Procesamiento de imágenes médicas con la aplicación de Snakes

Presentado por:

María Alejandra Flores Meneses-2520181032
Mileidy Alejandra Herrera Orjuela-2520172046
Harvy Jefferson Albarrán Salazar- 2181801786
Javier Manuel Delgado Farro- 2181801868
Nicolás Orlando Pérez cruz- 2220181014
Andy Josué Santisteban Ostos- 2201802996
Juan David forero acosta-2220181022

Presentado a:

Ing. Manuel Guillermo Forero Vargas

Universidad de Ibagué, Universidad Señor de Sipán
Facultad de Ingeniería
2021

TABLA DE CONTENIDO

RESUMEN

Al hablar de procesamiento de imágenes con el filtro Snake, se evidencia una curva elástica cercana que, colocada sobre una imagen, comienza a desfigurar a partir de una forma inicial con el fin de delimitar las regiones de interés en la escena. El propósito del presente proyecto está enfocado en presentar formalmente la ecuación del Snake como un caso particular de la teoría de la regularización para así, obtener una herramienta eficaz en múltiples tareas, como, por ejemplo, en el análisis de imágenes médicas, donde la baja relación señal/ruido hace insuficientes los resultados obtenidos mediante técnicas clásicas.
La elaboración y ejecución del presente filtro codificado en lenguaje Java se consiguió mediante el uso de los softwares ImageJ y NetBeans, permitiendo así comprobar que el método empleado aporta resultados satisfactorios en la segmentación de imágenes debido a que presenta mejores resultados en imágenes de endoscopia que en imágenes de ultrasonido.

Palabras claves: *Curva elástica, Snake, Imágenes, Segmentación, Java, ImageJ y NetBeans.*

ABSTRACT

When talking about image processing with the Snake filter, a close elastic curve is evident that, placed on an image, begins to distort from an initial shape in order to delimit the regions of interest in the scene. The purpose of this project is focused on formally presenting the snake equation as a particular case of the regularization theory in order to obtain an effective tool in multiple tasks, such as, for example, in the analysis of medical images, where the low the signal / noise ratio makes the results obtained by classical techniques insufficient.

The elaboration and execution of this filter encoded in Java was achieved by using the ImageJ and NetBeans software, thus allowing to verify that the method used provides satisfactory results in the segmentation of images due to the fact that it presents better results in endoscopy images than in images. ultrasound.

Keywords: *Elastic curve, Snakes, Images, Segmentation, Java, ImageJ and NetBeans.*

1 INTRODUCCIÓN

El procesamiento de imágenes tiene como propósito mejorar la calidad y aspecto de las imágenes para así hacer más evidentes en ellas ciertos detalles que se quiere hacer notar o destacar. Por otra parte, a lo largo de nuestra existencia el procesamiento de las imágenes se puede generar por medio de métodos relevantes como también ópticos, o mejor aún por medio de métodos digitales, es decir en una computadora. Por otro lado, el tratamiento digital de imágenes consiste en procesos algorítmicos que transforman una imagen en otra en donde se resalta cierta información de interés, y/o se atenúa o elimina información irrelevante para la aplicación. Así, las tareas del procesamiento de imágenes comprenden la supresión de ruido, mejoramientos de contraste, eliminación de efectos no deseados en la captura como difuminaciones o distorsiones por efectos ópticos o de movimiento, mapeos geométricos, transformaciones de color, etc.

Los contornos activos o también llamados snakes han ganado estimación y aun así se ha convertido en un mecanismo eficaz en numerosas tareas relacionadas con el procesamiento y el análisis de imágenes como puede ser, entre otras, la segmentación y el seguimiento de objetos móviles o su deformación. También se puede decir que la implementación de las imágenes médicas mediante la aplicación del modelo Snake es muy común en el área de medicina ya que se presentan buenos resultados para las imágenes de resonancia magnética en escala de grises y en imágenes sintéticas. El desarrollo que genera la implementación del modelo Snake en las imágenes médicas es muy alto ya que se puede realizar una segmentación de imágenes en la medicina. El análisis digital de imágenes es el área de Ingeniería que se encarga de la extracción de mediciones, datos e información contenida en una imagen, ofreciendo solución para diferentes problemas, donde se puede obtener de manera más precisa el análisis de las imágenes y la visión computacional y de esta manera poder definir, detectar, localizar y describir la forma de objetos. Una imagen digital está

definida como una función de dos dimensiones tanto en x como en y, un plano en el que se contienen los puntos de la misma, está compuesta por un numero finito de elementos y cada uno de estos tiene una localidad y valor particular.

El presente trabajo tiene como objetivo, presentar formalmente la ecuación de Snake como un caso característico de la teoría de la regularización mediante el lenguaje java ejecutando la aplicación de Snake mediante el software Imagej, así mismo ajustando aquella ecuación a la representación evidenciando una curva elástica, con el fin de relevar aspectos importantes de la imagen, también este modelo se convierte en una herramienta muy eficaz para la rama de la medicina como se nombró anteriormente.

2 MARCO TEÓRICO

En el presente proyecto se expone los principales conceptos que se han realizado en el modelo snakes siendo una herramienta teórica y manejable de buen funcionamiento y resultado en cuanto a las imágenes digitales del área de la medicina como lo son las Imágenes de ultrasonido, endoscopias y otras. Se define la forma en la que el modelo Snakes involucran los contornos, contrastes y demás características de las imágenes, En este caso se empleara el modelo Gaussian para atiborrar imágenes y eliminar posibles ruidos que se generan especialmente en las imágenes digitales del ultrasonido las cuales al sacar el examen generan Ondas Sonoras que se ven reflejadas en la Imagen, también hacer cambios de contraste, movimiento, color y otros con el uso del modelo Snakes. La combinación de la Ingeniería con la medicina en cuanto a las Imágenes digitales y el procesamiento de ellas hacen un buen enlace y avance.

ImagineJ es un programa el cual está diseñado para el análisis de imagines y el mejoramiento de las mismas, esta aplicación mide áreas, cuenta objetos, cuantifica la señal, mejoramiento de la calidad de las imágenes interrumpidas por las Ondas, como lo puede ser un examen de Ultrasonido en el caso medicinal.

Con la ejecución de la aplicación ImagineJ también se puede mejorar el aspecto de la imagen, manejo de pilas de imágenes como lo es el Staks, entre otros; Mediante “Pluggins” se pueden añadir muchas más funciones. Ya que los pluggins son módulos de Software adicional que sirven para realizar tareas más específicas.

En cuanto a la elaboración del proyecto también se implementó una aplicación llamada ImageJ es una UI interesante y de fácil manejo, cuando se procede a aplicar el filtro se puede apreciar parámetros que se mostraran más delante de forma más detallada en los resultados, donde este parámetro requiere de la derivada Gaussiano, en un plano cartesiano de dos dimensiones (x,y). Para este proyecto aparte de usar el método Snakes y la interfaz ImajeJ, también se ejecutó Pluggin, el cual se elaboró en lenguaje Java atreves de ImageJ, donde se busca tener un mejor aprecio de las imágenes médicas, en las partes en las que se requiera, en ocasiones se presentan casos de mapeo, efectos ópticos y otras causas que lo que logra es no tener un buen aprecio de la Imagen como tal, por ello se ejecuta Pluggin por medio de la aplicación ImagineJ teniendo en cuenta que el método utilizado para el presente proyecto es el método de Snake.

2.1 Snakes

El procesamiento de imágenes, el reconocimiento de ciertos objetos o la detección de ciertos rasgos partículas en este caso de las imágenes digitales en la medicina, han sido un caso muy estudiado.

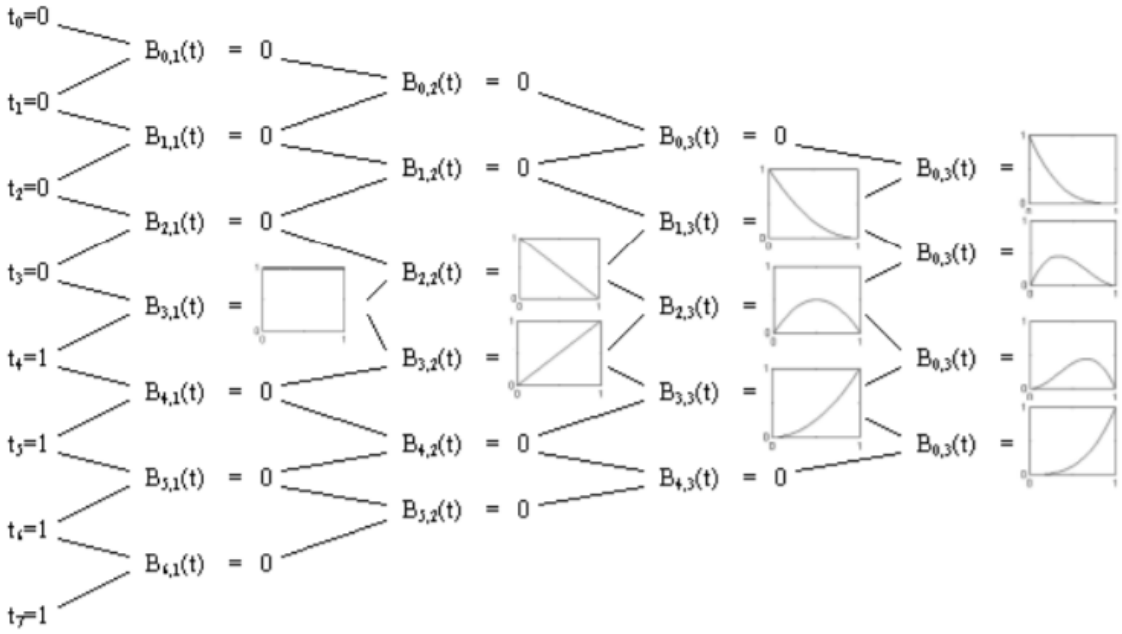


Ilustración 1. Funciones Base

En estos casos se han usado operadores como lo son las máscaras, las cuales cumple un diseño para producir otra imagen que resalte los rasgos más específicos de la imagen, donde se puede correlacionar la matemática o convolución. Cuando se obtiene la resultante de la correlación es una medida exacta del prototipo de rasgos en la imagen.

Para detectar los bordes en ocasiones es muy necesario que se tomen decisiones por cada pixel. La idea más esencial seria obtener los bordes por cada de los pixeles F(r) y de esta manera emplear el snake como una curva deformable r (s) la cual se dirija en movimiento hasta la altitud de las respuestas F, donde estas se obtienen maximizando F (r(s))

Con $0 \leq s \geq 1$

Las fuerzas que se ejercen internamente del Spline se componen de una condición de suavidad del Snake, debido a que las fuerzas de la imagen se dirijan hacia los bordes o contornos subjetivos.

A continuación, paramétricamente se impone la condición particular de suavidad del Snake. Donde la posición de un Snake se define así $v(s) = (\mathrm{x}\,(\,(s)), \mathrm{y}\,(\,\mathrm{s}))$ se puede escribir su energía funcional de la siguen manera

$$E^*_{snake} = \int_0^1 \big(v(s)\big)ds = \int_0^1 E_{int}\big(\,v(s)\big) + E_{image}\big(v(s)\big) + E_{con(v(s))}ds$$

Donde E_{int} representa la energía interna del spline E_{image} son las fuerzas que contienen las imágenes y el E_{con} es donde se aplicaran las fuerzas externas.

El modelo Snake puede ser un modelo deformable el cual se define como un modelo de contorno:

$$\Omega = [\,0.1] \rightarrow R^2$$

$$\mathrm{s} \rightarrow v(s) = (x(s), y\,(s))$$

Es así como definirá el modelo deformable en un espacio admisible Ad y una función E, a ser minimizada. Donde esta función será la que deberá representar la energía del modelo

$$\mathrm{E}: Ad \rightarrow R$$

$$\mathrm{v} \rightarrow E(v) = \int_\Omega \omega r\big|\, v'(s)R|^2 + \omega_2\,\big|v''(s)|ds\,((s)|^2 + P\,((v(s))ds$$

P es la potencia asociada a las fuerzas externas de la imagen digital. Se desea que el Snakes sea atraído hacia los bordes de la imagen donde la energía potencial extra dada en función del gradiente de la imagen. Ad es restringido por las condiciones de borde $v(0), v'(0), v(1) y\, v'(1)$ Serán conocidas

ω_j es una de las propiedades mecánicas del modelo controlado y de ella dependerá la elasticidad del modelo y su rigidez.

E satisface la ecuación asociada de Euler-Lagrange para las siguientes condiciones del modelo Snakes

$$-(\omega_1 v')' + (\omega_2 v'')'') + \nabla P(v) = 0$$

$$v(0), v'(0), v(1) y\ v'(1) \text{ son conocidos}$$

En la formulación cada termino aparece como una aplicación de fuerza. La curva está dada bajo la interacción de las fuerzas tanto internas como externas. Las constantes ω_1 y ω_2 se aplican las fuerzas internas y componen en la imagen digital la fuerza, elasticidad y rigidez del modelo, donde la potencia se da de la siguiente forma

$$\int_0^1 P\big(v(s)\big) ds$$

De manera más formal se presentará como en la siguiente ecuación:

$$P(v) = -\nabla Q(v)|^2$$

Q es la denotación de la imagen digital. Donde el 1 mínimo local de la potencia es hacia donde se eleva la curva, también expresado o conocido como gradiente local.

Teóricamente una Snake es un contorno paramétrico v(s,t)=(x(s,t),y(s,t))t , variable en el tiempo y definido en el plano de la imagen (x, y) ϵ R2, donde las coordenadas x(s,t), y(s,t)

del contorno son funciones de la variable paramétrica s ϵ [0,1], y del tiempo t. El contorno se supone cerrado, a través de condiciones de contorno.

La forma del contorno se manifiesta mediante la siguiente funcional de energía E, en el que se debe ser minimizada con el fin de determinar la forma y posición final de la Snake:

Etotal = 01E(v(s))ds=01(Eint(v(s))+Eext(v(s)))ds (Ecu. 1)

Eint=(s)vs2+ (s)2vs22 (Ecu. 2)

Eint y Eext corresponden a los términos de energía interna y externa, respectivamente. Eint Dada las singularidades de deformación del contorno elástico y las funciones α(s) y β(s) determinan el grado en el cual la Snake se puede estirar o curvar. Estas funciones son útiles para manipular el comportamiento físico y la continuidad local del modelo.

Una Snake 3D es un contorno paramétrico v(s,t)=[x(s,t),y(s,t), z(s,t)]t , variable en el tiempo y definido en la superficie (x, y, z) ϵ R3, donde las coordenadas x(s,t), y(s,t) y z(s,t) del contorno son funciones de la variable paramétrica s ϵ [0,1], y del tiempo t.

El objetivo de la Snake en 3D es moverse alrededor de la superficie con el fin de minimizar su función de energía

Etotal= 01E(v(s))ds = 01(Eint(v(s))+Eext(v(s)))ds (Ecu. 3)

El método propuesto para deformar la Snake en 3D alrededor de la superficie a segmentar consiste en lo siguiente:

- Dado un nodo de la Snake 2D de la imagen izquierda (p0,0) y su entorno 3x3

Ecu. 4

$$\begin{bmatrix} p_{-1,-1} & p_{0,-1} & p_{1,-1} \\ p_{-1,0} & p_{0,0} & p_{1,0} \\ p_{-1,1} & p_{0,1} & p_{1,1} \end{bmatrix}$$

- También se calcula la función de energía en el entorno 3x3 alrededor del nodo.

Ecu. 5

$$E_{izq} = \begin{bmatrix} E_{p-1,-1} & E_{p0,-1} & E_{p1,-1} \\ E_{p-1,0} & E_{p0,0} & E_{p1,0} \\ E_{p-1,1} & E_{p0,1} & E_{p1,1} \end{bmatrix}$$

- Al igual que, para cada punto del entorno 3x3, se calculará cuál es su correspondiente en la imagen derecha, usando para ello la indagación obtenida mediante el cálculo del mapa de disparidad.

Ecu.6

$$\begin{bmatrix} p_{-1,-1} \\ p_{0,-1} \\ p_{1,-1} \\ p_{-1,0} \\ p_{0,0} \\ p_{1,0} \\ p_{-1,1} \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \end{bmatrix}$$

$$\begin{bmatrix} p0,1 \\ p1,1 \end{bmatrix} \begin{bmatrix} p8 \\ p9 \end{bmatrix}$$

- Se calculará la energía de cada elemento del vector de correspondencia

Ecu. 7

$$Eder = \begin{bmatrix} Ep1 & Ep2 & Ep3 \\ Ep4 & Ep5 & Ep6 \\ Ep7 & Ep8 & Ep9 \end{bmatrix}$$

- Se suman las componentes de energía de la imagen izquierda con la obtenida con la imagen derecha

ESnake 3D = Eizq+ Eder (Ecu. 8)

- El elemento que posea la menor energía, será la nueva posición del nodo de la Snake

2.2 Filtros digitales

Los filtros digitales hacen parte de unos de los principales modos de operar en el procesamiento de imágenes digitales. También sirve para distintos fines, pero en todos los casos, el resultado sobre cada pixel depende de los pixeles de su entorno.

Una imagen se puede filtrar en el dominio del espacio, trabajando directamente sobre los píxeles de la imagen, o en el dominio de la frecuencia, donde las operaciones se llevan a cabo en la transformada de Fourier de la imagen. (Filtros, s.f.)

Los objetivos que tienen estos filtros son los siguientes:

- **Suavizar la imagen**: lo que hace es disminuir las variaciones de intensidad entre pixeles vecinos.
- **Eliminar ruido**: innovar aquellos pixeles cuyo nivel de intensidad es muy distinto al de sus vecinos.
- **Realzar la imagen:** aumentar las variaciones de intensidad.
- **Detectar bordes:** detectar pixeles que provoca un cambio brusco en la función de intensidad.

2.2.1 Filtro gaussiano

Normalmente este filtro se usa para emborronar imágenes y eliminar ruido. Es prácticamente similar al filtro de media, pero se usa una máscara diferente, un breve ejemplo es lo siguiente:

Ilustración 2. Imagen original

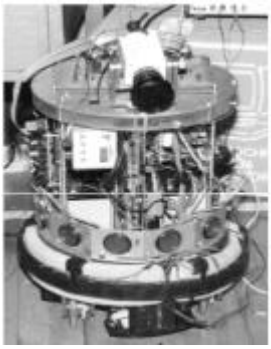


Ilustración 3. Imagen con filtro gaussiano con σ=1.0

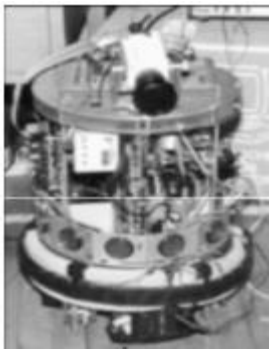


Ilustración 4. Imagen con filtro gaussiano con σ=2.0

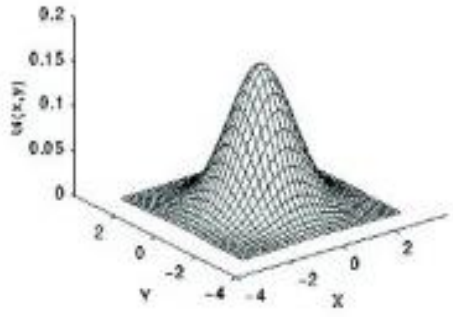
Ilustración 5. Imagen con filtro gaussiano con σ=4.0

Fuente: http://alojamientos.us.es/gtocoma/pid/tema3-1.pdf

2.2.2 Filtro Gaussiano Pasa Bajo

Este filtro es un operador bidimensional de convolución que se emplea para deshacer ruido y suavizar bordes, es decir, que es similar al filtro promediado pero utiliza una máscara diferente que representa la forma de una campana gaussiana, como se visualiza en la ilustración 6.

Ilustración 6. Campana gaussiana y su ecuación.



$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Fuente: http://dea.unsj.edu.ar/imagenes/recursos/capitulo3.pdf

La idea del suavizado gaussiano es utilizar la asignación gaussiana 2D como una función de desviación puntual, es decir que logra con la convolución. La imagen es almacenada como un conjunto de pixeles moderados en el cual se necesita producir una versión discreta aproximada a la función gaussiana. En general y en forma teóricamente, la distribución gaussiana es distinta de cero siempre, lo cual requeriría una máscara de convolución infinita, pero en práctica podemos suponer que se anual 3 veces la desviación estándar.

Ilustración 7. **Imagen filtrada con mascara 7x7 elementos e imagen con mascara de 3x3 elementos 3 veces.**

Fuente: http://dea.unsj.edu.ar/imagenes/recursos/capitulo3.pdf

El efecto del suavizado gaussiano es desdibujar una imagen, como lo hace el filtro de media como se visualiza en la ilustración anteriormente, también se puede decir que el grado de suavizado se determina a través del valor de la desviación estándar, es decir, que mientras más grande la desviación requiere una máscara de mayor tamaño.

2.3 Ruido

"El ruido en una imagen es un fenómeno muy frecuente que aparece de improvisto en la adquisición y/o transmisión por varias circunstancias, afectando a la calidad de la imagen. En el caso específico de las imágenes digitales lo definimos como los píxeles "erróneos", aleatorios que se entremezclan con los píxeles "aceptables" que componen la imagen y que

entorpecen su correcta reproducción. Reducir el ruido de una imagen es un tema que ha sido extensamente estudiado en el campo de procesamiento digital de imágenes". (Perez, 2015)

Existen distintos modelos de ruido, según las funciones de densidad de probabilidad que sigan sus intensidades r(x,y):

- **Ruido gaussiano**: Es aquel que modela el ruido provocado por los circuitos electrónicos o ruidos de los sensores por falta de iluminación y/o altas temperaturas. También se puede decir que la intensidad de todos los pixeles se ve afectada.
- **Ruido uniforme:** Este ruido se caracteriza por tomar valores en un determinado intervalo de forma equiprobable.
- **Ruido impulsivo:** Es aquella que se produce normalmente en la cuantificación que se realiza en el proceso de digitación.

Por otra parte, existen varios tipos de ruido en el cual el más común es el ruido gaussiano, "que aparece en el momento de la adquisición de la imagen por un dispositivo en malas condiciones, mala iluminación o altas temperaturas y tiene como consecuencia el emborronamiento de todos los píxeles de la imagen. En el ruido de tipo Gaussiano, la intensidad de todos y cada uno de los píxeles que componen la imagen se ven afectados y cambian su valor, de acuerdo con una distribución normal. Podría aplicarse otro tipo de distribución, sin embargo, se toma como modelo la gaussiana debido al teorema central del límite que nos asegura que la suma de diferentes ruidos tiende a aproximarse a una distribución normal". (Perez, 2015)

2.4 Normalización en el procesamiento de imágenes

Básicamente se refiere a una matriz en el que cada valor de la matriz se pueda dividir por la suma de los valores de la matriz esto con el fin de que la suma de los valores de la matriz sea igual a uno, siendo así que todos los valores sean mayores a 0. Por esta razón es eficaz porque una convolución entre una matriz de imagen y una matriz de nucleo concede una imagen de salida con valores entre 0 y el valor máximo de la imagen original. Por otra parte, se podría definir con dos sustentaciones, "el primero es "cortar" valores demasiado altos o demasiado bajos. es decir, si la matriz de imagen tiene valores negativos, uno los establece en cero y si la matriz de imagen tiene valores superiores al valor máximo, uno los

establece en valores máximos. El segundo es estirar linealmente todos los valores para ajustarlos al intervalo [0, valor máximo]". (Anonimo, 2015)

2.5 LUT de una imagen

Esta es la primera pregunta que suele surgir, o sea que vamos a resolverla juntos.

Un LUT, o Look up table, es un algoritmo que transforma unos parámetros de la imagen en otros. Generalmente estos parámetros son algunos de los siguientes:

- Contraste
- Saturación
- Tono
- Compresión de color

Obviamente no es necesario que un LUT cambie los ajustes de cada uno de estos parámetros, ya que con cambiar uno de ellos, ya es un LUT.

Pare resumir, un LUT es una tabla de valores que convierte los valores de entrada (aquellos que provienen de la cámara) en valores de salida (aquellos que queremos conseguir), dependiendo de la configuración del propio LUT (Mares, 2015)

3 ESTADO DE ARTE

Ante el problema de verificar una firma se propone un algoritmo que comprende las etapas de creación de la línea poligonal P, con vértices equiespaciados, sobre el trazo de la imagen de una firma, disponer de la línea poligonal sobre una firma a verificar para luego mediante el algoritmo Snake se pueda ajustar exactamente a la nueva imagen usando una medida de aumento de energía o de ajuste elástico referente a la deformación producida tras la adaptación. Ante distintas pruebas con diferentes combinaciones de valores en los parámetros determinantes en la prueba realizada a 25 individuos con 3 firmas auténticas cada

uno, se obtuvo un margen de error del 8% lo cual es muy bueno debido a la escasa base de datos con la que se trabajó (Vélez et al., 2005).

Con la implementación del Snake se puede obtener una correcta segmentación de imágenes médicas en la modalidad de endoscopía y ultrasonido, tanto para los que utilizan el gradiente de la imagen como los que aplican el flujo vectorial gradiente. Después de múltiples pruebas realizadas en imágenes médicas, se comprobó que es la herramienta correcta y eficiente; aportando mejores resultados en imágenes de endoscopía que en imágenes de ultrasonido, ya que estas últimas necesitan de un pre procesamiento mayor por el ruido que presentan (Herold y Escobedo, 2007).

Considerando que desarrollar nuevos algoritmos que faciliten una detección eficiente y automatizada del espacio global, no es una tarea sencilla. Se propone un algoritmo preprocesador que combina técnicas tradicionales como la umbralización y filtro de mediana basado en el modelo de contornos activos (Snake). Permitiendo distinguir dos niveles de intensidad, uno es el fondo y el otro es la glotis. Fundamentando su rendimiento a través de la validación de coeficiente Pratt y la comparación de una segmentación manual con otra automática basada en la transformada de watershed. Después de múltiples y exhaustivas pruebas se obtuvo al algoritmo Snake como el mejor preparado para dar seguimiento al proceso, lo cual puede ser muy beneficioso en la implementación de videos estroboscópicos; dejando de lado la propuesta del algoritmo watershed (Andrade, 2012).

Tabla 1. Estado de arte

TÍTULO	AUTOR	OBJETIVO	METODOLOGÍA	HALLAZGO	TIPO DE DOCUMENTO	LUGAR	ENLACE
Verificación Off-Line de Firmas Manuscritas: Una Propuesta basada en Snakes Paramétricos	José F. Vélez Ángel Sánchez Ana B. Moreno José L. Esteban	Verificar a falsificadores no entrenados, utilizando una firma en forma de imagen 2D bitonal obtenida	Investigaciones anteriores sobre el tema de Snake.	Un primer requisito de un sistema de verificación, utilizable en condiciones reales, consiste en que no es viable solicitar muchas firmas a cada usuario	Estudio universitario	Universidad Rey Juan Carlos, Madrid	https://www.researchgate.net/profile/Jose-Luis-Esteban/publication/237360001_Verificacion_Off-Line_de_Firmas_Manuscritas_Una_Propuesta_basada_en_Snakes_Parametricos/links/54a53fce0cf256bf8bb4c88f/Verificacion-Off-Line-de-Firmas-

				para construir el sistema.			Manuscritas-Una-Propuesta-basada-en-Snakes-Parametricos.pdf
Segmentación de imágenes médicas con la aplicación de Snakes	-Herold-García, Silena -Escobedo-Nicot, Miriela	Crear nuevos métodos que optimicen los existentes para realizar segmentación de imágenes	Investigaciones en páginas y libros	Los bordes que conforman la superficie potencial de una imagen corresponden a los bordes significativos de una imagen u objeto.	Tesis	Universidad de oriente.	https://www.redalyc.org/pdf/1813/181320170003.pdf
Automatic Pixel-Parallel Extraction of the Retinal Vascular Tree: Algorithm Design, On-Chip Implementation and Applications	Carmen Alonso Montes	la extracción del árbol arterio-venoso en imágenes digitales	Paginas e investigaciones hechas anteriormente.	La primera versión del algoritmo fue diseñada basándose en el paradigma CNN.	Tesis	Universidad Coruña	https://www.researchgate.net/publication/258225994_Automatic_Pixel-Parallel_Extraction_of_the_Retinal_Vascular_Tree_Algorithm_Design_On-Chip_Implementation_and_Applications
Segmentación de la glotis en imágenes laríngeas usando snakes.	Andrade Miranda, Gustavo Xavier	la detección automática del espacio glotal de imágenes laríngeas tomadas a partir de 15 vídeos grabados por el servicio ORL del hospital Gregorio Marañón de Madrid con luz estroboscópica	Fuentes bibliográficas de páginas investigadas.	se obtiene una imagen apropiada para el uso de las snakes. El valor escogido para el umbral es del 85% del pico máximo del histograma de la imagen; sobre este valor la información de los píxeles no es relevante.	Tesis	E.U.I.T Tele Comunicación (UPM) (Antigua Denominación)	http://oa.upm.es/13815/

Fuente: elaboración propia

4 JUSTIFICACIÓN

Se realizó la ejecución del lenguaje Java en la aplicación de la ecuación Snake, ejecutando la aplicación de Snake mediante Imagej, convirtiéndose en la modelación clave para el desarrollo del proyecto donde la Ingeniería ocupa un punto clave en la Rama de medicina

Con el modelo Snake lo que se quiere lograr es que la calidad y el aspecto de las imágenes médicas resalten lo más importante de ellas, logrando el mejoramiento de la apariencia de la imagen hacia el ojo humano resaltando los detalles más relevantes de la imagen y de esta manera mostrar las características médicas que se requiere para los exámenes de los pacientes mediante las imágenes.

El mejoramiento de la imagen en cuanto a su apariencia en contraste crece y se eliminaran efectos no deseados como lo pueden ser las distorsiones, efectos geométricos, efectos ópticos, mapeos, transformaciones de color (Escala de grises o color) y muchos más efectos que desfavorecen la imagen, logrando la segmentación de la imagen en medicina extrayendo los datos, mediciones e información que se logran con la implementación de la Ingeniería para dar la mejor solución computacional y óptica.

Mediante la ejecución del plugin Snake, se busca conseguir un mejor análisis de las imágenes médicas, las cuales con una mayor calidad y exactitud de procesamiento se logre resaltar lo más importante para el usuario, eliminando así, los efectos no deseados como lo pueden ser las distorsiones, efectos geométricos, efectos ópticos, mapeos, transformaciones de color y muchos más efectos que desfavorecen la imagen. De esta manera se lograría realizar dicho proceso de la mejor manera posible, con lo cual el ojo humano no siempre puede conseguir. Dicho análisis fundamenta su importancia en la extracción de datos, mediciones e información que se logran con la implementación de la Ingeniería para dar la mejor solución computacional y óptica en la medicina.

5 OBJETIVO

5.1 Objetivo general

Presentar formalmente la ecuación de Snake como un caso característico de la teoría de la regularización mediante el lenguaje Java ejecutando la aplicación de Snake mediante el software ImageJ.

5.2 Objetivo específicos

- Comprobar que el modelo funcione de manera útil y eficaz.
- Hacer uso de las herramientas plasmadas en clase sobre Procesamiento Digital de Imágenes en el caso del proyecto usando el modelo Snake.
- Realizar un prototipo de software implementando el funcionamiento de imágenes médicas mediante Snake.

6 MATERIALES

Para la elaboración e implementación del filtro se empleó el lenguaje Java brindándonos seguridad, fiabilidad y rapidez para la ejecución del mismo, además de elaborarlo con la intención de que sea de libre acceso en ImageJ, un software creado por Wayne Rasband y que según (González, 2018) nos dice que: “Es un programa de procesamiento de imágenes diseñado para imágenes científicas”. Además, se tomó en cuenta dos imágenes para el desarrollo y validación, las cuales son Lena y Dot Blot. Adicional a ello contamos con requisitos mínimos para realizar la ejecución con normalidad, ellos son: Windows 10, RAM de 128 mb, espacio en disco duro, una Pentium 2 a 266 Hz, exploradores como internet Explorer y superior.

7 MÉTODO

En el presente estudio se consideró la revisión de literatura científica los cuales se obtuvieron de portales web de revistas científicas y repositorios de tesis. Asimismo, se usaron búsquedas avanzadas con palabras clave como: Snake, curva elástica, segmentación, entre otros. Se examinaron dichos artículos para verificar que estos tengan información que ayude a implementar de la mejor manera el filtro. Es así, que se desarrolló bajo el paradigma POO (programación orientada a objetos), el manejo de clases especializadas para el manejo de los puntos (coordenadas), , para el Snake, es pide una serie de parámetros para trabajar con el ancho, tamaño y los puntos. Además, se empleó el filtro Gaussian para atiborrar imágenes y eliminar un posible ruido, sumándole a ello se aplicó la normalización de imágenes con la intención de realizar de buena forma el código correspondiente.

Lo primero que se elabora es el diseño de los correspondientes parámetros para usar durante la ejecución del filtro, dichos parámetros son, Radio del suavizado, El paso de integración de las líneas, las distancia que van a tener las líneas del Snake, definir si usar un solo color o no, El Lut de los colores, el controlador de la inversión del Lut, y por último el botón para definir si elimina o no los contornos abiertos. Estos atributos los definimos para que inicien con unos datos previamente establecidos y además que sea capaz de guardar los futuros cambios en el panel principal.

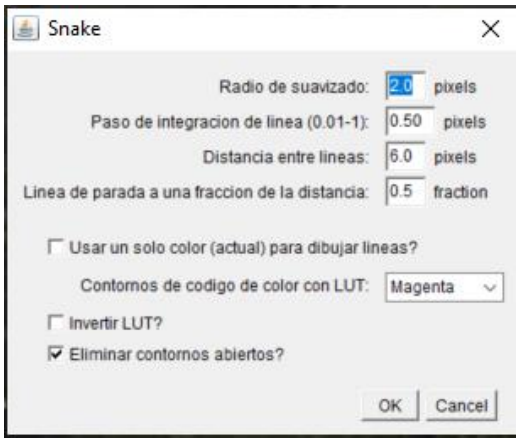


Figura 1. Panel de control de los parámetros

Justo después de declarar los parámetros para la hora de trabajar el Snake, se procede a construir los diversos atributos con el fin de crear las contenedoras necesarias para el desarrollo de cada uno de los contornos, los valores son los siguientes:

- Procesador actualmente activo
- Radio de suavizado utilizado para calcular el gradiente
- Derivadas de gaussiano (primer orden en x e y)
- Campo vectorial normalizado de gradiente
- Paso de tiempo de integración
- Cuadrícula de espacios ocupados
- Tamaño de la cuadrícula
- Nuevas coordenadas de puntos de semilla
- Distancia de separación entre curvas de nivel
- Detener la integración de la línea cuando está más cerca que * (dPararLineaSeparacion * distanciaSeparacion)
- Ignorar los contornos abiertos
- Tamaño de la grilla
- Tamaño de la imagen
- Nombre del LUT elegido
- Matriz de los colores o LUT
- Definidor si usar un solo color del LUT
- Color seleccionado de Selector de color / Barra de herramientas
- Superposición para renderizar
- Intensidades mínimas y máximas de la imagen tomadas de Contraste / Brillo
- Semilla inicial para la primera línea, punto con la pendiente mínima
- Número total de contornos

Con esto se puede desarrollar todo lo que se tiene planeado para el Snake.

Ahora lo que se procede a realizar el Run donde se inicializa las variables medidoras del tiempo y se procede a desarrollar el sistema para pedir los parámetros mostrados anterior mente, como se muestra a continuación:

```
// pedir parámetros

if(!showParametersDialog())
        return;

IJ.log("Filtro Snake ( contornos )");
IJ.log("Parametros:");
IJ.log("Radio de suavizado: "+Float.toString(dSuavizado)+" pixels");
IJ.log("Paso de integración de linea: "+Float.toString(dTiempoEjecucion)+" pixels");
IJ.log("Distancia entre lineas: "+Float.toString(distanciaSeparacion)+" pixels");
IJ.log("Linea de parada a una fracción de la distancia: "+Float.toString(dParar LineaSeparacion));
```

Figura 1.1 Código para pedir los atributos.

Después se procede a inicializar todos los valores necesarios como lo son el tamaño de la grilla, que se hace por la distancia de separación multiplicado por 0.5, luego se procede a calcular la derivada gaussiano.

```
/** Tamaño de la cuadrícula **/
dGridSize=0.5f*distanciaSeparacion;
nWG=(int)Math.ceil(nW/(float)dGridSize);
nHG=(int)Math.ceil(nH/(float)dGridSize);

Grid = new ArrayList[nWG][nHG];

/** Derivadas del cálculo gaussiano **/

double[][] kernel = computeKernelGaussian2D_du(dSuavizado,dSuavizado, 0.0f);
gaussianX= convolve(ip, kernel);

//new ImagePlus("dx", gaussianX).show();
kernel = computeKernelGaussian2D_dv(dSuavizado,dSuavizado, 0.0f);

gaussianY= convolve(ip, kernel);
//new ImagePlus("dy", gaussianY).show();
```

Figura 1.2 Cálculos e inicialización de los parámetros

Ahora se procede a calcular la normalización del campo de gradiente y configurar los colores de los contornos, como se muestra a continuación:

```
// normalizar el campo de gradiente

campoNormalizado  = new float [nW][nH][2];
fillNormalizedField();

image_overlay = new Overlay();

// configurar los colores de los contornos

if (bColorSimple)
        sistemaColor = Toolbar.getForegroundColor();
else
{
        getRGBLutTable();
        fInMin = (float) ip.getMin();
        fInMax=(float) ip.getMax();
        fInRango = fInMax -fInMin;

}
```

Figura 1.3 configuración de los colores y cálculo de la normalización.

Por consiguiente, se procede a elaborar el diseño de la primera línea del contorno, esto se hace por medio de una superponían, y se implementó una ventana donde se va contando las líneas creadas durante el proceso.

```
// primera linea

new_line=buildLine(iniSemilla.getX(),iniSemilla.getY());
if(new_line!=null)
{
        // agregar a la superposición

        image_overlay.add(new_line);
        nLineas++;
        IJ.log("Linea "+ Integer.toString(nLineas)+" Añadir.");
}

// Abrir la ventana
imp.setOverlay(image_overlay);
imp.updateAndRepaintWindow();
imp.show();
```

Figura 1.4 Creación de las líneas del Snake.

Para finalizar el método Run, se procede a diseñar un sistema de semillas de la primera línea de contorno, pasando por un proceso semillas de celdas vacías, y va pasando por cada

intento hasta que se pueda desarrollar, después se lanza el mensaje del tiempo que costo ejecutar todo el modelo del Snake.

```
// Utilizando semillas de la primera línea de contorno

if (generateLinesFromSeedList())
{
        // Comprobar semillas de celdas vacías

        refillSeedList();
        // ¡Otro intento! ¡Oye, oye!

        generateLinesFromSeedList();
}

IJ.log("Hecho");
contourTime = System.nanoTime() - startTime;
IJ.log("Tiempo Total: " + String.format("%.2f",((double)Math.abs(contourTime))*0.000000001) + " s");
```

Figura 1.5 Comprobación de las líneas y el cálculo del tiempo total.

Ahora procedemos a diseñar el modo de generar líneas usando la cola puntos Base, para hacer esto pasamos por los puntos de la cola de la semilla, para ir generando las líneas, luego se mira hasta que la cola este totalmente vacía, luego por medio de un ciclo se desarrolla las condiciones para verificar que la cola no este vacía, luego se hace otro condicional para poder evaluar el punto de semilla está demasiado cerca de otra línea de contorno, saca la semilla de la cola. Y luego se procede a mirar que el punto este bien, por medio de un else.

```
/** Generar líneas usando la cola puntosBase **/
boolean generateLinesFromSeedList()
{

        // Pasando por la cola de puntos semilla
        // Generado por líneas

        PolygonRoi new_line;
        boolean goOn=true;
        boolean bFoundSeed;
        Float [] seed_point = new Float [2];

        // Hasta que la cola esté vacía

        while (goOn)
        {
                if(puntosBase.size()==0)
                {
                        goOn=false;
                }
```

```
while (!bFoundSeed)
{
        // La cola está vacía, terminar

        if(puntosBase.size()==0)
        {
                goOn = false;
                break;
        }
        seed_point= puntosBase.get(0);
```

```
        seed_point= puntosBase.get(0);

        if(isBusy(seed_point[0],seed_point[1],0.9f*distanciaSeparacion))
        {
                // El punto de semilla está demasiado cerca de otra línea de contorno,
                // saca la semilla de la cola

                puntosBase.remove(0);
        }
        else
        {
                // El punto de semilla está bien

                bFoundSeed=true;
                puntosBase.remove(0);
        }
}
```

Figura 1.6 Método para generar el listado de las semillas.

Por consiguiente, el método, consiste en generar una línea de contorno desde el punto inicial, donde seguirá evaluando por medio de condicionales, si la línea está o no, hasta que se va añadiendo en cada uno de los métodos y lanzamos un mensaje de que la imagen fue cerrada con éxito.

```
// Generar una línea de contorno desde el punto inicial

if(goOn && bFoundSeed)
{
        new_line=buildLine(seed_point[0],seed_point[1]);
        if (new_line!=null)
        {
                if(new_line.size()>2)
                {
                        image_overlay.add(new_line);

                        nLineas++;
                        imp.setOverlay(image_overlay);
                        imp.updateAndRepaintWindow();
                        imp.show();
                        IJ.log("Linea "+ Integer.toString(nLineas)+" Añadir.");
                        if(!imp.isProcessor())
                        {
                                IJ.log("Imagen cerrada. Terminando.");
                                return false;
                        }
                }
        }
```

Figura 1.7 Generador de la línea en el punto inicial.

El siguiente paso es crear la rutina de integración de la línea para esto se utiliza el método de Newton simple con un paso definido por el usuario (paso de integración), esto

funciona dentro del método BuildLine donde la auto intersecciones es igual al tamaño de la imagen. La precisión de más de 1 píxel parece innecesaria.

```
public PolygonRoi buildLine(float xstart, float ystart)
{
        /** Propia cuadrícula de la línea actual para verificar si hay auto-intersecciones.
         * Es igual al tamaño de la imagen. La precisión de más de 1 píxel parece innecesaria **/
        int [][] ownGrid  = new int [nW][nH];

        PolygonRoi polyline;
        float xcurr, ycurr;
        float xvel,yvel;
        ArrayList<Float> px = new ArrayList<Float>();
        ArrayList<Float> py = new ArrayList<Float>();

        int gx, gy;
        int gxn, gyn;
        boolean bEnd;

        px.add(xstart);
        py.add(ystart);
        int nDirection;
```

Figura 1.8 BuildLine para la creación de la línea.

Por consiguiente, se procede a desarrollar un ciclo para que las líneas crecer hacia adelante y hacia atrás como una variable específica, luego se procede a marcar el pixel osngrid como una posición actual, lo que significa que podemos hacer crecer la línea dentro de ella, con esto se puede construir el gradiente en la posición actual siendo + o – dependiendo de la dirección, luego se comprueba que la fuente sea un sumidero o cero pixeles.

```
for(nDirection=1;nDirection>-2;nDirection-=2)
{

        bEnd = false;
        xcurr=xstart;
        ycurr=ystart;

        // Marcar el pixel OwnGrid como actual (1)
        // Significa que podemos hacer crecer la linea dentro de ella
        gx = (int) Math.floor(xstart);
        gy = (int) Math.floor(ystart);
        ownGrid[gx][gy]=1;

        while (!bEnd)
        {
                // Gradiente en la posición actual (+/- depende de nDirection)

                xvel = ((float)nDirection)*campoNormalizado[Math.round(xcurr)][Math.round(ycurr)][0];
                yvel = ((float)nDirection)*campoNormalizado[Math.round(xcurr)][Math.round(ycurr)][1];

                // Compruebe que no sea un sumidero o una fuente o simplemente cero píxeles

                if(Float.isNaN((Float)xvel))
                {
                        bEnd=true;
                        ownGrid[gx][gy]=2;
                }
```

```
else
{
if(Float.isNaN((Float)xcurr))
{
        bEnd = true;
        ownGrid[gx][gy]=2;
}
else
{
        // Hacer crecer una línea por paso de integración
        xcurr = xcurr +xvel * dTiempoEjecucion;
        ycurr = ycurr +yvel * dTiempoEjecucion;

        // La línea está cerca de otra línea, abortar el crecimiento
        if(isBusy(xcurr,ycurr,distanciaSeparacion*dPararLineaSeparacion))
        {
                bEnd = true;
                ownGrid[gx][gy]=2;
        }
        else
        {
                // Revisemos la auto-intersección
                gxn = (int) Math.floor(xcurr);
                gyn = (int) Math.floor(ycurr);
                // Fuera de imagen, abortar!
                if(gxn<0 || gyn<0 || gxn>(nW-1)|| gyn>(nH-1))
                {
                        bEnd = true;
                }
                else
                {
```

Figura 1.9 creador del gradiente

Luego sigue con el proceso verificando la auto intersección, por lo que el pixel píxel de "crecimiento" actual vamos a marcarlo con (2) = visitado y marcar el nuevo píxel como actual (1), luego se procede a agregar un punto al final de la matriz de líneas (crecimiento hacia

adelante), por último, se crea un punto al comienzo de la matriz de líneas (crecimiento hacia atrás) de esta manera los puntos en la matriz se indexan continuamente.

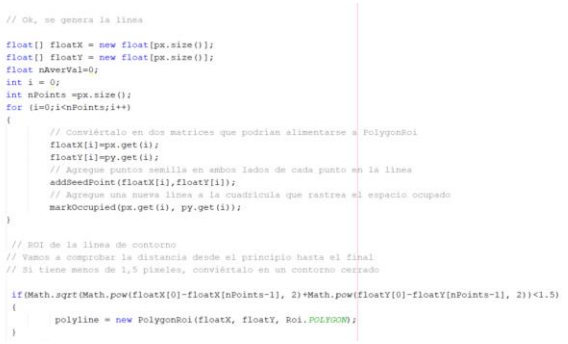
```
else
{
        // Salimos del pixel de "crecimiento" actual
        // Vamos a marcarlo con (2) = visitado
        // y marcar el nuevo pixel como actual (1)

        if(gxn!=gx || gyn!=gy)
        {
                ownGrid[gx][gy]=2;
                gx=gxn;
                gy=gyn;
                ownGrid[gx][gy]=1;
        }
        // agregar un punto al final de la matriz de líneas (crecimiento hacia adelante)
        if(nDirection==1)
        {
                px.add(xcurr);
                py.add(ycurr);

        }
        // Agregar un punto al comienzo de la matriz de líneas (crecimiento hacia atrás)
        // de esta manera los puntos en la matriz se indexan continuamente
        else
        {
                px.add(0,xcurr);
                py.add(0,ycurr);
        }
```

Figura 1.10 agregar puntos en las dos direcciones

Para finalizar se crean las polilíneas para garantizar que todo esté funcionando correctamente.



```
// Ok, se genera la linea

float[] floatX = new float[px.size()];
float[] floatY = new float[px.size()];
float nAverVal=0;
int i = 0;
int nPoints =px.size();
for (i=0;i<nPoints;i++)
{
        // Conviértalo en dos matrices que podrían alimentarse a PolygonRoi
        floatX[i]=px.get(i);
        floatY[i]=py.get(i);
        // Agregue puntos semilla en ambos lados de cada punto en la línea
        addSeedPoint(floatX[i],floatY[i]);
        // Agregue una nueva línea a la cuadrícula que rastrea el espacio ocupado
        markOccupied(px.get(i), py.get(i));
}

// ROI de la línea de contorno
// Vamos a comprobar la distancia desde el principio hasta el final
// Si tiene menos de 1,5 píxeles, conviértalo en un contorno cerrado

if(Math.sqrt(Math.pow(floatX[0]-floatX[nPoints-1], 2)+Math.pow(floatY[0]-floatY[nPoints-1], 2))<1.5)
{
        polyline = new PolygonRoi(floatX, floatY, Roi.POLYGON);
}
```

```
if(Math.sqrt(Math.pow(floatX[0]-floatX[nPoints-1], 2)+Math.pow(floatY[0]-floatY[nPoints-1], 2))<1.5)
{
        polyline = new PolygonRoi(floatX, floatY, Roi.POLYGON);
}
else //polyline
{
        if(bEliminarAbierto)
        {       return null;}
        else
        (polyline = new PolygonRoi(floatX, floatY, Roi.POLYLINE);)
}
// bueno, de un solo color

if(bColorSimple)
{
        polyline.setStrokeColor(sistemaColor);
}
// Uso de colores LUT para codificar la profundidad del color

else
{
        // Calcular la intensidad media por línea

        nAverVal=0;
        for (i=0;i<nPoints;i++)
        {
                nAverVal += ip.getf((int)Math.floor(floatX[i]), (int)Math.floor(floatY[i]));
        }
        nAverVal/=px.size();
```

```
        for (i=0;i<nPoints;i++)
        {
                nAverVal += ip.getf((int)Math.floor(floatX[i]), (int)Math.floor(floatY[i]));
        }
        nAverVal/=px.size();

        // Elegir la configuración actual de contraste / brillo de la imagen

        if(nAverVal>fInMax)
                nAverVal=255;
        else
        {
                if(nAverVal<fInMin)
                        nAverVal=0;
                else
                        nAverVal=Math.round(255.0f*(nAverVal-fInMin)/fInRango);
        }

        polyline.setStrokeColor(new Color(RGBLutTabla[(int)nAverVal][0],RGBLutTabla[(int)nAverVal][1],RGBLutTabla[(int)nAverVal][
}

/polyline.setStrokeWidth(dStrokeWidth);

eturn polyline;
```

Figura 1.11. Polilínea final del método build

El paso a seguir es desarrollar un punto dentro de la variable, y la grilla, para esto se debe rastrear la densidad local de las curvas de nivel en la correspondiente imagen previamente seleccionada, luego se le agrega más datos.

```
/** La función agrega un punto a la variable Grid
 * que rastrean la densidad local de las curvas de nivel en la imagen **/
void markOccupied(float xin, float yin)
{
        int gx,gy;

        ArrayList<Point> currarr;

        //grid's cell coordinate
        gx = (int) Math.floor(xin/dGridSize);
        gy = (int) Math.floor(yin/dGridSize);
        currarr =(ArrayList<Point>) Grid[gx][gy];
        if(currarr==null)
        {
                Grid[gx][gy] = new ArrayList<Point>();
                currarr =(ArrayList<Point>) Grid[gx][gy];
        }
        currarr.add(new Point(xin,yin));
}
```

Figura 1.12 Agregar un punto en grid

El siguiente método consiste en Agregar dos puntos de semilla a la cola puntos Base (si no están en una forma de otra línea, es decir, más lejos que el valor distanciaSeparacion) El siguiente método consiste en añadir un punto.

```
float xn,yn;
float xvel, yvel;

Float[] spoint;
int nDirection;
nDirection =1;
for(nDirection=1;nDirection>-2;nDirection-=2)
{
        // Dirección perpendicular
        xvel = campoNormalizado[Math.round(xs)][Math.round(ys)][1]*((float)nDirection);
        yvel = -campoNormalizado[Math.round(xs)][Math.round(ys)][0]*((float)nDirection);
        // Punto especial (plano o fuente o fregadero)
        if(Float.isNaN((Float)xvel) ||Float.isNaN((Float)yvel))
        {
                return;
        }
        xn=xs+xvel*distanciaSeparacion*1.1f;
        yn=ys+yvel*distanciaSeparacion*1.1f;

        if(xn<0 || yn<0 || xn>(nW-1)|| yn>(nH-1))
        {
        }
        else
        {
                spoint = new Float [2];
                spoint[0]=xn;
                spoint[1]=yn;
                puntosBase.add(spoint);
        }
}
```

Figura 1.13 método para añadir un punto

El siguiente método consiste en la función de generar puntos de semilla adicionales en las celdas de las gillas donde no hay puntos, esto es con el fin de llenar toda la imagen, para hacer esto se necesita las coordenadas en los puntos x e y para poder después limpiar las celdas y por último agregarlas.

```
/** La función genera puntos de semilla adicionales en las celdas de Grid.
 * donde no hay puntos (para llenar la imagen completa **/
void refillSeedList()
{
        int gx,gy;
        ArrayList<Point> currarr;
        Float[] spoint;
        //coordenada de la celda de la cuadrícula
        for (gx=0;gx<nWG;gx++)
                for (gy=0;gy<nHG;gy++)
                {
                        currarr =(ArrayList<Point>) Grid[gx][gy];
                        //Limpiar celdas
                        if(currarr==null)
                        {
                                // Agregue un punto de semilla en el medio de la celda
                                spoint = new Float [2];
                                spoint[0]=((float)gx)*dGridSize+0.5f*dGridSize;
                                spoint[1]=((float)gy)*dGridSize+0.5f*dGridSize;
                                puntosBase.add(spoint);
                        }
                }
}
```

Figura 1.14 Método rellenar la lista de semillas

Un paso importante para la efectividad del Snake es comprobar si el punto xc, yc está cerca de cualquier otro existente punto (almacenados en Grid) por la distancia cDist, esto se hace por medio del cálculo de que si un dato esta por fuera o no de la imagen y una comprobación de las celdas vacías por medio de los rangos.

```
public boolean isBusy(float xc, float yc, float dDist)
{
        int gx, gy;
        int x,y;
        int i,j,k;
        ArrayList<Point> currarr;
        Point point;
        int nRange=(int) Math.ceil(dDist/dGridSize);
        x= Math.round(xc);
        y= Math.round(yc);
        // fuera de imagen
        if(x<0 || y<0 || x>(nW-1)|| y>(nH-1))
        {
                return true;
        }
        // indice de celda de cuadricula
        gx = (int) Math.floor(xc/dGridSize);
        gy = (int) Math.floor(yc/dGridSize);
        // Comprobando las celdas vecinas también
        for (i=gx-nRange;i<=gx+nRange;i++)
                for (j=gy-nRange;j<=gy+nRange;j++)
                {

                    currarr = (ArrayList<Point>) Grid[i][j];
                    if (currarr == null) {

                    } else {
                        for (k = 0; k < currarr.size(); k++) {
                            point = currarr.get(k);
                            if (point.distance(xc, yc) < dDist) {
                                return true;
                            }
```

Figura 1.15 comprobación de la línea.

Para finalizar se trabajaron los métodos vistos en clase como lo son la normalización de una imagen y el filtrado de la información.

8 DISCUSIÓN

Mediante las pruebas realizadas con nuestro plugin Snake en distintas imágenes se obtuvo buen manejo del método dé snakes y sus respectivas técnicas y herramientas debido a que se obtuvieron resultados satisfactorios, ya que la segmentación de las imágenes digitales en la modalidad de endoscopia y ultrasonido es la correcta, por lo que se comparte la agradable experiencia con otros proyectos ya anteriormente realizados; donde después de múltiples y exhaustivas pruebas el algoritmo Snake es considerado el mejor preparado para dar seguimiento al proceso de detección eficiente y automatizada del espacio glotal, lo cual puede ser muy beneficioso en la implementación de videos estroboscópicos; así mismo para una correcta segmentación de imágenes médicas en la modalidad de endoscopía y ultrasonido y finalmente puede ser plasmado en otras ramas como para la autenticación de firmas.

A pesar de las distintas limitaciones que se presentaron como la poca información en la web, la adaptación de lenguajes y de imágenes para las respectivas pruebas se obtuvieron resultados positivos que favorecen el análisis de imágenes médicas donde a través de una

mayor calidad y exactitud de procesamiento se resalta lo más importante para el usuario, eliminando así, los efectos no deseados; contribuyendo en la extracción de datos, mediciones e información que se implementa en la Ingeniería para dar la mejor solución computacional y óptica en la medicina u otras ramas similares.

Con las pruebas realizadas se comprobó que la herramienta es correcta y eficiente. Además, se realizaron comparaciones con otras fuentes relacionadas con Snake y se puede decir que con la base brindada por otros autores se obtuvieron resultados similares y satisfactorios.

9 RESULTADOS

- La técnica de snakes es fundamental en muchos casos de la Ingeniería biomédica para la detección de contornos y partes importantes en los exámenes médicos.
- Con la implementación del Snake se puede obtener una correcta segmentación de imágenes médicas en la modalidad de endoscopía y ultrasonido.
- Después de múltiples pruebas realizadas en imágenes médicas, se comprobó que es la herramienta correcta y eficiente; aportando mejores resultados en imágenes de endoscopía que en imágenes de ultrasonido, ya que estas últimas necesitan de un preprocesamiento mayor por el ruido que presentan.
- Mejoramiento del aspecto de las imágenes y buen resultado de las imágenes sintéticas y de resonancia.
- Con el objetivo de evaluar y comprobar el filtro desarrollado se creyó conveniente hacer uso de una aplicación para el procesamiento de imágenes con snakes. La figura 2.1 se muestra una imagen de la aplicación.
- A continuación, se tendrán algunas de las imágenes digitales diferentes del presente proyecto de procesamiento digital de imágenes en nuestro caso estas imágenes fueron con las que se elaboró el método Snakes con el respectivo uso de la aplicación ImagineJ y Pluggin.

Figura 2.1 – interfaz de ImageJ

Esta aplicación llamada ImageJ es una UI muy fácil de usar. Al aplicar el filtro a una imagen podemos apreciar que aparece otra interfaz como notamos en la figura 2.2, con la intención de poder asignarle ciertos parámetros como lo es el radio suavizado para notar la convolución de la imagen por medio de la derivada de Gaussiano implementándolo en las coordenadas X, Y.

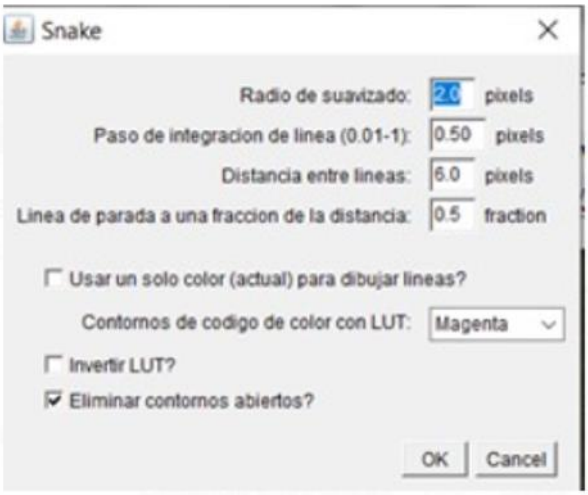


Imagen 2.2 – interfaz que aparece para realizar los ajustes a la imagen

Estas son otras de las figuras que se tomaron en cuenta y se llevaron a cabo en la elaboración del proyecto de procesamiento digital de Imágenes en el área de la medicina, en estas imágenes se puede observar, la disminución de mapeos, el resalte de bordes y se destacan de la imagen las partes más necesarias para las imágenes médicas que en este caso son exámenes médicos.

Al realizar la configuración con un radio suavizado, se obtuvo como resultado la figura 2.3 donde se calcularon todos los pixeles y cambiándolos a cada uno de ellos, hallándose todos los contornos que puede conseguir la imagen de forma más precisa, aunque la ejecución tome más tiempo.

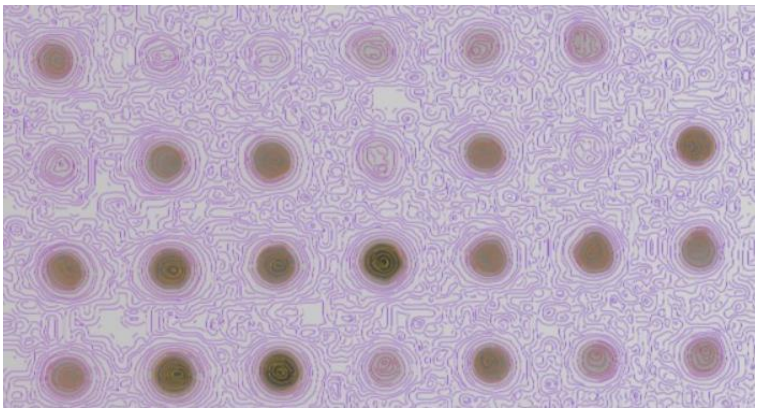


Figura 2.3- Radio de suavizado

En cambio, al realizar el ajuste de la siguiente imagen digital se obtuvo incluso resultados más precisos y en menor tiempo de ejecución mostrándose tal cual como en la figura 2.4.

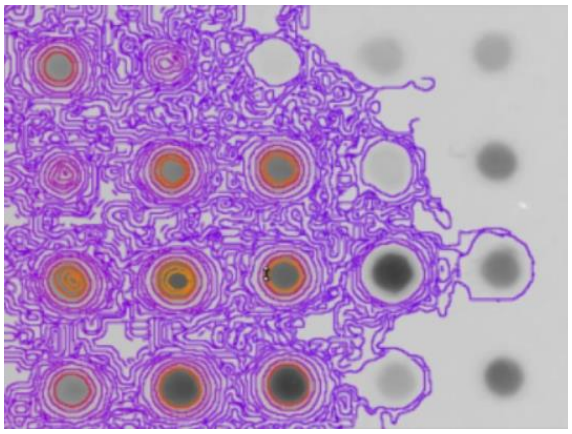


Figura 2.4- Paso de integración de línea

Mientras que, con un arreglo entre la distancia de las líneas, haciendo notar que las curvaturas sean menos exactas pero que ninguna genere la unión entre ellas como se visualiza en la figura 2.5, demostrando así que el Snake funciona correctamente.

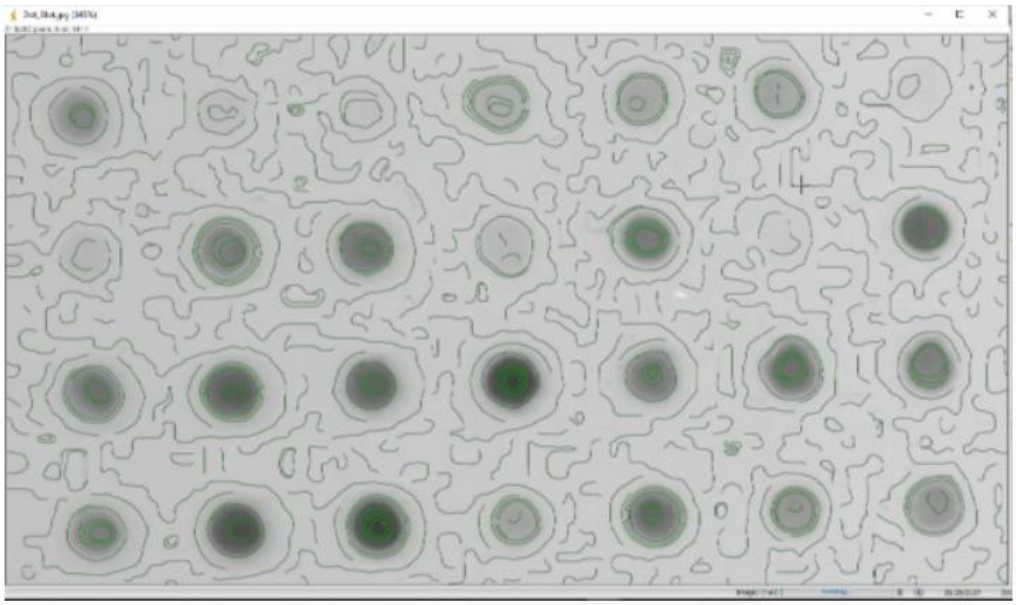


Figura 2.5- Distancia entre líneas

Además, a través de las líneas de abstracción de distancia y realizando modificaciones, obtuvimos un resultado distinto, así como podemos ver en la figura 2.6, tomando más de tiempo al momento de ejecución y notando que la separación entre líneas es corta.

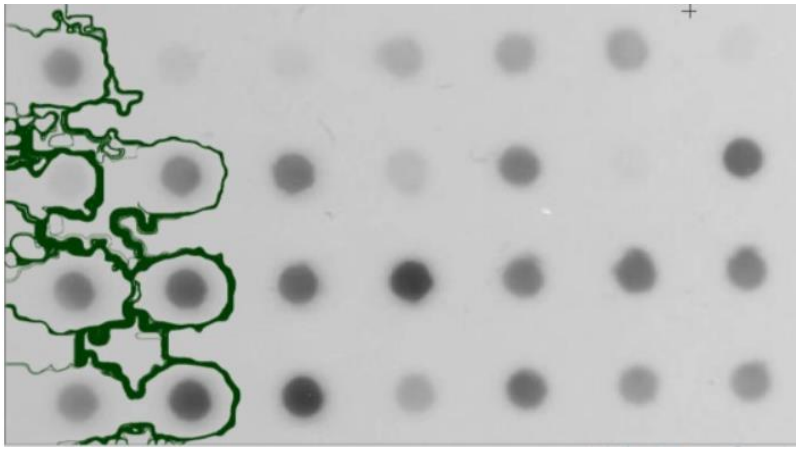


Figura 2.6. Línea de parada a una fracción de la distancia

Esta es la encargada de eliminar todas las líneas del método Snake, esto se realiza para que las líneas se unan unas con otras, como se puede observar en la imagen lo que se forma es una línea continua, en todas entonces el comienza a buscar, entonces se pregunta: ¿esta línea esta junta a otra?, si es verdad la deja quieta, si la respuesta es no, entonces la elimina, se realiza esto en toda la imagen hasta que solamente deja las Juntas. (Es la encargada de manejar el distanciamiento necesario para apreciar de la mejor manera cada uno de los detalles resaltados en la imagen)

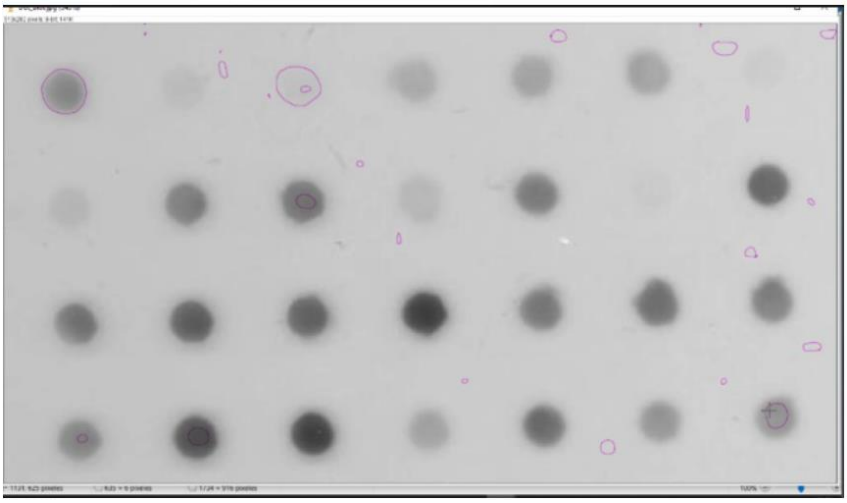


Figura 2.7 Eliminar contornos abiertos

Esta lo que hace es coger todos los Snakes y los multiplica por RBG, los colores que más resaltan la imagen son: red, blue y Green, debido a que se le quieren dar colores que resalten en la imagen, que sean notorios. En caso de que quiera que salga a blanco y negro se debe invertir el LUT en la tabla de parámetros para lograrlo.

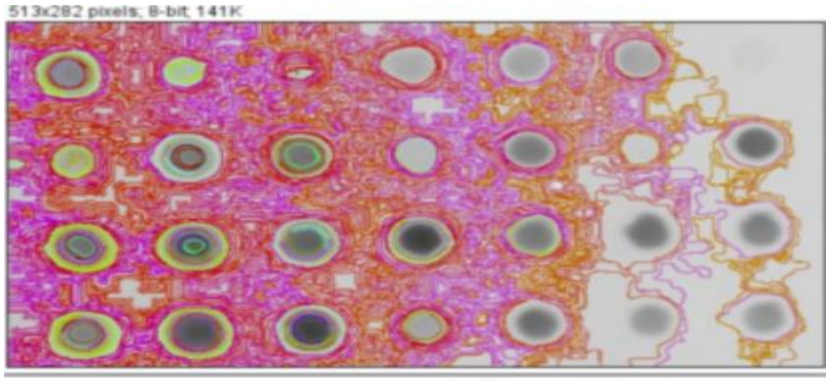


Figura 2.8 Usar tablas de búsqueda de colores (LUT)

En esta imagen digital se define que todas las líneas Snake salgan con un color en específico como se puede observar en la imagen, en este caso Cyan y también funciona por medio del RBG, los cuales funcionan con los colores primarios en todas las imágenes, se realiza este proceso para poder sacar el Snake.

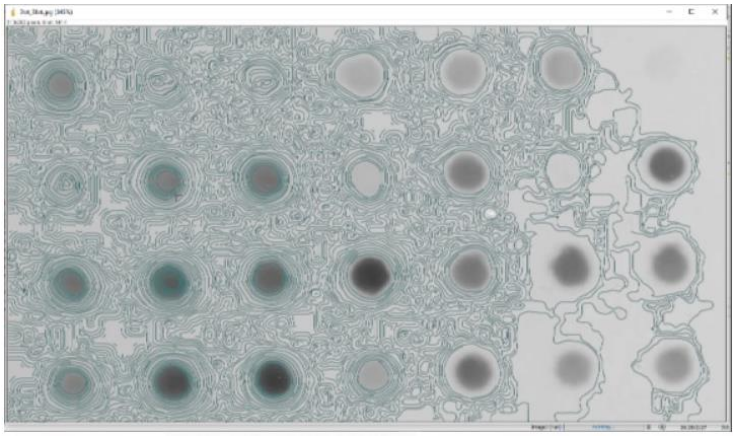


Figura 2.8 Usar un solo color

10 CONCLUSIONES

1. Con el prototipo de software implementado se puede obtener la segmentación de imágenes médicas de la modalidad de endoscopia y ultrasonido aplicando la modelación de snakes. También se puede decir que la implementación de esta técnica es útil y eficiente para el procesamiento de imágenes digitales.
2. Se observó que la técnica de snakes es fundamental para muchos casos de la ingeniería biomédica en importantes centros científicos de investigación del exterior, donde su principal aplicación es para la detección automática de contornos o bordes en imágenes médicas.
3. Al implementarse los cuatro tipos de snakes, dos de ellos rinde de manera positivo para imágenes sintéticas y también de resonancia magnética, en el cual se utiliza en primera instancia la técnica de suavización de imágenes de Gauss y luego calcula la energía externa utilizando los niveles máximos de gradiente de la imagen.

4. Con ayuda de las herramientas vistas en clase, se puede decir que tuvimos un resultado eficaz y positivo en nuestra modelación ya que se obtuvo un mayor rendimiento del Snake.

11 BIBLIOGRAFIA

[1] Andrade, G. (2012) Segmentación de la glotis en imágenes laríngeas usando snakes. Tesis (Master) https://oa.upm.es/13815/

[2] González, A. (2018). ImageJ: una herramienta indispensable para medir el mundo biológico. Repositorio Institucional CONICET Digital. https://ri.conicet.gov.ar/handle/11336/82733

[3] Vélez, J., Sánchez, A., Moreno, A. y Esteban J. (2005) Verificación Off-Line de Firmas Manuscritas: Una Propuesta basada en Snakes Paramétricos https://www.researchgate.net/profile/Jose-Luis-Esteban/publication/237360001_Verificacion_Off-Line_de_Firmas_Manuscritas_Una_Propuesta_basada_en_Snakes_Parametricos/links/54a53fce0cf256bf8bb4c88f/Verificacion-Off-Line-de-Firmas-Manuscritas-Una-Propuesta-basada-en-Snakes-Parametricos.pdf

[4] Herold, S. y Escobedo, M. (2007) Segmentación de imágenes médicas con la aplicación de Snakes Crear nuevos métodos que optimicen los existentes para realizar segmentación de imágenes https://www.redalyc.org/pdf/1813/181320170003.pdf

[5] Website: file:///C:/Users/PC/Downloads/osot%20(1).pdf recuperado: 28 de Octubre de 2021

[6] Rogers F.R., Adams A.J. Mathematical elements for computer graphics. 2a. edición. Mc Grawhill.

[7] (Filtros, s.f.) recuperado: http://alojamientos.us.es/gtocoma/pid/tema3-1.pdf

[8] (Perez, 2015) recuperado: https://m.riunet.upv.es/bitstream/handle/10251/73099/TFM_Cristina_P%C3%A9rez_Benito.pdf?sequence=1&isAllowed=y http://dea.unsj.edu.ar/imagenes/recursos/capitulo3.pdf

[9] Website: https://www.it-swarm-es.com/es/image-processing/normalizacion-en-el-procesamiento-de-imagenes/1056738917/ recuperado: 9 de noviembre de 2015

[10] (Mares, 2015) recuperado: https://www.chamanexperience.com/video/que-es-un-lut/#que_es_un_lut