

Sistemas Distribuidos y Paralelos
Departamento de Ingeniería en Informática
LAB 1: OpenMP

1 Objetivo

El objetivo de este laboratorio es diseñar e implementar una aplicación paralela en sistemas de memoria compartida para detectar elipses en imágenes radiointerferométricas de discos protoplanetarios.

2 Imágenes de discos protoplanetarios

La imagen de la Figura 1 (a) muestra una típica imagen de un sistema protoplanetario. Esta imagen es el resultado de un proceso de síntesis o reconstrucción a partir de los datos capturados por observatorios de radio interferometría, tales como ALMA y SKA. Las imágenes reconstruidas de discos protoplanetarios generalmente, pero no siempre, muestran discos de alta intensidad, concéntricos, y separados por gaps de muy baja intensidad. Los discos de alta intensidad reflejan la presencia de polvo circundante a la estrella central, la cual nunca se ve en estas imágenes. Solo se observa la radiación producida por la interacción entre los fotones de luz y el polvo alrededor de la estrella. Luego, los gaps oscuros representarían regiones donde hay muy poco polvo, y donde se hipotetiza existen protoplanetas, o planetas en formación.

En este laboratorio implementaremos una aplicación paralela que encuentre las elipses presentes en la imagen.

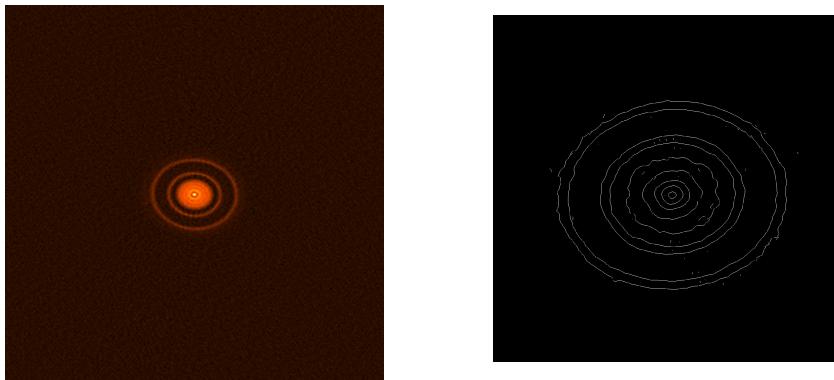


Figure 1: (a) Imagen sintetizada del disco AS209 (izquierda). (b) Imagen de bordes (derecha)

Sin embargo, las imágenes de borde contienen varios píxeles blanco que no pertenecen a ningún borde y algunos píxeles negros que debería haber sido marcados como blancos.

3 Parametrización de elipses

El siguiente algoritmo de detección de elipses es del paper "ELLIPSE DETECTION WITH HOUGH TRANSFORM IN ONE DIMENSIONAL PARAMETRIC SPACE", de Alex Yong Sang Chia, Maylor

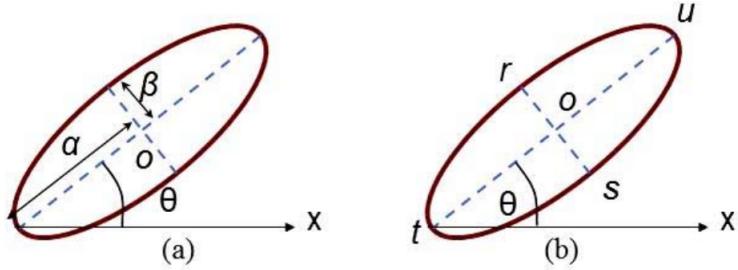


Figure 2: Parametrización de una elipse.

K. H. Leung, How-Lung Eng, y Susanto Rahardja, 2007 IEEE Int. Conf. Imag. Proc.

La Figura 2 (a) muestra una elipse con centro $o = (o_x, o_y)$, semieje mayor α y semieje menor β . El ángulo θ es el ángulo que forma el semieje mayor con el eje x . Luego, una elipse puede ser representada unívocamente por la 5-tupla $(o_x, o_y, \alpha, \beta, \theta)$.

La misma elipse puede ser descrita como se muestra en la Figura 2 (b). Sean $t = (t_x, t_y)$ y $u = (u_x, u_y)$ dos puntos sobre la elipse, como se muestra en la figura. Luego, es posible calcular 4 de los 3 parámetros anteriores, como sigue:

$$o_x = \frac{t_x + u_x}{2} \quad (1)$$

$$o_y = \frac{t_y + u_y}{2} \quad (2)$$

$$\alpha = \frac{\sqrt{(u_x - t_x)^2 + (u_y - t_y)^2}}{2} \quad (3)$$

$$\theta = \tan^{-1}\left(\frac{u_y - t_y}{u_x - t_x}\right) \quad (4)$$

Solo faltaría calcular β .

Considere la elipse de la Figura 3 con focos en los puntos w y v . Sea k un punto arbitrario sobre la elipse, y $l_{w,k}$, $l_{k,v}$ los largos de los segmentos entre los focos y el punto k . Para una elipse, se cumple que la suma de estos dos largos es siempre la misma para cualquier punto k sobre la elipse, y es igual a dos veces el semieje mayor, es decir

$$\sqrt{((k_y - w_y)^2 + (k_x - w_x)^2} + \sqrt{(k_y - v_y)^2 + (k_x - v_x)^2} = 2\alpha$$

con

$$w_x = o_x - \cos|\theta| \sqrt{\alpha^2 - \beta^2} \quad (5)$$

$$w_y = o_y - \sin|\theta| \sqrt{\alpha^2 - \beta^2} \quad (6)$$

$$v_x = o_x + \cos|\theta| \sqrt{\alpha^2 - \beta^2} \quad (7)$$

$$v_y = o_y + \sin|\theta| \sqrt{\alpha^2 - \beta^2} \quad (8)$$

Luego, dado un punto arbitrario k sobre la elipse, se puede calcular β con la siguiente ecuación:

$$\beta = \sqrt{\frac{\alpha^2 \delta^2 - \alpha^2 \gamma^2}{\alpha^2 - \gamma^2}}$$

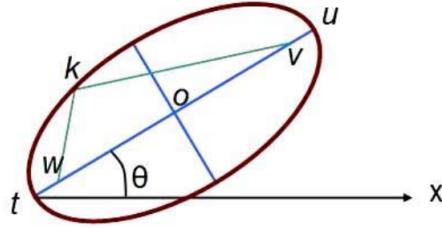


Figure 3: Elipse.

donde

$$\delta = \sqrt{(k_y - o_y)^2 + (k_x - o_x)^2} \quad (9)$$

$$\gamma = \sin |\theta|(k_y - o_y) + \cos |\theta|(k_x - o_x) \quad (10)$$

4 Algoritmo de Hough

El algoritmo de Hough permite detectar formas geométricas en imágenes de bordes. Una imagen de borde es una imagen binaria como lo muestra la Figura 1 (b). Los píxeles que pertenecen a los bordes de los discos son representados con el color blanco (255), y los píxeles que no pertenecen a algún borde con el color negro (0).

La idea básica del algoritmo de Hough es mapear una imagen binaria a un espacio de parámetros que describen las formas geométricas de la imagen. En general, si la forma geométrica se puede representar con dos parámetros, el espacio de Houghería bidimensional.

En el caso particular de la parametrización descrita en la sección anterior, el espacio de parámetros sería unidimensional (β). Cada punto borde de la imagen de entrada, "vota" a favor de todas las posibles elipses a las que dicho punto podría pertenecer.

El algoritmo considera cada par de puntos bordes de la imagen, como posible puntos t y u , de una posible elipse, con lo que se calcula la tupla $(o_x, o_y, \alpha, \theta)$ según las ecuaciones (1), (2), (3) y (4). Luego, cada otro punto borde k votará a favor del semieje menor con largo β , según las otras ecuaciones. Note que cada pixel borde calculará un solo β correspondiente a la elipse hipotética que pasa por t , u , y k .

Considere entonces que todos los puntos que están sobre la elipse, calcularán valores iguales o similares de β . Luego, este β tendrá más votos que otros betas y podrá ser identificado posteriormente como una elipse $(o_x, o_y, \alpha, \beta, \theta)$.

El siguiente es el pseudo-código de esta parte del algoritmo:

```

newEP = []

for each edge pixels t {
    for each edge pixel u {

        Initialize vote[] = 0

        compute o_x, o_y, alpha, theta
        for each other edge pixel k != t, u {
    }
}

```

```

        compute beta
        vote[beta]++
    }

    {beta_i, i=1, ... ,n } = (vote[] > 0 )
    for each found ellipse i {
        newEP = append(newEP, (o_x, o_y, alpha, theta, beta_i))
    }
}

```

Al final del loop interno, `vote[beta]` representará el número de veces que una elipse que pasa por los puntos t , u y cuya mitad de su semieje menor es igual a β . Dado que la imagen posee ruido y la detección de bordes no es perfecta, es posible que se detecten varias elipses "falsas" al finalizar el loop interno. El autor del paper introduce varias optimizaciones para elegir solo elipses que cumplan con ciertos requisitos. Por ejemplo,

1. Si el valor de α es muy pequeño, se descarta dicha elipse.
2. Si la distancia entre el centro de la elipse y el punto k es mayor que α , se descarta.
3. Si la votación final no es mayor que un cierto porcentaje de la circunferencia de la elipse hipotética, también se descarta.

La lista `newEP` contiene todas las elipses encontradas en la imagen.

5 Detalles de implementación

El primer paso del programa consistirá en leer la imagen de bordes que estará en formato FITS. El tamaño de la imagen se extraerá del header de este archivo. A partir de esta imagen, se debe crear una lista de puntos bordes. Es decir, se debe recorrer la imagen, y cada pixel (i, j) cuyo valor sea 255, debe ser agregado a esta lista de puntos. De esta forma, el algoritmo de Hough, trabajará con esta lista y no con la imagen de entrada.

Este procesamiento debe ser hecho en paralelo, y el resultado final es una lista con todos los píxeles bordes.

La imagen de entrada, en formato FITS, se debe leer usando las funciones de la librería `cfitsio` para C.

Al finalizar el programa, éste debe imprimir por `stdout` los parámetros de cada elipse encontrada, una por línea. Estos parámetros podrán ser usados en un script de Python para visualizar las elipses.

Dado que β toma valores reales, es necesario discretizar este espacio, de tal forma que la instrucción `vote[beta]++` tenga sentido. Recordando que β es la mitad del semieje menor, podemos decir que su valor máximo no excederá el tamaño de la imagen. Suponga una elipse circular centrada en la imagen de tamaño N . Luego, el valor de β será menor o igual a $N/2$. Entonces, si deseamos discretizar β en B valores (discretos), tenemos que

$$\Delta_\beta = \frac{N}{2B}$$

Entonces, dado un β , tenemos que el índice para incrementar el arreglo `vote[]` es

$$i = \left\lfloor \frac{\beta}{\Delta_\beta} \right\rfloor$$

y

$$\beta_i = i \times \Delta_\beta \quad i = 0, \dots, B - 1$$

Claramente, esta discretización tiene implicancias directas en el rendimiento computacional y en la calidad de la solución.

6 Ejecución

El programa se ejecutará como sigue

```
$ ./lab1 -i imagein.fits -a alphamin -r relativevote -b numerobetas -T numerohebras
```

- `-i` especifica el nombre de la imagen de entrada
- `-a` el valor mínimo para α
- `-r` el porcentaje aplicado a la circunferencia
- `-b` el número de betas
- `-T` el número de hebras

Debe usar `getopt()` para el manejo de argumentos de línea de comandos.

7 Formato FITS

Las imágenes estarán almacenadas en formato FITS, que es el formato más común para imágenes astronómica.

- Para visualizar una imagen fits, instale `ds9`.
- Para leer, escribir y manipular imágenes fits instale la librería `cfitsio`. Instale la versión de developer.
- Existen paquetes para ubuntu en ambos casos.
- Existen varios ejemplos en la web de cómo usar `cfitsio`

Enviaré por Classroom un ejemplo de lectura y escritura con `cfitsio`.

8 Entregables

Tarree, comprima y envíe a `fernando.rannou@usach.cl` al menos los siguientes archivos:

1. `Makefile`: archivo para make que compila los programas
2. Uno o más archivos con apropiada documentación en código.