)
894 {
895     hist[i] = (float) hist[i] / max;
896 }
897
898 //Histogram equalization using C++ Image Processing
899 #include <iostream>
900 #include <opencv2/opencv.hpp>
901 #include <opencv2/imgproc/imgproc.hpp>
902 using namespace cv;
903 using namespace std;
904 using namespace cv;
905 using namespace cv;
906
907 void initializeImage (int histogram[])
908 {
909     for(int i = 0; i < 256; i++)
910     {
911         histogram[i] = 0;
912     }
913 }
914
915 //Calculando el numero de pixeles de cada valor de intensidad
916 for(int i = 0; i < image.rows; i++)
917 {
918     for(int j = 0; j < image.cols; j++)
919     {
920         histogram[histImage.at<uchar>(i,j)]++;
921     }
922 }
923
924 //Funcion acumulativa del histograma
925 void calculateCumulativeHistogram (int histogram[])
926 {
927     cumulativeHistogram[0] = histogram[0];
928     for(int i = 1; i < 256; i++)
929     {
930         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
931     }
932 }
933
934 //Funcion que devuelve los histogramas
935 void normalizeHistogram (int histogram[], const char* name)
936 {
937     int hist[256];
938     for(int i = 0; i < 256; i++)
939     {
940         hist[i] = histogram[i];
941     }
942
943     //Calculando los histogramas
944     int hist_w = 512; int hist_h = 4096;
945     int bin_w = cvRound((double) 256/hist_w);
946     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
947
948     //Find the maximum intensity element from histogram
949     int max = hist[0];
950     for(int i = 1; i < 256; i++)
951     {
952         if(hist[i] > max)
953             max = hist[i];
954     }
955
956 //normalize the histogram between 0 and 255
957 for(int i = 0; i < 256; i++)
958 {
959     hist[i] = (float) hist[i] / max;
960 }
961
962 //Histogram equalization using C++ Image Processing
963 #include <iostream>
964 #include <opencv2/opencv.hpp>
965 #include <opencv2/imgproc/imgproc.hpp>
966 using namespace cv;
967 using namespace std;
968 using namespace cv;
969 using namespace cv;
970
971 void initializeImage (int histogram[])
972 {
973     for(int i = 0; i < 256; i++)
974     {
975         histogram[i] = 0;
976     }
977 }
978
979 //Calculando el numero de pixeles de cada valor de intensidad
980 for(int i = 0; i < image.rows; i++)
981 {
982     for(int j = 0; j < image.cols; j++)
983     {
984         histogram[histImage.at<uchar>(i,j)]++;
985     }
986 }
987
988 //Funcion acumulativa del histograma
989 void calculateCumulativeHistogram (int histogram[])
990 {
991     cumulativeHistogram[0] = histogram[0];
992     for(int i = 1; i < 256; i++)
993     {
994         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
995     }
996 }
997
998 //Funcion que devuelve los histogramas
999 void normalizeHistogram (int histogram[], const char* name)
1000 {
1001     int hist[256];
1002     for(int i = 0; i < 256; i++)
1003     {
1004         hist[i] = histogram[i];
1005     }
1006
1007     //Calculando los histogramas
1008     int hist_w = 512; int hist_h = 4096;
1009     int bin_w = cvRound((double) 256/hist_w);
1010     Mat histImage(hist_h, hist_w, CV_32F, Scalar(0, 0, 0, 0));
1011
1012     //Find the maximum intensity element from histogram
1013     int max = hist[0];
1014     for(int i = 1; i < 256; i++)
1015     {
1016         if(hist[i] > max)
1017             max = hist[i];
1018     }
1019
1020 //normalize the histogram between 0 and 255
1021 for(int i = 0; i < 256; i++)
1022 {
1023     hist[i] = (float) hist[i] / max;
1024 }
1025
1026 //Histogram equalization using C++ Image Processing
1027 #include <iostream>
1028 #include <opencv2/opencv.hpp>
1029 #include <opencv2/imgproc/imgproc.hpp>
1030 using namespace cv;
1031 using namespace std;
1032 using namespace cv;
1033 using namespace cv;
1034
1035 void initializeImage (int histogram[])
1036 {
1037     for(int i = 0; i < 256; i++)
1038     {
1039         histogram[i] = 0;
1040     }
1041 }
1042
1043 //Calculando el numero de pixeles de cada valor de intensidad
1044 for(int i = 0; i < image.rows; i++)
1045 {
1046     for(int j = 0; j < image.cols; j++)
1047     {
1048         histogram[histImage.at<uchar>(i,j)]++;
1049     }
1050 }
1051
1052 //Funcion acumulativa del histograma
1053 void calculateCumulativeHistogram (int histogram[])
1054 {
1055     cumulativeHistogram[0] = histogram[0];
1056     for(int i = 1; i < 256; i++)
1057     {
1058         cumulativeHistogram[i] = cumulativeHistogram[i-1] + histogram[i];
1059     }
1060 }
1061
1062 //Funcion que devuelve los histogramas
1063 void normalizeHistogram (int histogram[], const char* name)
1064 {
1065     int hist[256];
1066     for(int i = 0; i < 256; i++)
1067     {
1068         hist[i] = histogram[i];
1069     }
1070
1071     //Calculando los histogramas
1072     int hist_w = 512; int hist_h = 4096;
1073     int bin_w = cv
```

- Reprobación automática a quien copie códigos de Internet y los reporte como suyos, además de una nota en su expediente con copia para el consejo de calidad



<https://github.com/naman14/AlgorithmVisualizer-Android>



“Finalmente son jóvenes que están en la preparatoria y que deben de leer su convocatoria con toda claridad, si no cumplen con los requisitos, si no pueden leer una convocatoria que dice tienes que traer número uno esto, número dos esto, número tres esto, no están listos para ser **estudiantes de educación superior**, así lo digo con toda claridad”.

Sara Ladrón de Guevara.

Rectora de la Universidad Veracruzana (2013-2017 y 2017-2021).

# CONCLUSIÓN

