UNSA
Universidad Nacional de San Agustín

FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS

ESCUELA PROFESIONAL INGENIERÍA DE SISTEMAS

Proyecto Final

Informe técnico

Calidad de Software - Gestión de Proyectos de Software

Docente:

Ing. Jesús Martín Silva Fernandez

Integrantes:

- Diaz Portilla Carlo Rodrigo
- Mamani Cañari Gabriel Antony
- Ordoño Poma Gustavo Eduardo

AREQUIPA - PERÚ

2024

Índice

1. Gradient Vector Flow (GVF) y Balloon Model..... 2
2. Chan–Vese Segmentation..... 6

Informe técnico

1. Gradient Vector Flow (GVF) y Balloon Model

1.1. Definición de Kernels Sobel y Laplaciano

```python
SOBEL_X = np.array([[1.0, 2.0, 1.0], [0.0, 0.0, 0.0], [-1.0, -2.0, -1.0]])
SOBEL_Y = np.array([[1.0, 0.0, -1.0], [2.0, 0.0, -2.0], [1.0, 0.0, -1.0]])
LAPLACIAN = np.array([[0.0, 1.0, 0.0], [1.0, -4.0, 1.0], [0.0, 1.0, 0.0]])
```

Se definen matrices para aplicar operadores Sobel y Laplaciano:

- Sobel detecta bordes al aproximar derivadas en las direcciones
- Laplaciano mide la curvatura de la imagen, crucial para suavizar fuerzas y regularizar la forma del campo vectorial.

1.2. Cálculo del Laplaciano

```python
def _calcLaplacian(x):
    """Aplica el operador Laplaciano"""
    return convolve(x, LAPLACIAN, mode="nearest")
```

Esta función aplica el operador Laplaciano mediante convolución. Su objetivo es suavizar variaciones bruscas en la imagen o en el campo vectorial durante el cálculo del GVF. El Laplaciano se usa en la difusión para extender la información del gradiente hacia regiones homogéneas.

1.3. Cálculo del Gradiente

```python
def _calcGradient(x):
    """Calcula el gradiente de la imagen"""
```

```
    pad = np.pad(x, pad_width=1, mode="edge")
    gradx = 0.5 * (pad[2:, 1:-1] - pad[:-2, 1:-1])
    grady = 0.5 * (pad[1:-1, 2:] - pad[1:-1, :-2])
    return (gradx, grady)
```

El gradiente se calcula como la diferencia central en las direcciones x e y. Este gradiente representa los cambios de intensidad en la imagen y es la base para generar el flujo vectorial de gradiente (GVF).

1.4. Generación del Mapa de Bordes

Python
```
def _getEdgeMap(u, sigma=2):
    """Genera el mapa de bordes usando Sobel"""
    u = gaussian_filter(u, sigma=sigma)  # Suaviza la imagen
    fx = convolve(u, SOBEL_X, mode="nearest")
    fy = convolve(u, SOBEL_Y, mode="nearest")
    return np.sqrt(0.5 * (fx**2.0 + fy**2.0))
```

El mapa de bordes se genera aplicando un filtro gaussiano para suavizar la imagen y luego calculando el gradiente mediante operadores Sobel. La magnitud del gradiente $\sqrt{f_x^2 + f_y^2}$ destaca las regiones con cambios de intensidad significativos.

1.5. Definición del Constructor para Configuración Inicial

Python
```
def __init__(self, alpha=0.01, beta=0.1, gamma=0.01, maxIter=1000,
maxDispl=1.0, eps=0.1, period=10):
    self.alpha = alpha  # Continuidad del contorno
    self.beta = beta  # Suavidad del contorno
    self.gamma = gamma  # Paso en el tiempo
    self.maxIter = int(maxIter)  # Iteraciones máximas
    self.maxDispl = float(maxDispl)  # Desplazamiento máximo
    self.eps = eps  # Tolerancia para convergencia
    self.period = period  # Historial de energía
```

El constructor define los parámetros del snake:

- α: Controla la continuidad del contorno.
- β: Favorece la suavidad.
- γ: Paso temporal para la iteración numérica.

Estos parámetros afectan cómo el contorno se ajusta a la forma deseada.

1.6. Cálculo del Flujo Vectorial de Gradiente

```python
def getGradientVectorFlow(self, fx, fy, CLF=0.25, mu=1.0, dx=1.0, dy=1.0,
maxIter=1000, eps=1.0e-6):
    dt = CLF * dx * dy / mu
    b = fx**2.0 + fy**2.0
    c1, c2 = b * fx, b * fy
    currGVF = (fx, fy)
    for i in range(maxIter):
        nextGVF = (
            (1.0 - b * dt) * currGVF[0] + CLF * _calcLaplacian(currGVF[0]) +
c1 * dt,
            (1.0 - b * dt) * currGVF[1] + CLF * _calcLaplacian(currGVF[1]) +
c2 * dt,
        )
        delta = np.sqrt((currGVF[0] - nextGVF[0]) ** 2.0 + (currGVF[1] -
nextGVF[1]) ** 2.0)
        if np.mean(delta) < eps:
            break
        currGVF = nextGVF
    return currGVF
```

El flujo vectorial de gradiente se calcula resolviendo iterativamente:

$\frac{\partial v}{\partial t} = \mu\nabla^2 v - |\nabla f|^2 (v - \nabla f)$

donde:

- $\mu\nabla^2 v$: Suaviza el campo vectorial.
- $|\nabla f|^2 (v - \nabla f)$: Alinea el flujo con los bordes detectados.

1.7. Normalización de la Imagen

```python
image = (image - image.min()) / (image.max() - image.min())
```

La imagen se normaliza al rango [0, 1] para garantizar estabilidad numérica y uniformidad en los cálculos del gradiente y flujo vectorial.

1.8. Generación del Contorno Inicial

```python
contour = np.array(getContour(seed), dtype=np.float32)
tck, _ = splprep(contour.T, s=0)
snake = splev(np.linspace(0, 1, 2 * len(contour)), tck)
snake = np.array(snake).T.astype(np.float32)
```

El contorno inicial se obtiene a partir de una región semilla. Se interpolan puntos para formar un contorno continuo, que será deformado por el snake.

1.9. Optimización del Contorno

```python
for step in range(self.maxIter):
    fx = xinterp(xx, yy, grid=False)
    fy = yinterp(xx, yy, grid=False)
    xn = inv @ (self.gamma * xx + fx)
    yn = inv @ (self.gamma * yy + fy)
    dx = self.maxDispl * np.tanh(xn - xx)
    dy = self.maxDispl * np.tanh(yn - yy)
    xx, yy = xx + dx, yy + dy
```

El contorno se optimiza iterativamente combinando:

- Fuerzas internas: Suavizan y mantienen la continuidad del contorno.
- Fuerzas externas: Atraen el contorno hacia los bordes utilizando el GVF.

1.10. Representación del Resultado

```python
seed = cv2.fillConvexPoly(seed, points=snake.astype(int), color=1)
```

Finalmente, el contorno ajustado se representa en un mapa binario donde las regiones segmentadas están marcadas como 1.

2. Chan–Vese Segmentation

2.1. Importación de Bibliotecas

```python
import sys
import cv2
import math
import matplotlib.pyplot as plt
import numpy as np
```

Este bloque importa las bibliotecas necesarias para el procesamiento de imágenes (cv2), cálculos matemáticos (math, numpy), y visualización (matplotlib.pyplot). Estas herramientas son fundamentales para la implementación del algoritmo de Chan-Vese y el análisis visual de su evolución.

2.2. Carga e Inicialización de la Imagen

```python
Image = cv2.imread(filepath, 1)
image = cv2.cvtColor(Image, cv2.COLOR_BGR2GRAY)
img = np.array(image, dtype=np.float64)
```

Se carga una imagen desde un archivo y se convierte a escala de grises, lo que reduce la dimensionalidad y permite que la segmentación se base únicamente en intensidades. Esta simplificación es fundamental en Chan-Vese, ya que el método trabaja con las intensidades para definir las regiones dentro y fuera del contorno. En relación con GVF, el cálculo del gradiente también se realiza sobre imágenes en escala de grises para obtener mapas vectoriales robustos. Además, el Balloon Model se apoya en estos valores de intensidad para expandir o contraer el contorno, dependiendo de las regiones homogéneas a las que debe ajustarse.

2.3. Inicialización de la Función de Nivel

```python
IniLSF = np.ones((img.shape[0], img.shape[1]), img.dtype)
IniLSF[30:80, 30:80] = -1
IniLSF = -IniLSF
```

Se define una función de nivel inicial (Φ) con valores positivos fuera del contorno y negativos dentro. Esta inicialización establece una curva inicial desde la cual evolucionará el contorno. La función de nivel inicial (Φ) define el contorno como una superficie implícita. Los valores positivos ($\Phi > 0$) representan el exterior del contorno, mientras que los negativos ($\Phi < 0$) representan el interior. Esta representación permite que el contorno evolucione de forma flexible durante la segmentación.

2.4. Subfunción mat_math

```python
def mat_math(intput, str):
    output = intput
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if str == "atan":
                output[i, j] = math.atan(intput[i, j])
            if str == "sqrt":
                output[i, j] = math.sqrt(intput[i, j])
    return output
```

Esta función aplica operaciones matemáticas como la tangente inversa ($arctan$) o la raíz cuadrada ($\sqrt{\ }$) a cada píxel de una matriz. Estas operaciones son necesarias para regularizar términos como la función Heaviside suavizada y su derivada. La función Heaviside regularizada es clave en Chan-Vese para evitar discontinuidades bruscas en el contorno. En GVF, las regularizaciones matemáticas, como el término de difusión $\mu\nabla^2 v$, cumplen un propósito análogo: suavizar el flujo vectorial y extender la influencia de los bordes.

2.5. Subfunción CV

```python
def CV(LSF, img, mu, nu, epison, step, show=False):
    Drc = (epison / math.pi) / (epison * epison + LSF * LSF)
```

```
Hea = 0.5 * (1 + (2 / math.pi) * mat_math(LSF / epison, "atan"))
Iy, Ix = np.gradient(LSF)
s = mat_math(Ix * Ix + Iy * Iy, "sqrt")
Nx = Ix / (s + 0.000001)
Ny = Iy / (s + 0.000001)
Mxx, Nxx = np.gradient(Nx)
Nyy, Myy = np.gradient(Ny)
cur = Nxx + Nyy
Length = nu * Drc * cur
```

Esta función calcula los términos necesarios para la evolución del contorno. La curvatura (κ) regula la suavidad del contorno, mientras que el término de longitud ($Length(C)$) penaliza contornos largos, favoreciendo formas más regulares. En GVF, la curvatura es utilizada de manera similar para suavizar el campo vectorial y extender gradientes hacia regiones homogéneas. En el Balloon Model, la fuerza de inflación ($kn(s)$) introduce un efecto comparable al empujar el contorno hacia afuera o adentro de manera uniforme.

2.6. Energía de Región

Python
```
s1 = Hea * img
s2 = (1 - Hea) * img
s3 = 1 - Hea
C1 = s1.sum() / Hea.sum()
C2 = s2.sum() / s3.sum()
CVterm = Drc * (-1 * (img - C1) * (img - C1) + 1 * (img - C2) * (img - C2))
```

El término de energía de región se calcula utilizando los valores promedio dentro (c_1) y fuera (c_2) del contorno. Este término atrae el contorno hacia regiones homogéneas, minimizando las diferencias entre los valores de intensidad y estos promedios. En GVF, una fuerza externa basada en el gradiente ($|\nabla f|^2$) atrae el flujo hacia los bordes detectados. En el Balloon Model, la energía de región puede interpretarse como una fuerza de inflación que empuja el contorno hacia regiones con diferencias de intensidad significativas.

2.7. Evolución de la Curva

```python
LSF = LSF + step * (Length + Penalty + CVterm)
```

La evolución del contorno se realiza mediante un esquema iterativo que combina los términos de curvatura, longitud y energía de región. Esto permite que el contorno se ajuste a los bordes de las regiones homogéneas minimizando la energía total. Este proceso es similar a la evolución en GVF, donde el flujo vectorial se actualiza iterativamente para alinear el campo con los bordes. En el Balloon Model, una fuerza constante dirige la expansión o contracción del contorno en cada iteración, un comportamiento que podría integrarse en este esquema sumando un término adicional de fuerza de inflación.

2.8. Proceso Iterativo

```python
for i in range(1, num):
    LSF = CV(LSF, img, mu, nu, epison, step, False)
```

El contorno evoluciona iterativamente minimizando la energía de Chan-Vese hasta que converge hacia una solución estable que representa los bordes segmentados. Este proceso iterativo es comparable al cálculo del flujo vectorial en GVF, donde se resuelve una ecuación diferencial parcial hasta alcanzar la estabilidad. En el Balloon Model, las iteraciones ajustan la posición del snake empujándolo hacia los bordes detectados.