Instituto Tecnológico de Buenos Aires

22.90 Automación Industrial

Segundo Parcial

Alumno:
Lambertucci, Guido Enrique 58009

Profesores
Arias, Rodolfo Enrique
Spinelli, Mariano Tomás
Avogadro, Federico Sofio

Presentado: 18/11/21

Índice

1. Filtrado de Ruido

Para el filtrado de ruido se opto por utilizar la función del paquete de vision de Peter Corke “irank()”. Esto se debe a que el tipo de ruido que contamina la imagen es de tipo salt and pepper, caracterizado pro su alta amplitud. Justamente irank es una tecnica de filtrado no lineal que permite quitar este ruido sin perder definición ni contaminar el promedio de la imagen con los valores de los ruidos. También se consideró utilizar tecnicas como iopen o iclose pero al ver la efectividad de irank se opto por ella.

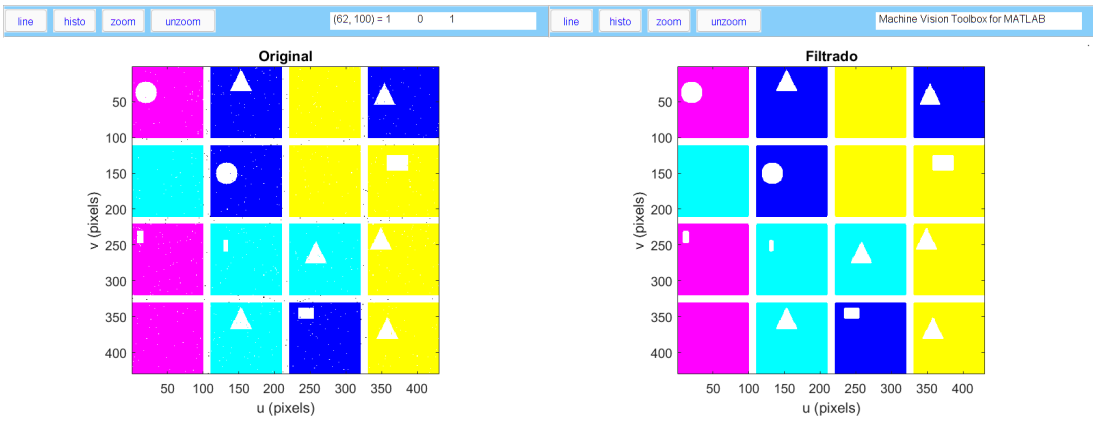

Figura 1: Comparativa original y ruido.

Se puede observar el buen resultado de utilizar la función irank

2. Idea de resolución

Se separo el problema en 4 partes, primero filtro utilizando irnak, despues se buscó aislar en el cuadro principal en los 16 subcuadros. Luego individualmente reconocer que forma hay en el subcuadro y finalmente cambiar los valores acorde a la siguiente regla.

- Si el cuadrante tiene un triángulo, elimine la componente roja de ese cuadrante.
- Si el cuadrante tiene un rectángulo, elimine la componente azul de ese cuadrante.
- Si tiene un círculo, elimine la componente verde de ese cuadrante.

3. Explicación de implementación

3.1. Main

La función irank usa dos parametros, el orden y el structuring element, para el caso en particular se optó por orden 4, con una ventana de 3x3. Para realizar el filtrado se tuvo que separar la imagen en RGB dado que irank no funciona con marcos RGB directamente. Luego una vez filtrada la imagen se la imprime, y se procede al “problema 2” el cual es la separación en marcos individuales. Para esto se tomo del marco grande los submarcos teniendo en cuenta la la separación que tienen las imágenes entre si. Otra cosa a notar es que para el procesamiento de las formas se eligió trabajar directamente en escala de grises ya que es mas sencillo. Así queda separado en submarcos. Después se taclea el problema de detectar la forma de las piezas, para esto se utilizará la función iblobs.

Esto se logra utilizando las funciones “filterDotsAndArea” y “detectForm”. Devolviendo un vector con las formas. Luego se imprimen las formas de manera matricial y se cambian los colores originales por los propuestos por la consigna con la función “changeCcolors” y finalmente mostrar el marco final.

```
%Posibles formas
nada=0;rectangulo=1;triangulo=2;
circulo=3;undefined=4;
%Muestro cuadro original
idisp(basecolor)
title('Original')
%Separo en marcos R G y B la imagen original
```

```
fotor=(basecolor(:,:,1));
fotog=(basecolor(:,:,2));
fotob=(basecolor(:,:,3));

mnr = irank(fotor, 4,1);
mng = irank(fotog, 4,1);
mnb = irank(fotob, 4,1);

basecolor_nl=basecolor.*0;%armo un cuadro de iguales dimensiones
basecolor_nl(:,:,1)=mnr;
basecolor_nl(:,:,2)=mng;
basecolor_nl(:,:,3)=mnb;
%Asigno los marcos filtrados

figure()
idisp(basecolor_nl)
title('Filtrado')%%Muestro el filtrado

gray_figures=rgb2gray(basecolor_nl);
iblobs(gray_figures)%%Lo paso a grayscale

figs=zeros(100,100,16);

for i = 1:4
    for j=1:4
        id=(i-1)*110+1;
        jd=(j-1)*110+1;
        figs(:,:,(i-1)*4+j)=gray_figures(id:(id+99),jd:(jd+99));%Lo separo en marcos
    end
end


forms=[];
%Hasta aca agarro los cuadros por separados y filtrados
colored_frame=basecolor_nl;
%copio el marco filtrado

for i=1:16
    blob_data = filterDotsAndArea(figs(:,:,i));
    %Saco los vacios
    forms(i)=detect_form(blob_data);
    %Detecto formas
end
print_forms(forms)
%Los imprimo
new_colors=change_colors(forms,basecolor_nl);
%Cambio los colores acorde a la consigna
figure()
idisp(new_colors)
title('Colores Cambiados')
```

Aqui se ve la salida del programa con la imagen filtrada y el reconocimiento de las figuras

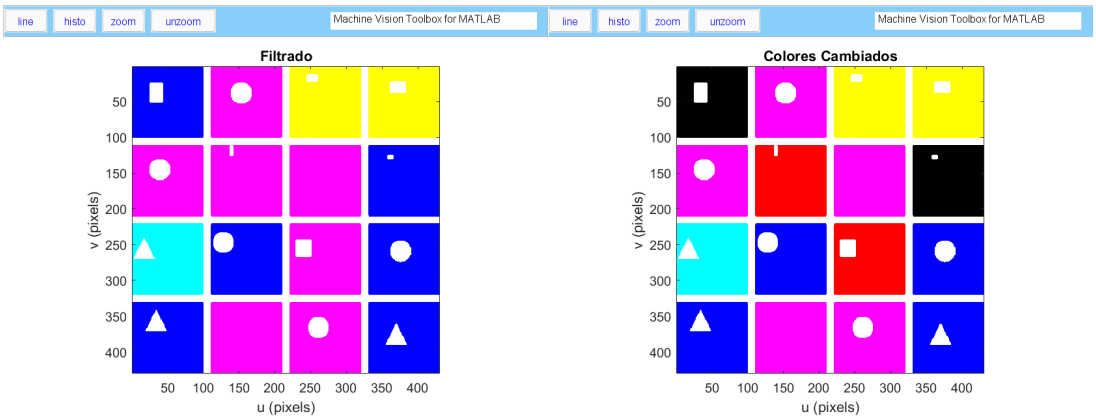

Figura 2: Comparativa filtrada y salida.

```
"Rectangulo"    "Circulo"       "Rectangulo"    "Rectangulo"
"Circulo"       "Rectangulo"    "Vacio"         "Rectangulo"
"Triangulo"     "Circulo"       "Rectangulo"    "Circulo"
"Triangulo"     "Vacio"         "Circulo"       "Triangulo"
```

Figura 3: Command window.

3.2. FilterDotsAndArea

Ahora explicaré la función "filterDotsAndArea". Aquí lo que se hace es fijarse las áreas, dado que siempre tiene un tamaño fijo los submarcos se puede saber el área si el marco esta vacío sin ningún tipo de ruido, lo que se hizo fue agregar unos thresholds para asegurarme de que aunque haya algo de ruido remanente todavía funcione. Para saber las areas se utiliza al función iblobs, esta devuelve un vector de blobs, lo que se hace aqui es sacar los blobs correspondientes a vaciío y unitarios.

```
1
2  function blobData = filterDotsAndArea(fig)
3  %aqui solo saco los que son vacios
4  blobData=iblobs(fig);
5  [~, blobCount]=size(blobData);
6  if blobCount>1
7      i=1;
8      while(blobCount-1)
9          if((blobData(1,i).area>9000) || (blobData(1,i).area<10))
10             blobData(i)=[];
11             blobCount=blobCount-1;
12         else
13             i=i+1;
14         end
15     end
16
17 end
18 end
```

3.3. detect form

Próximo es el código de la función detect form: La estrategia fue buscar cualidades características en las formas. Algo que se noto es que los círculos y triángulos todos mantienen su forma y área(teniendo en cuenta el ruido remanente). Y que los cuadrados no podrán tener un tamaño inferior a 5x5 y no mayor a 30x30. La diferenciación de los triangulo se hace mediante su relación de aspecto. Debido a que siempre es un triangulo isósceles la relación $\frac{b}{a}$ será la misma.

Luego se diferencian los rectángulos a partir de su área estimada. utilizando el ancho y alto del cuadrado se la estima y luego se la compara con la devuelta por iblobs. Para el circulo basta con que la relación $\frac{b}{a}$ sea aproximadamente 1. Claro que esto sería un problema si se lo enfrenta con un cuadrado perfecto ya que su relación ba es 1 también, pero estos son descartados antes ya que son catalogados como rectángulos antes gracias a la aproximación del área. Adicional mente cabe mencionar que el triangulo también se lo podría diferenciar por la área, en vez de por la relación ba. Esta opción sería mas robusta antes triángulos de diversos ángulos internos. Pero debido a que en este problema esa situación no se da no se optó por utilizar.

```
function form =detect_form(blob_data)
nada=0; rectangulo=1; triangulo=2; circulo=3; undefined=4;
form=undefined;%defualt es indefinido
ba=blob_data.b/blob_data.a;%saco la relacion ba
H=blob_data.vmax-blob_data.vmin;
%Aharro el ancho y alto
W=blob_data.umax-blob_data.umin;

delta_area=abs(H*W-blob_data.area);
%calculo el delta de area
threshold_area_rectangulo=50;
if((blob_data.area>9000) || (blob_data.area<10))
    form=nada;%%No era nada
elseif(delta_area< threshold_area_rectangulo)
    form=rectangulo; %%Aproximo el area, si da cerca era un rectangulo
elseif (ba>0.99)
    form=circulo;%%Este es el caso para diferenciar del cuadrado perfecto

elseif ((ba> 0.851) && (ba < 0.887))
    form=triangulo;%%Me fijo la razon del triangulo dado que siempre e sle mismo tipo
        de triangulo
    %en el caso de que haya triangulos de diversos angulos, utilizaria
    %una verificacion similar a las otras donde me fijo si es de un
    %area similar
    %unicamente la  verificacion de areas
end
end
```

3.4. print forms

Print forms: recibe el vector de formas e imprime de manera estética la matriz de formas.

```
function print_forms(forms)
words=["","","","","";
      "","","","","";
      "","","","","";
      "","","","",""];
nada= 0;
rectangulo=  1;
triangulo =2;
circulo  =3;
undefined=4;

for i=1:4
    for j=1:4
        if(forms((i-1)*4+j) ==nada)
            words(i,j)="Vacio";
        elseif(forms((i-1)*4+j) ==rectangulo)
            words(i,j)="Rectangulo";
        elseif(forms((i-1)*4+j) ==triangulo)
            words(i,j)="Triangulo";
        elseif(forms((i-1)*4+j) ==circulo)
```

```
21              words(i,j)="Circulo";
22          elseif(forms((i-1)*4+j) ==undefined)
23              words(i,j)="Indedfinido";
24          end
25
26      end
27  end
28
29  disp(words)
30  end
```

3.5. change colors

Finalmente queda la función change colors: Esta recibe el marco filtrado original y el vector de formas. Y devuelve el marco con colores nuevos. Utiliza el vector de formas para saber que forma está mirando (acorde a los i, j) y así decide que hacer en el switch, sea caso rectángulo, triangulo, circulo o nada. Se recorre cada bit preguntándose si es blanco o no, para saber si se multiplica por cero o no el valor.

```
1   function new_colors = change_colors(forms,figure)
2   new_colors=figure;
3   for i=1:4
4       for j=1:4
5           id=(i-1)*110+1;
6           jd=(j-1)*110+1;
7           switch forms((i-1)*4+j)
8               case 1%Si es un rectangulo saco el azul
9                   for idd=0:99
10                      for jdd=0:99
11                          if(is_not_white(new_colors(id+idd,jd+jdd,:)))
12                              new_colors(id+idd,jd+jdd,3)= new_colors(id+idd,jd+jdd,3)
                                    *0;
13                          end
14
15                      end
16                  end
17              case 2%si e sun triangulo saco el rojo
18                  for idd=0:99
19                      for jdd=0:99
20                          if(is_not_white(new_colors(id+idd,jd+jdd,:)))
21                              new_colors(id+idd,jd+jdd,1)= new_colors(id+idd,jd+jdd,1)
                                    *0;
22                          end
23                      end
24                  end
25              case 3%si es un circulo saco el verde
26                  for idd=0:99
27                      for jdd=0:99
28
29                          if(is_not_white(new_colors(id+idd,jd+jdd,:)))
30                              new_colors(id+idd,jd+jdd,2)= new_colors(id+idd,jd+jdd,2)
                                    *0;
31                          end
32                      end
33                  end
34          end
35      end
36
37  end
38  end
```

3.6. is not white

La función is not white hace exactamente lo que el nombre sugiere. Recibe el valor de un bit RGB y devuelve 1 si no es blanco y 0 si es blanco.

```
function not_white=is_not_white(val)
    if((round(val(1),2)==1) && (round(val(2),2)==1) && (round(val(3),2)==1))
        not_white=0;
    else
        not_white=1;
    end
end
```