Proyecto 1 - Implementación de un algoritmo para aproximación de rectas a imágenes de patas de rata

Erick Salvador Alvarez Valencia, Centro de Investigación en Matemáticas, A.C.

Index Terms—Imagenes binarias, Aproximación de rectas.

1. Introducción

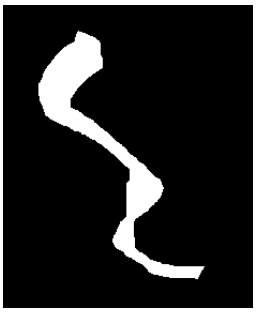
En el presente reporte se analizará la implementación de un algoritmo el cual recibe una imagen binaria que consiste en un dibujo de una pata de rata, y el mismo generará un conjunto un modelo de rectas que se ajusta a dicha imagen. De la misma forma el algoritmo se encarga de calcular las longitudes y los ángulos correspondientes a dicho conjunto de rectas.
En las secciones consiguientes se irá analizando la elaboración de dicho algoritmo, así como las pruebas realizadas y las conclusiones proporcionadas, las cuales incluyen mejoras para trabajos futuros.

mds
Octubre, 2017

2. Desarrollo del algoritmo de aproximación

2.1. Bases y conocimientos necesarios

Primero comenzaremos analizando el problema, el cual consiste en: dado un conjunto de imágenes binarias de patas de rata, se quiere realizar una caracterización de movimiento, para ello, se debe generar un ajuste de rectas (o en su defecto rectángulos) para poder encontrar los ángulos de inclinación de las mismas con respecto al eje de las ordenadas. A continuación se mostrará una imagen binaria con la que se trabajó.



(a) Figura 1. Imagen binaria de una pata de rata.

En la Figura 1. podemos observar que la imagen contiene el dibujo muy aproximado de una pata de rata, y a su vez la misma imagen se divide en dos clases de pixeles los cuales denominaremos como cero y no cero, esto último ya que pueden existir pixeles que no sean necesariamente blancos pero tampoco sean ceros. Antes de continuar vale la pena destacar que las imágenes se encuentran en formato pgm, las cuales son fáciles de procesar ya que con tienen una matriz con los datos de los pixeles en escala de gris.

Para generar el modelo de rectas que se ajusten a dicha imagen, se pueden aplicar varias ideas, entre las cuales destacan las vías genéticas y heurísticas, con algoritmos como UMDA, también podemos irnos por el camino del ajuste de curvas con algoritmos como mínimos cuadrados. Pero en este caso se optó por una implementación de un algoritmo sencillo que se encarga de encontrar rectas en función de un conjunto de puntos dados. Para lo anterior comenzamos haciendo un poco de tratamiento a la imagen, ya que las muestras son bastante grandes y se determinó que éstas contienen un conjunto de puntos clase cero, el cual no se necesitan, se hizo un algoritmo que recorte las imágenes.

El algoritmo de recorte realiza un barrido a toda la imagen y encuentra los puntos $x_min, y_min, x_max, y_max$, los cuales son puntos frontera, ya que en base a ellos se realiza un rectángulo que engloba la parte de la pata. Esto anterior es posible en todas las imágenes ya que en ninguna encontraremos elementos no conexos.

2.2. Esqueletización de la imagen

Una vez recortada la imagen, el siguiente paso es realizar una esqueletización a la misma. Dicha esqueletización se realizará debido a que ésta generará un modelo lo más delgado posible de la pata, y a su vez, dicho modelo se encontrará lo más centrado posible, la idea es encontrar el conjunto de rectas en base a dicha esqueletización, ya que podremos decir que se ajustarán correctamente al modelo original. Como se comentó anteriormente, se podrán lograr mejores resultados con otro tipo de algoritmos, pero el que se describirá en este reporte arroja resultados bastante decentes.

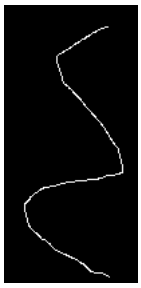
Para la esqueletización se usará el algoritmo de Zhang-Suen, el cual genera una esqueletización topológica la cual nos asegura que el resultado mantendrá la forma de la figura original así como otras propiedades, tales como: conectividad, dirección y longitud.
La idea principal de este algoritmo es realizar dos barridos a la imagen y por cada barrido ir obteniendo los puntos de interés que cumplan ciertas reglas, una vez obtenidos los puntos, éstos cambiarán a clase cero. Este proceso

se realizará mientras existan cambios en la imagen. A continuación se mostrará el pseudocódigo del algoritmo de Zhang-Suen.

Algorithm 1 Algoritmo de esqueletización de Zhang-Suen.

```
1: procedure SKELETONIZE(ori)
2:     dest ← Copia de la imagen original
3:     height ← ori.height
4:     while Haya cambios do
5:         list ← []
6:         for i ← 1 to height - 1 do
7:             for j ← 1 to width - 1 do
8:                 if ori.mat[i][j] != 0 then
9:                     if ori.mat[i][j] Cumple con todas los filtros then
10:                        list.add(ori.mat[i][j])
11:        for i ← list.size() do
12:            ori.mat[list[i].i][list[i].j] Se cambia a cero
13:        lis2 ← []
14:        for i ← 1 to height - 1 do
15:            for j ← 1 to width - 1 do
16:                if ori.mat[i][j] != 0 then
17:                    if ori.mat[i][j] Cumple con todas los demás filtros then
18:                        list2.add(ori.mat[i][j])
19:        for i ← list.size() do
20:            ori.mat[list[i].i][list[i].j] Se cambia a cero
21:    return dest
```

De entre los filtros que debe cumplir el pixel en la posición i, j se encuentran validaciones de conectividad y de que el pixel no sea parte de un borde. A continuación se muestra un resultado de la esqueletización aplicada a una imagen.



(b) Figura 2. Resultado de la esqueletización de una imagen.

Podemos ver que la imagen mostrada en la Figura 2. conserva la forma topológica de la original, y de la misma forma se puede verificar que dicha imagen encaja justamente en medio de la original.

2.3. Recorrido de la esqueletización y búsqueda de puntos extremos

Una vez que se tiene la esqueletización realizada, el siguiente paso es encontrar los puntos extremos y la ruta que genera la misma. Esta última nos servirá en el siguiente paso del algoritmo.
La búsqueda de puntos extremos se realiza recorriendo la imagen de forma conveniente hasta que se encuentre un pixel que no se de clase cero. Por forma conveniente, nos referimos a la dirección que irán tomando los X's y los Y's, es evidente que si las Y's empiezan en cero y van aumentando, eventualmente encontraremos el punto extremo de la parte de arriba. Hay que notar que esta no es la forma más óptima de encontrar los puntos extremos pero genera buenos resultados.

Para generar el recorrido de la esqueletización y encontrar la ruta de la misma se optó por un algoritmo de búsqueda por anchura (DFS), el cual nos garantiza visitar todos los puntos de la esqueletización dado uno inicial. para muestra se indica el pseudocódigo de la búsqueda.

Algorithm 2 Búsqueda en esqueletización por DFS.

```
1: procedure DFS(ori, p_ini, path)
2:     mark ← matriz binaria.
3:     stack ← nueva Pila.
4:     while stack no esté vacía do
5:         pact ← stack.pop()
6:         neight ← Obtener vecinos de clase no cero de pact.
7:         Añadir neight a path.
8:         for i ← 1 to neight.size() do
9:             aux ← neight[i]
10:            if aux no ha sido marcado then
11:                Añadir aux a stack.
12:                Marcar aux como visitado.
```

Al concluir el DFS podremos encontrar los puntos de la esqueletización de forma ordenada en el vector $path$. Cuando corramos el algoritmo hay que pasar un punto de los extremos como el inicial.

2.4. Búsqueda de las rectas con el algoritmo Raper-Douglas-Peucker

Para la parte más crucial del programa, encontraremos las rectas mediante el algoritmo recursivo de Ramer-Douglas-Peucker o RDP el cual se encarga dado un conjunto discreto de puntos que representan un camino (curvas o rectas) encuentra un conjunto de rectas que se ajusten con un valor épsilon dado. Cabe destacar que las rectas encontradas se relacionan con los máximos y mínimos locales que hay en el conjunto de puntos. Otra cosa importante a destacar es que el conjunto discreto de puntos debe estar bien definido y seguir una secuencia clara, ya que el RDP generará las rectas en base a la secuencia que vaya siguiendo del conjunto de puntos, por esa razón se optó por hacer el algoritmo de DFS para encontrar dicha secuencia. A continuación se verá la implementación en pseudocódigo del RDP.

Algorithm 3 Algoritmo de Ramer-Douglas-Peucker.

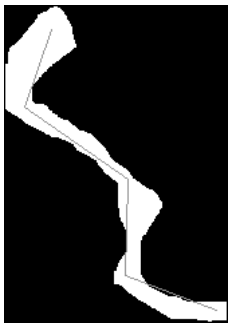
```
1: procedure RDP(pointsList, out, eps)
2:     dmax ← 0
3:     index ← 0
4:     for i ← 2 to pointList.size() - 1 do
5:         d ← Distancia perpendicular entre el punto i y la
   línea generada por el primer y último punto.
6:         if d > dmax then
7:             index ← i
8:             dmax ← d
9:     if dmax > eps then
10:        r1, r2 ← []
11:        fl ← Vector con los puntos que van desde 1 a
   index
12:        ll ← Vector con los puntos que van desde index
   a pointList.size()
13:        RDP(fl, eps, r1)
14:        RDP(ll, eps, r2)
15:        Copiar los resultados de r1 y r2 a out
16:    else
17:        Limpiar out
18:        Añadir pointList[0] a out
19:        Añadir pointList[end] a out
```

En el algoritmo anterior la distancia perpendicular se puede calcular de la siguiente forma: $distance(P_1, P_2, \{x_0, y_0\}) = |(y_2 - y_1)x_0 - (x_2 - x_1)y_0 + x_2y_1 - y_2x_1|/\sqrt{((y_2 - y_1)^2 + (x_2 - x_1)^2)}$

Para este algoritmo hay que tener mucho cuidado en dar un épsilon correcto para encontrar las rectas correctas, lo que se hizo fue correr el algoritmo con diferentes valores del épsilon hasta que el conjunto de rectas generadas era la deseada. Hay que mencionar que el RDP dio más puntos de los buscados, así que se tuvo que hacer una selección de los mejores.
La selección de puntos se hizo iterando el conjunto que retornó el algoritmo y se fue buscando primeramente el punto que estuviera más a la izquierda en la componente x, posteriormente el que estuviera más a la derecha y finalmente, el de más a la izquierda, todo esto avanzando en la componente y.

3. Ejemplos de ejecución y resultados

En esta sección se mostrarán algunas de las imágenes que generó el programa, así como capturas de pantalla sobre la consola donde se muestran algunos de los ángulos obtenidos. Como el programa genera las imágenes con respecto a la aproximación realizada y las tablas de los ángulos de dichas aproximaciones, lo más fácil será ver algunas imágenes de las aproximaciones y posteriormente veremos la gráfica de los ángulos y longitudes con respecto al tiempo.



(c) Figura 3. Modelo de aproximación de la pata de rata.

(d) Figura 4. Modelo de aproximación de la pata de rata.

(e) Figura 5. Modelo de aproximación de la pata de rata.

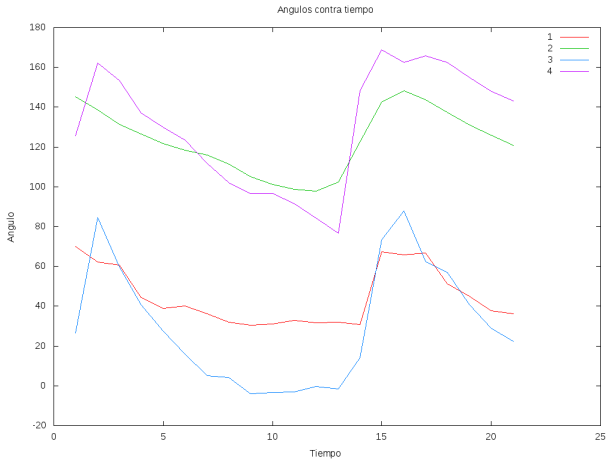
(f) Figura 6. Modelo de aproximación de la pata de rata.

Podemos ver en las Figuras 3 a 6 las aproximaciones generadas en diferentes tiempos para la pata de rata, se puede apreciar que los resultados no son bastante exactos

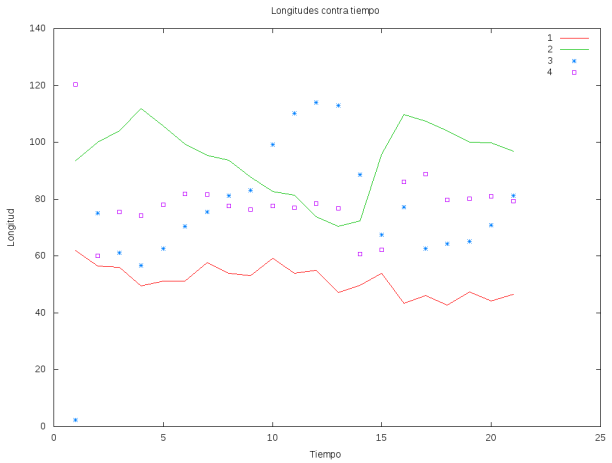
pero si entran en un rango de buena aproximación, donde podemos de ahí generar nuestro análisis con los ángulos y las longitudes.

4. Caracterización del movimiento

En esta sección veremos las gráficas generadas de ángulos contra tiempo y longitudes contra tiempo. Cabe mencionar que dichas gráficas se hicieron con el software GNUPlot.



(g) Figura 7. Ángulos contra tiempo.



(h) Figura 8. Longitudes contra tiempo.

Mediante las gráficas anteriores podemos ver cómo se comporta y caracteriza el movimiento ya sea de una rata en estado sano o con alguna enfermedad, dicha información puede servir para compararla con otros conjuntos de ángulos y longitudes.

5. Conclusiones

Se pudo ver en el presente reporte una propuesta de solución para el modelamiento de una imagen correspondiente a una pata de rata y posteriormente su caracterización con respecto al movimiento. De la misma forma nos pudimos dar cuenta que el método propuesto es completamente determinístico, por lo cual dará los mismos resultados, aunque se encontró un error el cual en algunas ocasiones se equivocaba en elegir la mejor recta de las generadas por el RDP ya que esta se encontraba en la misma componente y de otra línea, la solución fue añadir un poco de ruido para quitarlas de la componente.
Una mejora a proponer es el limpiado de la esqueletización, hay que ver que el algoritmo de Zhang-Suen nos arroja buenos resultados en general, y en algunos de ellos podemos encontrar bifurcaciones en donde las cuales hay un buen punto de articulación, si se mejora el algoritmo DFS para tomar en cuenta esas bifurcaciones, los resultados serán más exactos.

Referencias

[1] T. Y. Zhang, C.Y. Suen, "A Fast Parallel Algorithm for Thinning Digital Patterns", Communications of the ACM, pp.236-239, Vol. 27, Num. 3, March 1984.

[2] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required for represent a digitzed line or its caricature. Canadian Cartographer, 10(2):112-122, 1973.