Instituto Tecnológico de Buenos Aires

Automación Industrial

“Trabajo práctico Vision”

Alumno:

Maria Luz Stewart Harris 57676
Gonzalo Silva 56089
Sebastian Milhas 55198
Matias Nicolas Pierdominici 57498

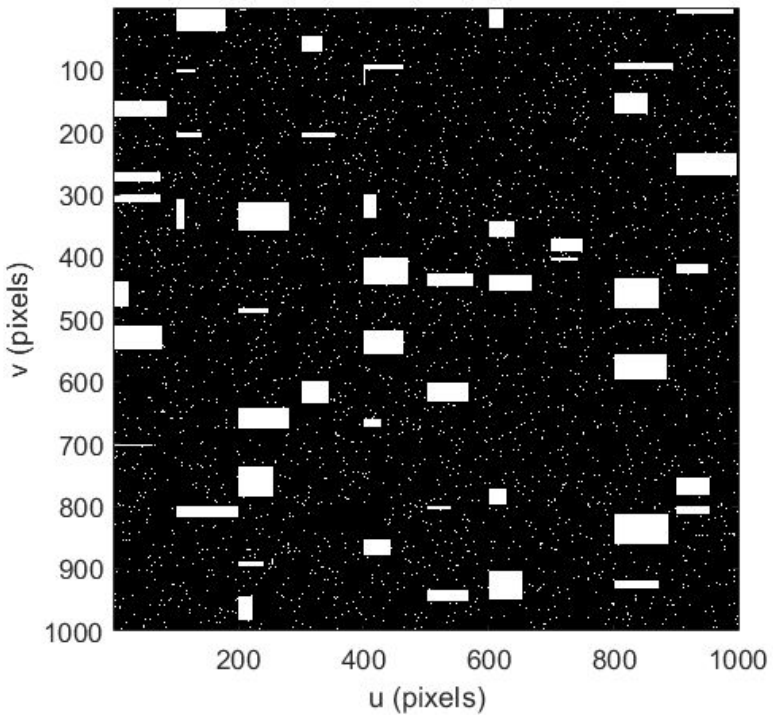
Profesores:

Rodolfo Enrique Arias
Federico Avogadro
Mariano Tomas Spinelli

Filtrado de ruido

Identificación del tipo de ruido

Se corrió el código provisto por la cátedra, obteniéndose la siguiente imagen



En la figura se observan cuadrados blancos(información valiosa) y puntos(ruido). Este tipo de ruido es conocido como ‘salt and pepper’, presenta gran amplitud respecto a los pixeles que lo rodean.

Remoción de ruido

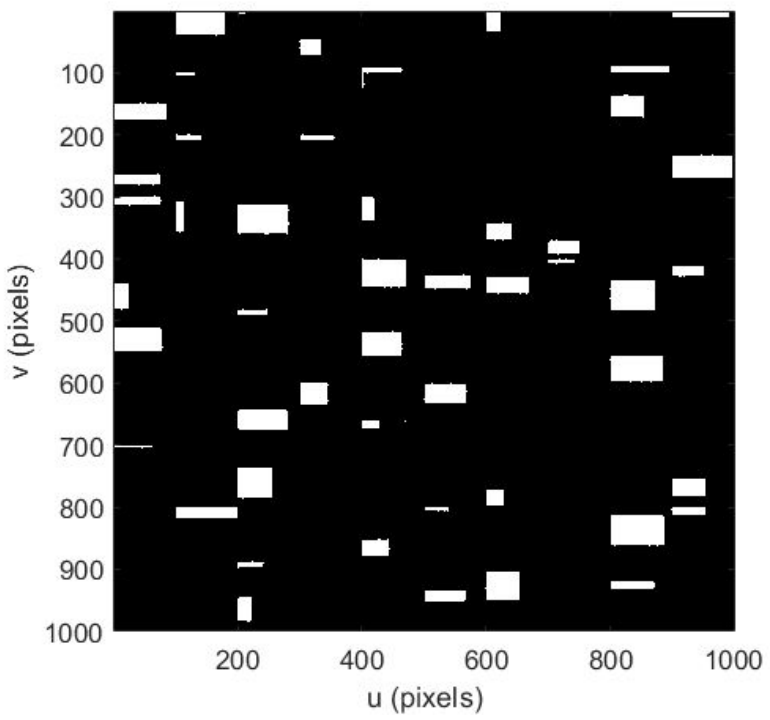
Para el filtrado de ruido se analizaron diferentes métodos. Teniendo en cuenta el funcionamiento del algoritmo para cada una de las alternativas, se analizó cuál de todas podría llegar a tener el mejor resultado. Luego se optó por la que dió el mejor resultado, con prueba y error.

Filtrado Lineal

En este caso se probó filtrar el ruido con un procesos de erosión y dilatación de la imágen. En particular se implementaron diferentes combinaciones de las funciones iclose y e iopen. Finalmente una alternativa que dió buen resultado es la que se muestra en el siguiente fragmento de código.

```
1. S2=kcircle(1);
2. filtrado_02=iopen(iclose(imagen,S2),S2);
```

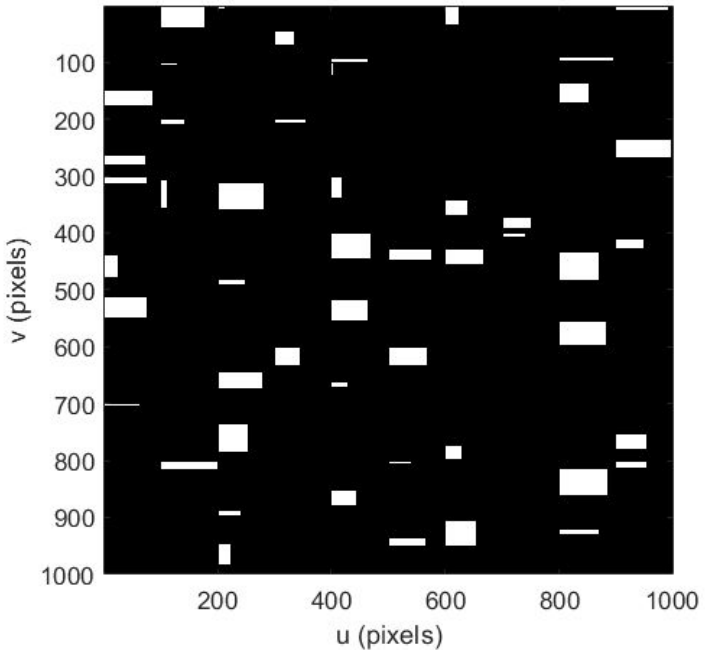
```
3. figure();idisp(filtrado_02);
```



Como se observa en la figura, el método removió el ruido del fondo de la imagen. Sin embargo, en los bordes de los rectángulos donde había ruido cerca se observan pequeñas espurias. Además los rectángulos finos se ven fuertemente afectados.

Filtrado no Lineal

Para este caso se utilizó la función irank, provista en el toolbox. Esta alternativa fue la que dió el mejor resultado final. Modificando el tamaño del kernel que utiliza la imágen y el rango que utiliza para definir el pixel de salida, se probaron diferentes variantes. Finalmente se decidió que la combinación de un kernel de 3x3 con un rango 8,es decir toma el octavo valor del arreglo,ordenado por intensidad, de 9 pixeles. La combinación indicada resultó ser la que arrojó los mejores resultados.



En la imágen anterior se puede observar el resultado final. Si comparamos esta imágen con la original, podemos observar que el error es mínimo. Cabe destacar que para los casos en que un rectángulo pequeño no es ruido, este puede terminar siendo filtrado. Esto puede ocurrir por la forma en que funciona el algoritmo.

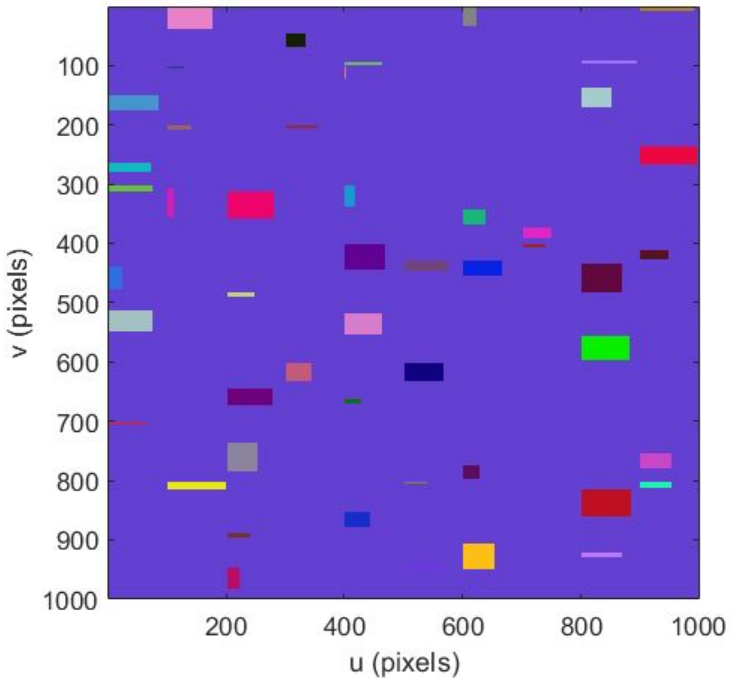
Finalmente este fue el método elegido para realizar el filtrado de la imagen.

Trazado de líneas

Identificación de rectángulos

La información necesaria para realizar la tarea propuesta es el centro de cada rectángulo, su ancho y alto.

Para hallar los rectángulos se utilizó la función iblobs. Esta recibe como parámetro la imagen y devuelve un arreglos de objetos de tipo *RegionFeature*, con información de los objetos encontrados y la imagen segmentada. Con la información de estos objetos se puede obtener los puntos que definen las aristas de los rectángulos, junto con sus centroides. Esto último se puede realizar de forma manual o directa, llamando ciertos métodos del tipo *RegionFeature.*




```
val =

(1) area=114, cent=(3.5,10.0), theta=1.57, b/a=0.312, color=1, label=1, touch=1, parent=0
(2) area=973195, cent=(499.8,501.1), theta=0.75, b/a=0.996, color=0, label=2, touch=1, parent=0
(3) area=552, cent=(124.5,27.5), theta=0.00, b/a=0.260, color=1, label=3, touch=0, parent=2
(4) area=1081, cent=(325.0,43.0), theta=0.00, b/a=0.489, color=1, label=4, touch=0, parent=2
(5) area=34, cent=(810.0,32.5), theta=0.00, b/a=0.102, color=1, label=5, touch=0, parent=2
(6) area=1404, cent=(727.5,124.0), theta=0.00, b/a=0.519, color=1, label=6, touch=0, parent=2
(7) area=350, cent=(18.0,137.5), theta=0.00, b/a=0.284, color=1, label=7, touch=1, parent=0
(8) area=702, cent=(410.5,175.0), theta=1.57, b/a=0.461, color=1, label=8, touch=0, parent=2
```

Método identificación de líneas y aristas

Para hallar el ancho y alto de cada rectángulo, probamos realizar un código que a través de detección de esquinas, obtuvieramos el ancho alto de cada imagen.
Los pasos que realizamos fueron los siguientes:

1) Identificar los bordes de cada rectángulo
2) A partir de la imagen con bordes identificados obtuvimos las esquinas
3) Los vértices obtenidos no estaban agrupados por rectángulos, por ende los ordenamos
4) Como en ciertos casos(rectángulos chicos), detectaba más de 4 vértices por rectángulos, filtramos esos vértices.
5) Agrupamos los vértices filtrados con el centroide de iblobs y completamos la siguiente estructura

```
classdef Rectangulo

%Asi se usa
% A = Rectangulo(2); crea un arreglo con 2 elementos que son de tipo
% Rectangulo
% A(1).ycentro = 1;
% A(2).ycentro = 2;

    properties
        ucentro;
        vcentro;
        deltaU;
        deltaV;
    end
```

Sin embargo este método posee el problema que para la identificación de ancho y alto depende de la identificación de esquinas, y este método no arrojó resultados tan precisos. Por ende se optó por el método expuesto en la siguiente sección.

Metodo solo iblobs

Como se mencionó anteriormente, la función iblobs devuelve un arreglo de estructuras de tipo RegionFeature. Cada uno de estos objetos se le llama Blob y cuentan con métodos que devuelven la coordenada para los puntos máximos y mínimos de la región que ocupan. Volviendo a nuestro caso de estudio, teniendo en cuenta que todas las figuras que nos interesan en la imagen son rectángulos, podemos asumir que las coordenadas en los extremos antes mencionadas son los vértices superior izquierdo e inferior derecho de los rectángulos. Luego, esta información nos simplifica el cálculo de las aristas que definen cada Blob. Además, cada Blob posee como dato el centroide del mismo. Finalmente, con los datos mencionados, tenemos toda la información necesaria para continuar con la solución del problema.

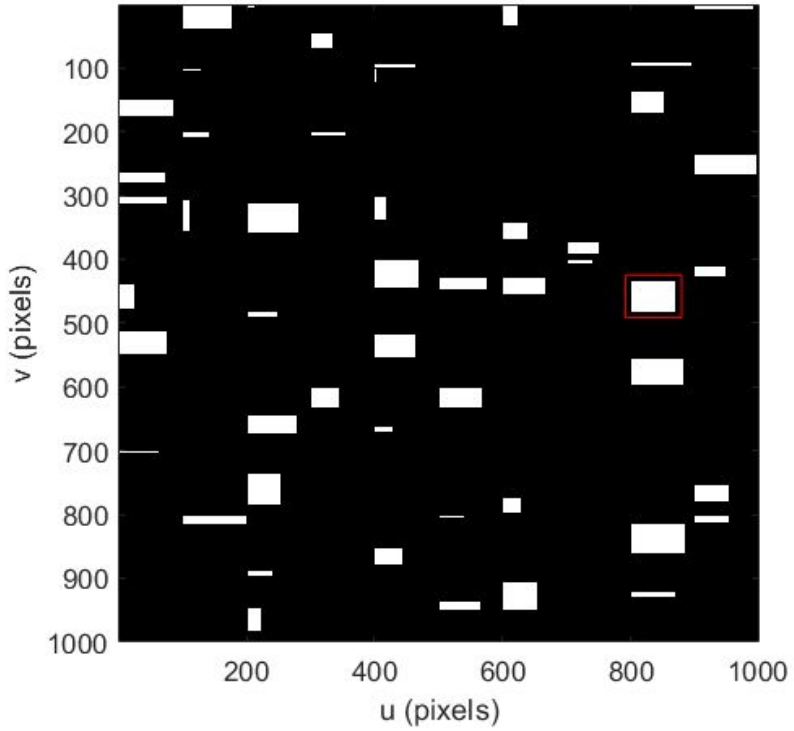
Obtención y dibujado del rectángulo objetivo

En primer lugar, del arreglo de Blobs obtenido, se retira el Blob que define el fondo de la imágen. Luego, realizando la diferencia entre los xmax e xmin(x puede ser u o v), buscamos el que posea la mallor altura en V y al menos 10 U.

```
1. function [Pmax,hmax,bbox] = g1RequiredBlob(P, margin)
2. %g1RequiredBlob función que me devuleve el Blob de mayor
   altura y al menos
3. %10 de ancho
4. %   Recive una Region Feature con todos los Blobs
5. %   devuelve el Blob de mayor altura y al menos 10 de ancho
6.     hmax = 0;
7.     for i=1:length(P)
8.         width = P(i).umax - P(i).umin;
9.         if width < 500 %para sacar el blob fondo
10.                if width >= 10
11.                    height = P(i).vmax - P(i).vmin;
```

```
                    if height > hmax
                        Pmax = P(i);
                        hmax = height;
                    end
                end
            end
        end
        bbox = [Pmax.umin - margin, Pmax.umax + margin;
                Pmax.vmin - margin, Pmax.vmax + margin];
    end
```

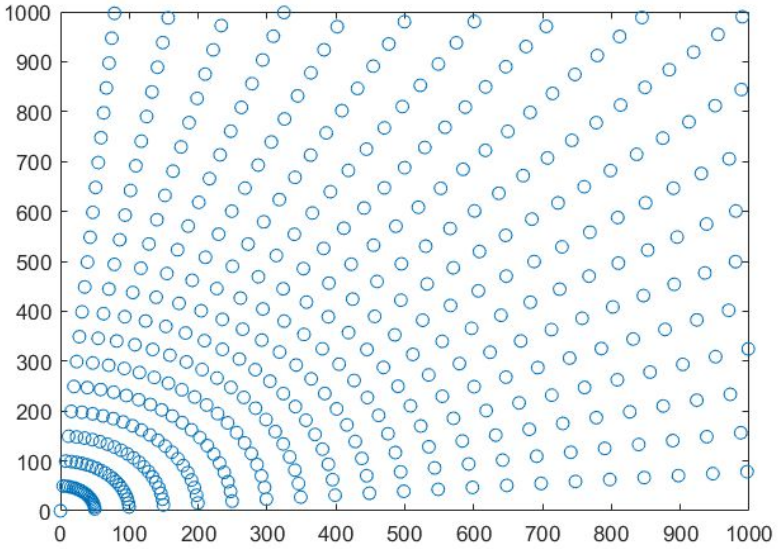
Una vez hallado el objeto que cumple las condiciones requeridas se procede a realizar el recuadro rojo sobre ella. Para ello se utiliza la función plot_box del toolbox. Como parámetro recibe la las coordenadas del cuadrado a dibujar, esto se obtuvo con las dimensiones del cuadrado, agregando un margen.



Calculado y trazado del camino al cuadrado objetivo

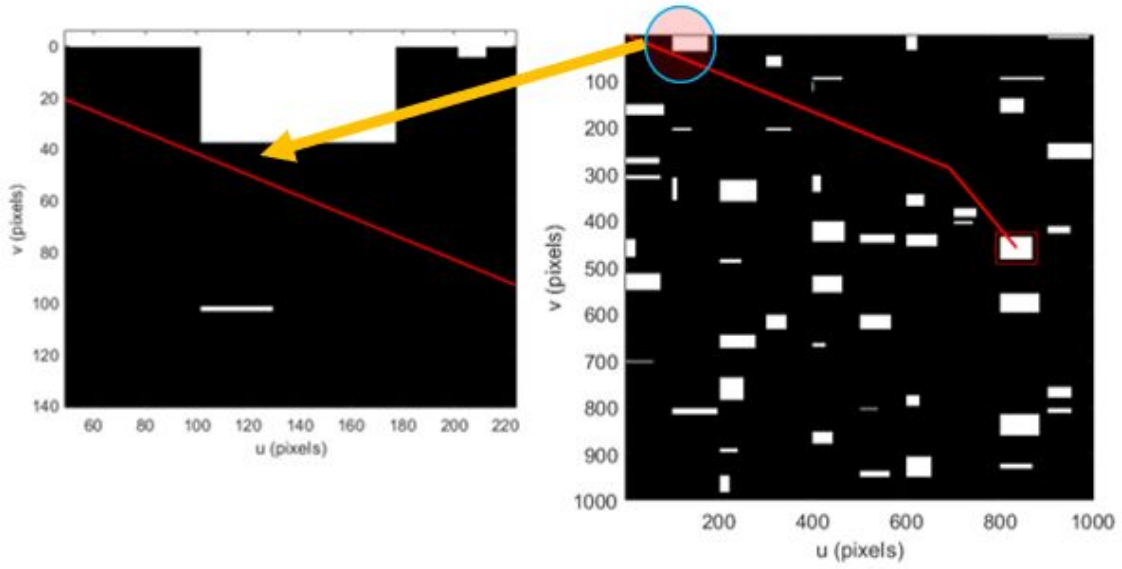
Descripción del algoritmo

Comienza generando rectas que salen del orígen [1,1] con distintas pendientes, el ángulo mínimo entre rectas es un parámetro. Posteriormente se toman puntos pertenecientes a las rectas, cada uno espaciados entre si un determinado delta, también un parámetro.



A partir de este momento los puntos mencionados(puntos azules en la imagen anterior), se llamarán puntos de inflexión.
Una vez obtenidos los puntos de inflexión, para cada uno se generan dos segmentos. El primero corresponde al comprendido entre el punto [1,1] y el punto de inflexión. El segundo comprendido entre el punto de inflexión y el centroide del rectángulo objetivo.
Posteriormente se chequea si alguno de los dos segmentos se intersectan con alguna arista de los rectángulos.
Finalmente, con los dos segmentos obtenidos que no tocan ninguna arista, se trazan las dos rectas sobre la imagen.

El resultado obtenido es el de la siguiente imagen,



Explicación de funciones

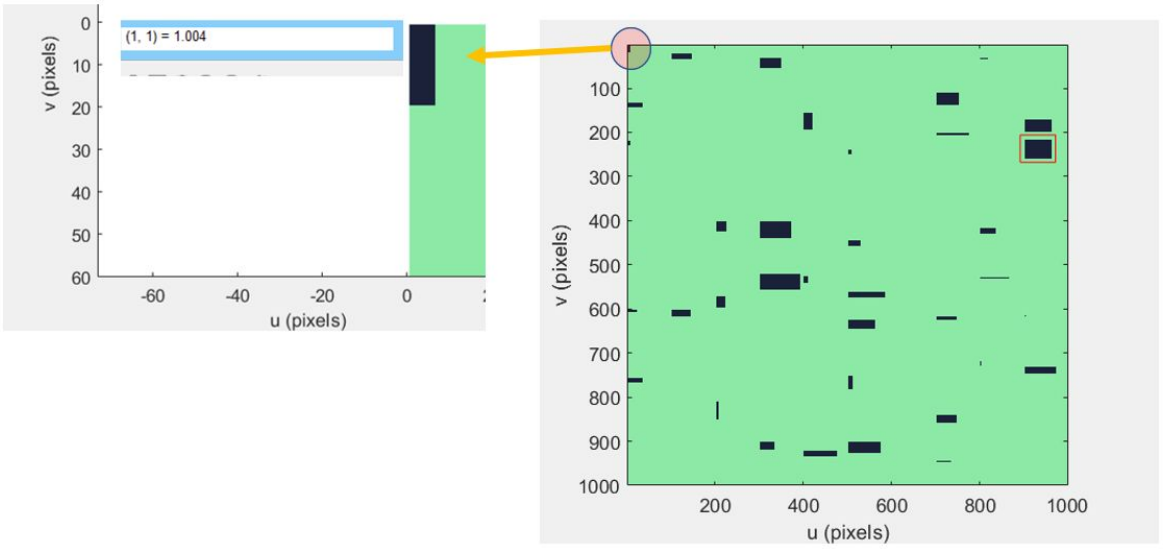
Las cuatro funciones que se utilizan para ejecutar el algoritmo son las siguientes.

```
1. function [points] = g1getPuntitos(side, angular_sep,
   radial_sep)
2.
3. function [X] = g1getInflectionPoint(points, P, XCent)
4.
5. function [goesThroughBlob,intersectNum] =
   g1goesThroughBlobs(segment, P)
6.
7. function out = lineSegmentIntersect(XY1,XY2)
```

La función `g1getPuntitos(side, angular_sep, radial_sep)` se encarga de generar el arreglo de puntos [x y] que serán evaluados como puntos de inflexión.
La función `g1getInflectionPoint(points, P, XCent)` se encarga de encontrar el punto X = [x y] que finalmente cumple con la condición de punto de inflexión.
Dentro de la función antes mencionada se utiliza la función `g1goesThroughBlobs(segment, P)` que indica si un determinado segmento de recta, definido por un par de puntos [x1 y1; x2 y2] atraviesa algún Blob en el arreglo de Blobs P.
Finalmente en la función antes mencionada, se utiliza la función `lineSegmentIntersect(XY1,XY2)` que indica la forma en la que se intersectan dos arreglos de segmentos. Para nuestro caso de uso, un arreglo de segmentos va a estar constituido por las aristas de todos nuestros Blobs y el otro simplemente va a ser el segmento desde el origen al punto de inflexión o desde el punto de inflexión al centroide que encontramos anteriormente. Esta función, no es de nuestra autoría ya que no consideramos pertinente realizar el algoritmo que detecte la intersección entre dos rectas. Por eso es que utilizamos una implementación open source, el siguiente link lleva a la documentación de la misma lineSegmentIntersect.
Cabe destacar que la forma en la que se implementó el algoritmo no es óptima, pero dado que la solución se encuentra en un tiempo despreciable, no nos pareció oportuno tratar de mejorar la eficiencia del mismo.

Existencia de solución

Existen casos en los que el sistema con encuentra solución, cómo lo expuesto en la siguiente imagen.



Esto se debe a que hay un rectángulo sobre el pixel 1 1, por ende no existe línea que pase por ese punto y no toque ningún rectángulo.