PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE UN ALGORITMO PARA EL REGISTRO DE IMÁGENES AÉREAS ORTORECTIFICADAS APLICADO EN LA SUPERVISIÓN AGRÍCOLA

Tesis para optar el Título de **INGENIERO ELECTRÓNICO,** que presenta el bachiller:

Fernando Javier Meza Zurita

ASESOR: Andrés Flores Espinoza

Lima, noviembre de 2014

Índice

Anexo 1: Mosaicos desarrollados a partir de las imágenes de prueba

1.1 Cambios de Desenfoque

Imagen 1

Imagen 2

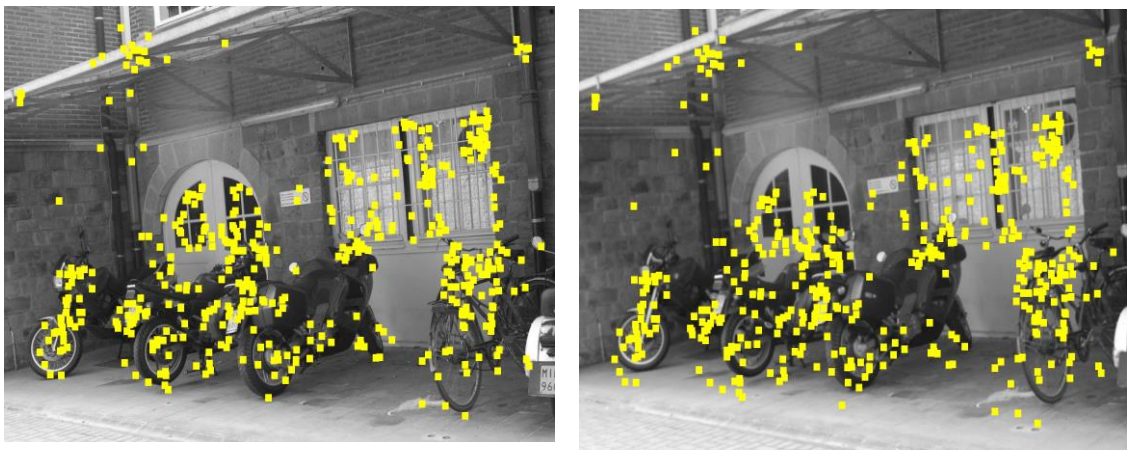
Imagen 3

Imagen 4

Imagen 5

Imagen 6

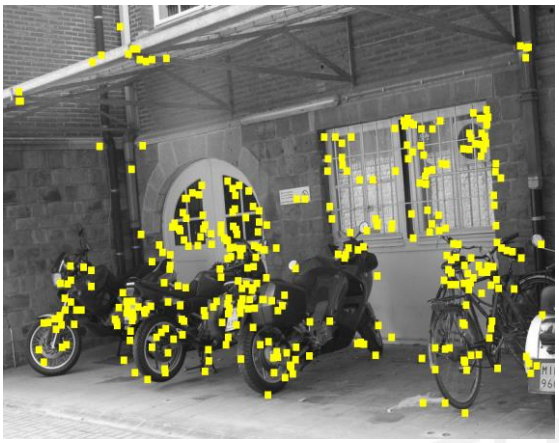
Figura 1: Puntos de control seleccionados para el banco de imágenes con efectos de desenfoque. El efecto de desenfoque se va creciendo desde la imagen 2 a la imagen 6. En la imagen 6 se puede ver el peor efecto de desenfoque con el que se trabaja.



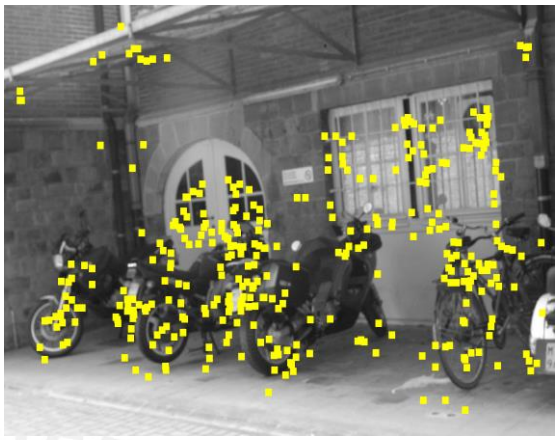
(a) (b)

(c)

Figura 2: Mosaico generado a partir de imágenes 1 y 2. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 2 con puntos de correspondencia hallados. (c) Mosaico desarrollado.



(a)

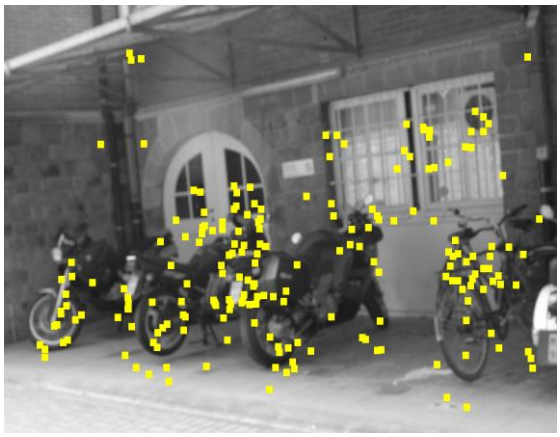


(b)

(c)

Figura 3: Mosaico generado a partir de imágenes 1 y 3. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 3 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

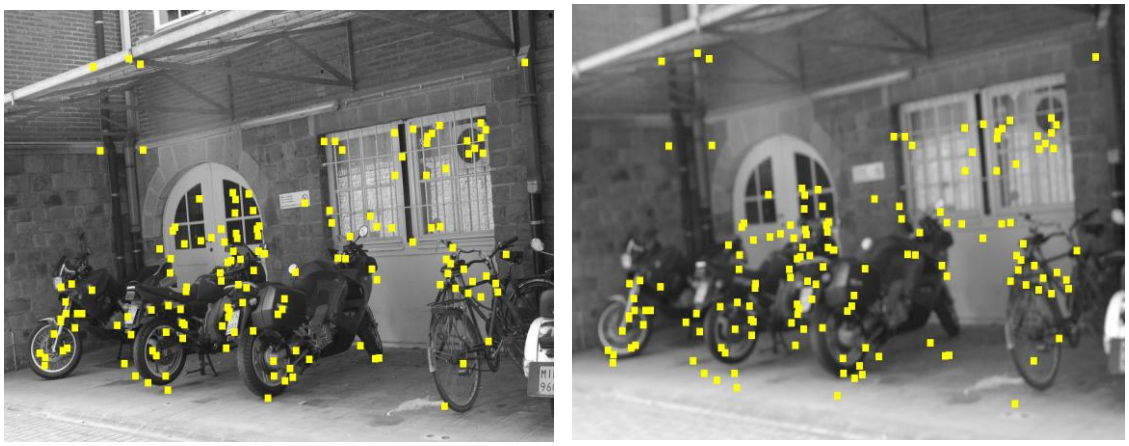
(a)



(b)

(c)

Figura 4: Mosaico generado a partir de imágenes 1 y 4. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 4 con puntos de correspondencia hallados. (c) Mosaico desarrollado.



(a) (b)

(c)

Figura 5: Mosaico generado a partir de imágenes 1 y 5. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 5 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a)

(b)

(c)

Figura 6: Mosaico generado a partir de imágenes 1 y 6. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 6 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

1.2 Cambios de escala y rotación

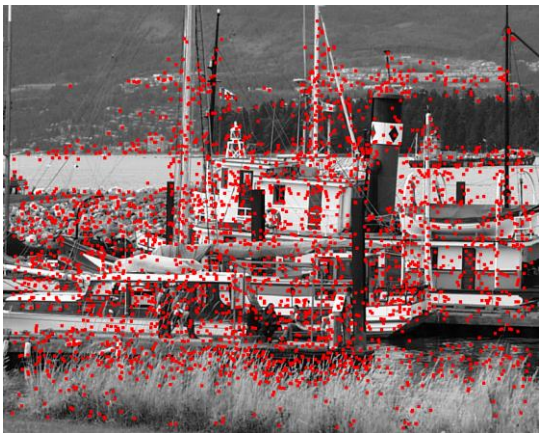


Imagen 1

Imagen 2

Imagen 3

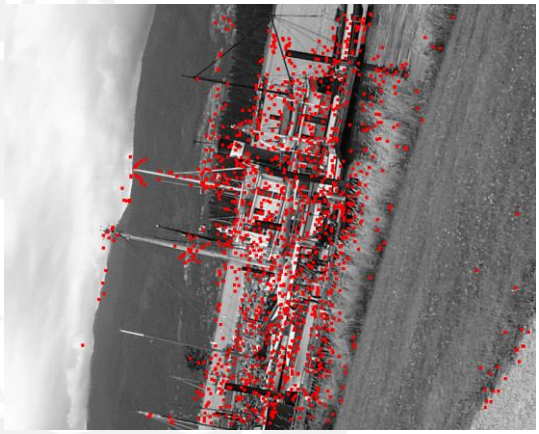


Imagen 4

Imagen 5

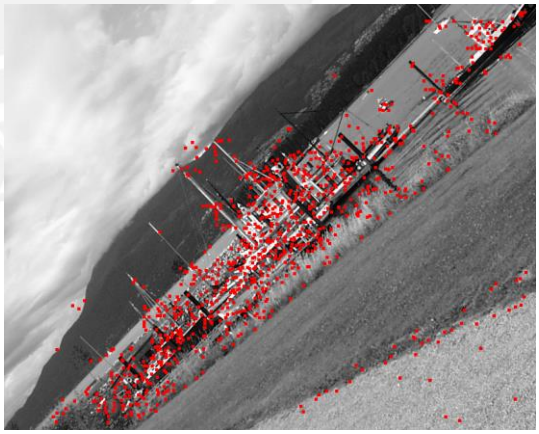
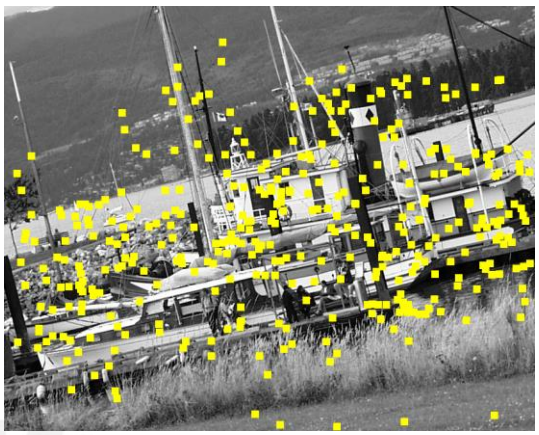


Imagen 6

Figura 7: Puntos de control seleccionados para el banco de imágenes con efectos de escala y rotación. El efecto de rotación y escala va creciendo desde la imagen 2 a la imagen 6. En la imagen 6 se puede ver el peor efecto de escala (doble) con el que se trabaja.

(a)

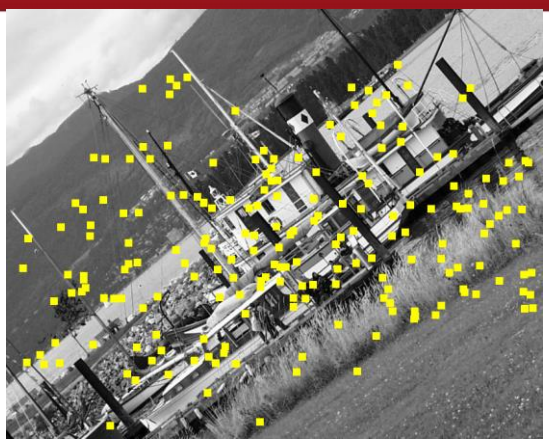


(b)

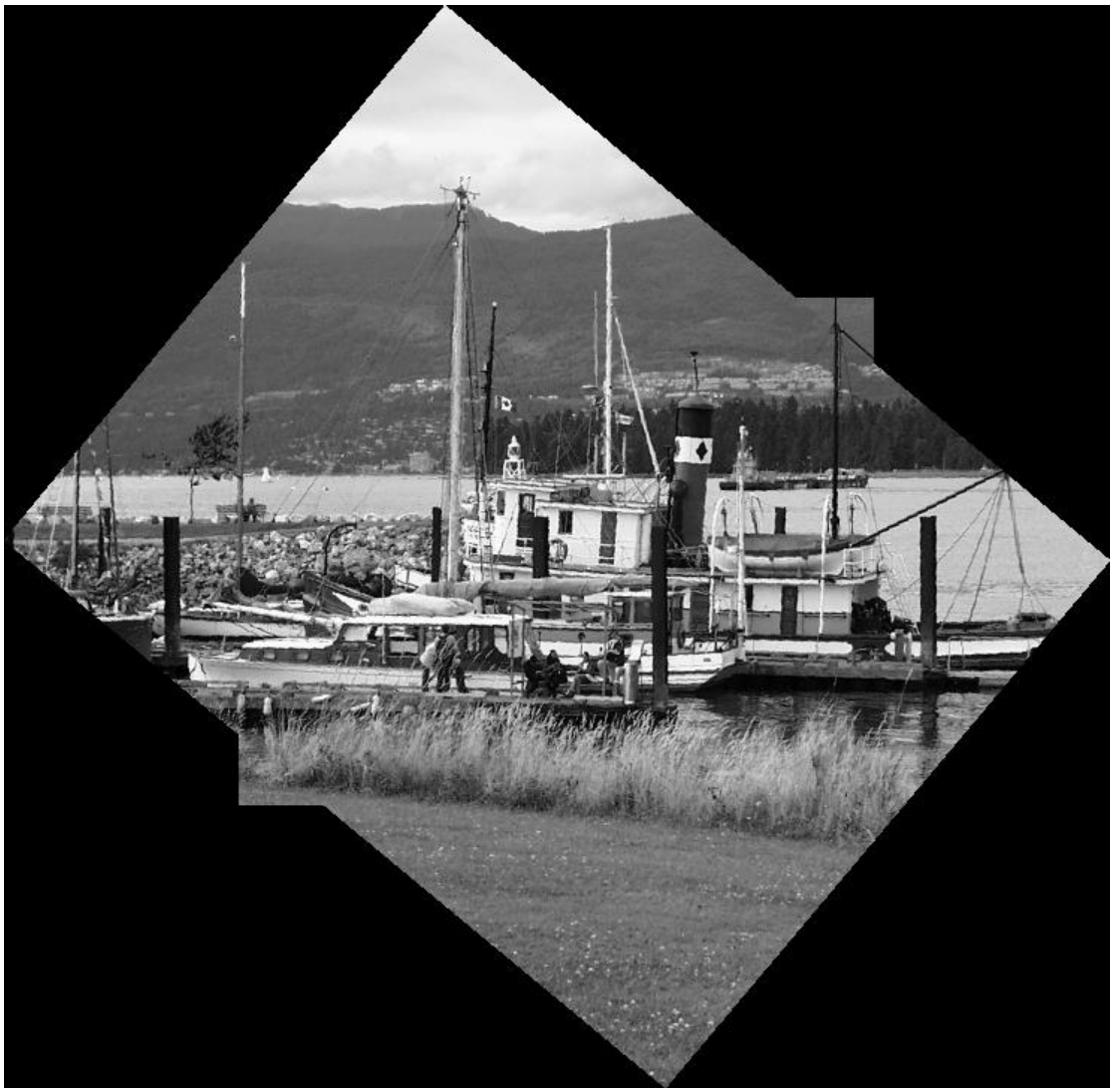
(c)

Figura 8: Mosaico generado a partir de imágenes 1 y 2. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 2 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a)

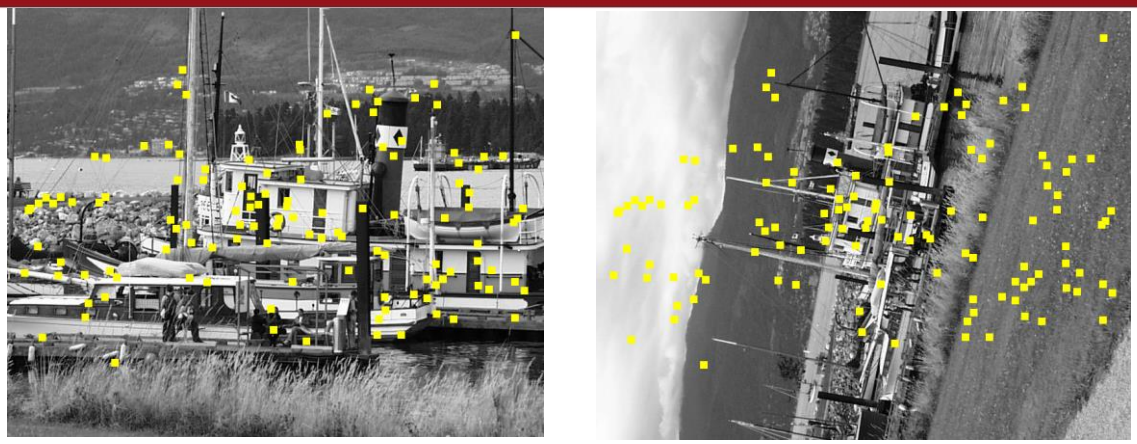


(b)

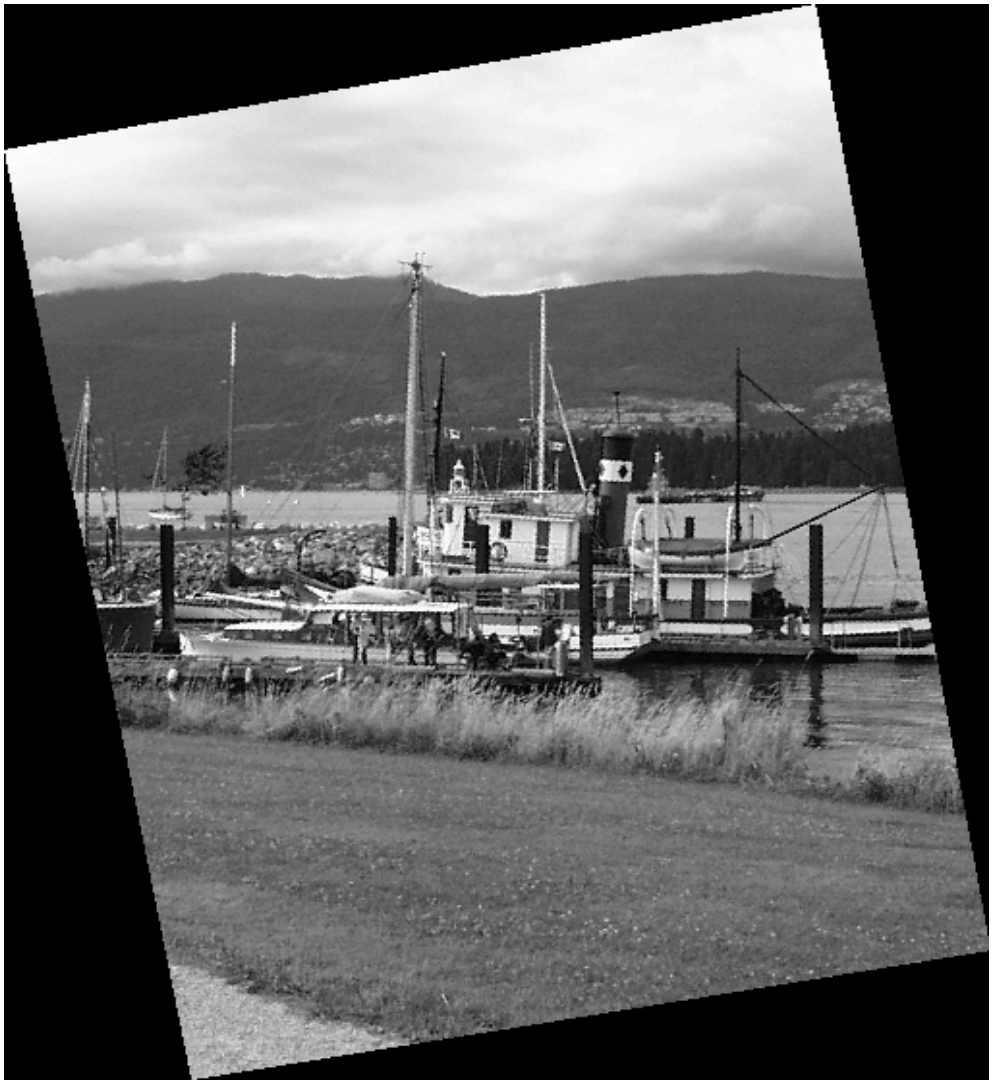


(c)

Figura 9: Mosaico generado a partir de imágenes 1 y 3. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 3 con puntos de correspondencia hallados. (c) Mosaico desarrollado.



(a) (b)



(c)

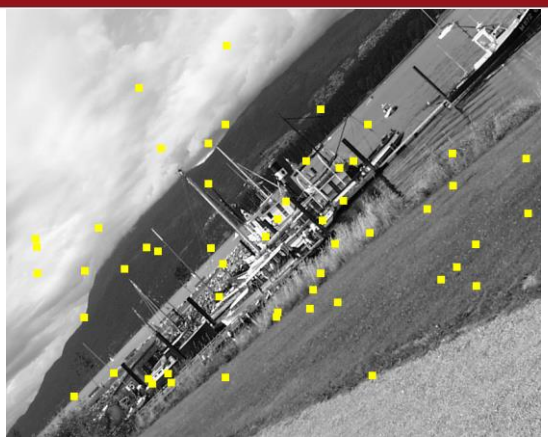
Figura 10: Mosaico generado a partir de imágenes 1 y 4. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 4 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a) (b)

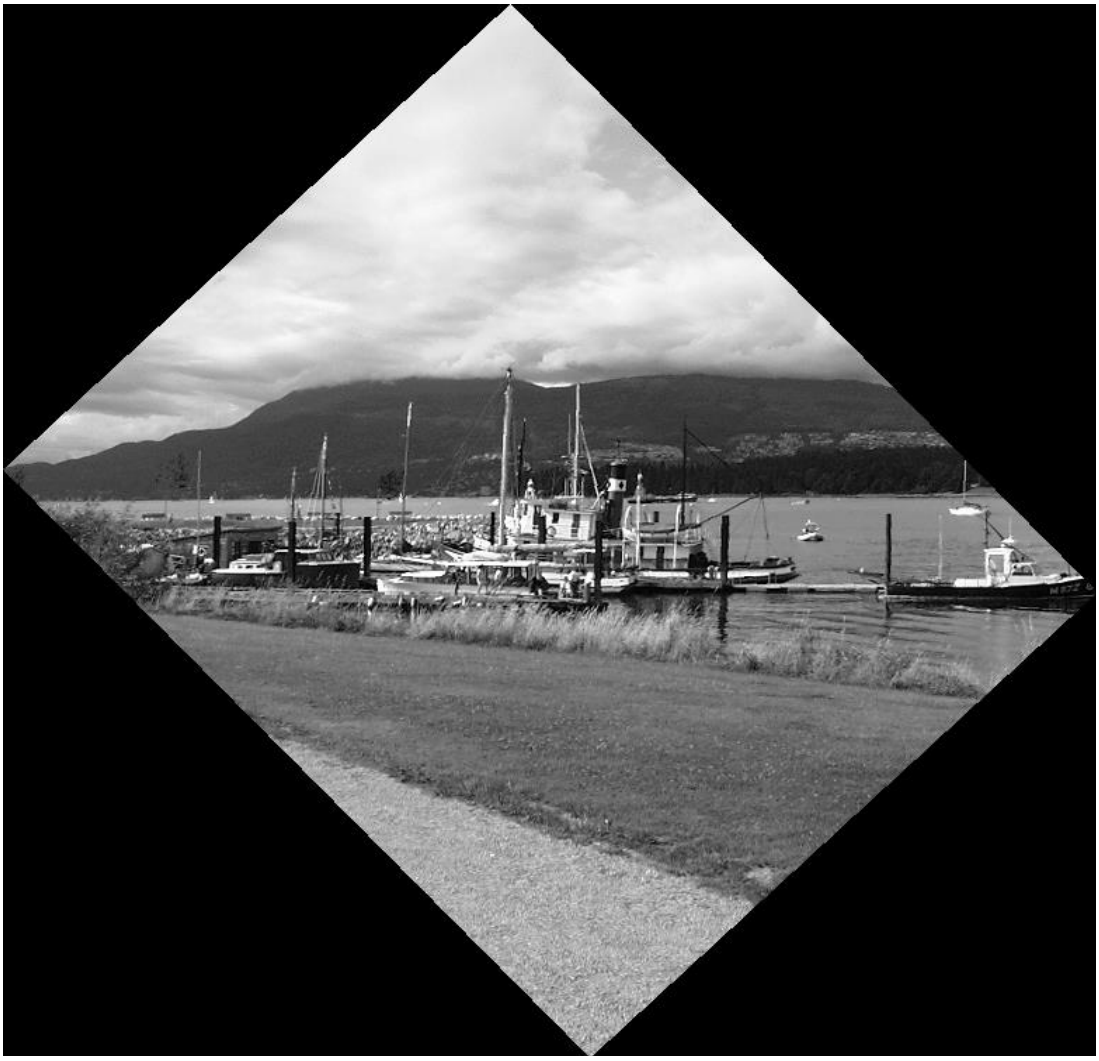
(c)

Figura 11: Mosaico generado a partir de imágenes 1 y 5. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 5 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a)



(b)



(c)

Figura 12: Mosaico generado a partir de imágenes 1 y 6. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 6 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

1.3 Cambios de perspectiva

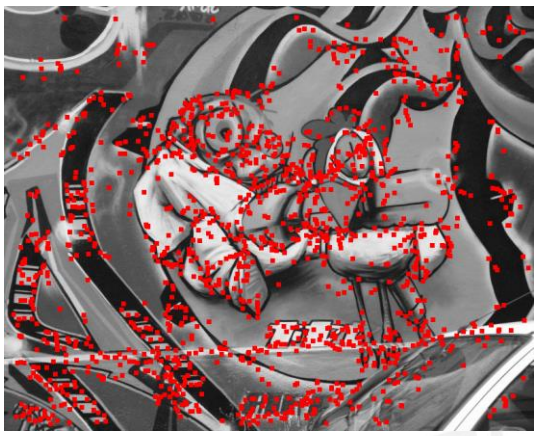


Imagen 1

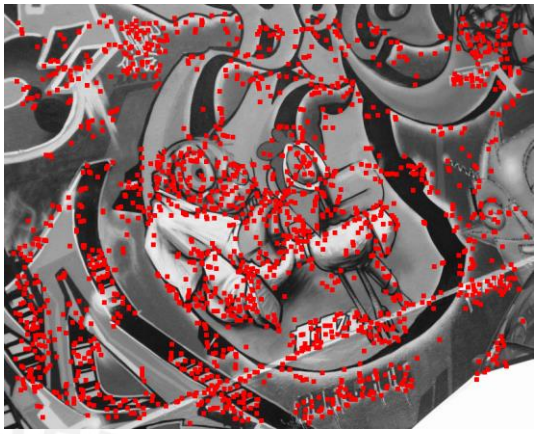


Imagen 2

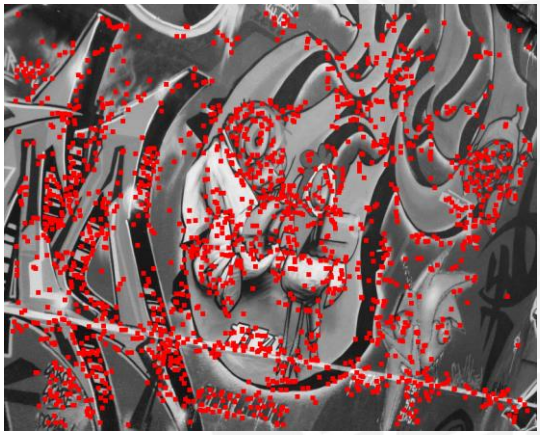


Imagen 3

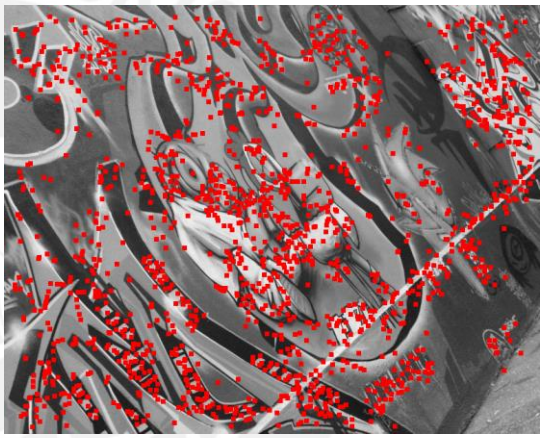


Imagen 4

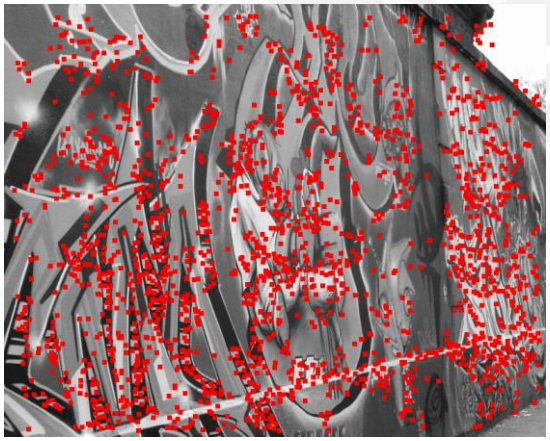


Imagen 5

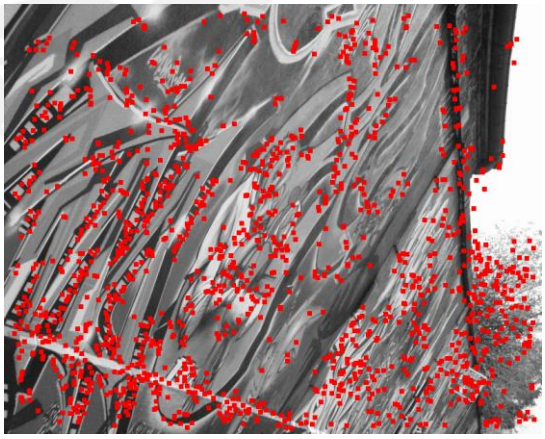
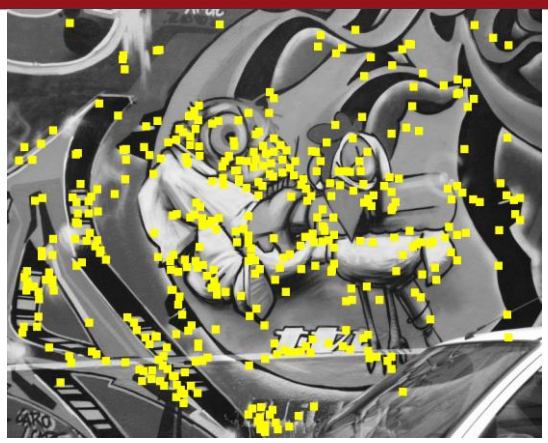
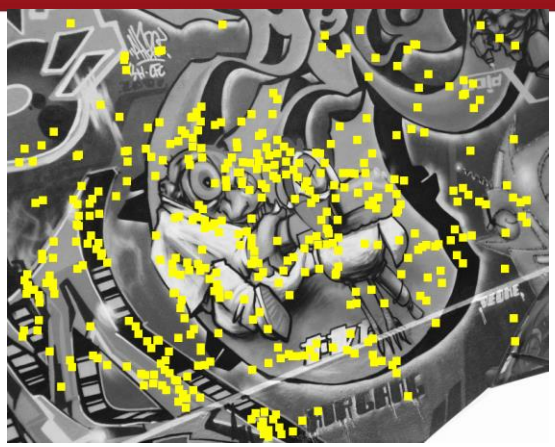


Imagen 6

Figura 13: Puntos de control seleccionados para el banco de imágenes con efectos de perspectiva. El efecto de perspectiva va creciendo desde la imagen 2 a la imagen 6. En la imagen 6 se puede ver el peor efecto de perspectiva con el que se trabaja.



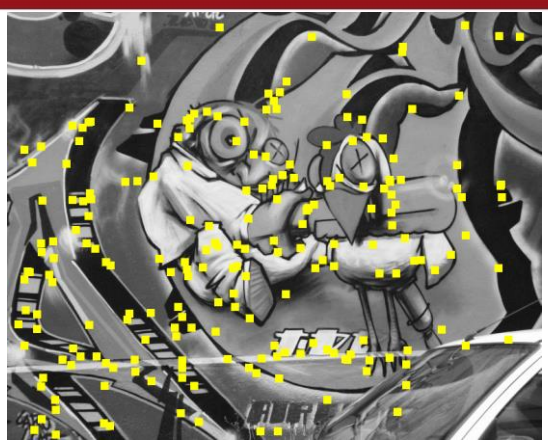
(a)



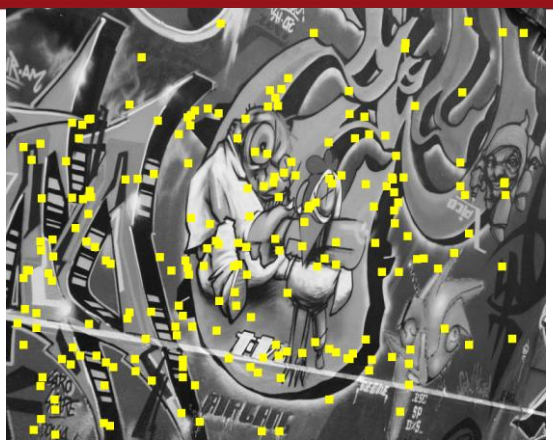
(b)

(c)

Figura 14: Mosaico generado a partir de imágenes 1 y 2. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 2 con puntos de correspondencia hallados. (c) Mosaico desarrollado.



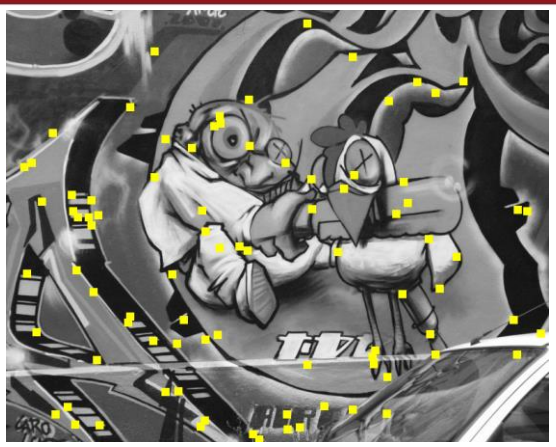
(a)



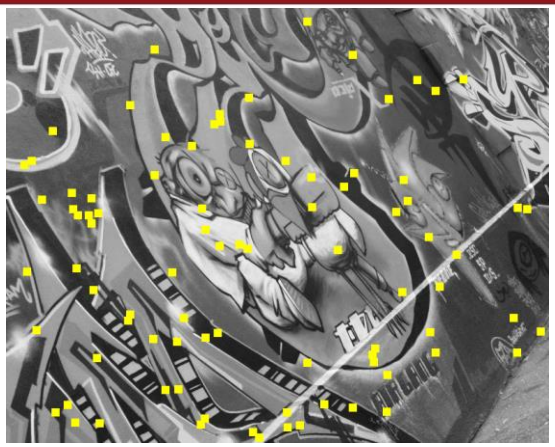
(b)

(c)

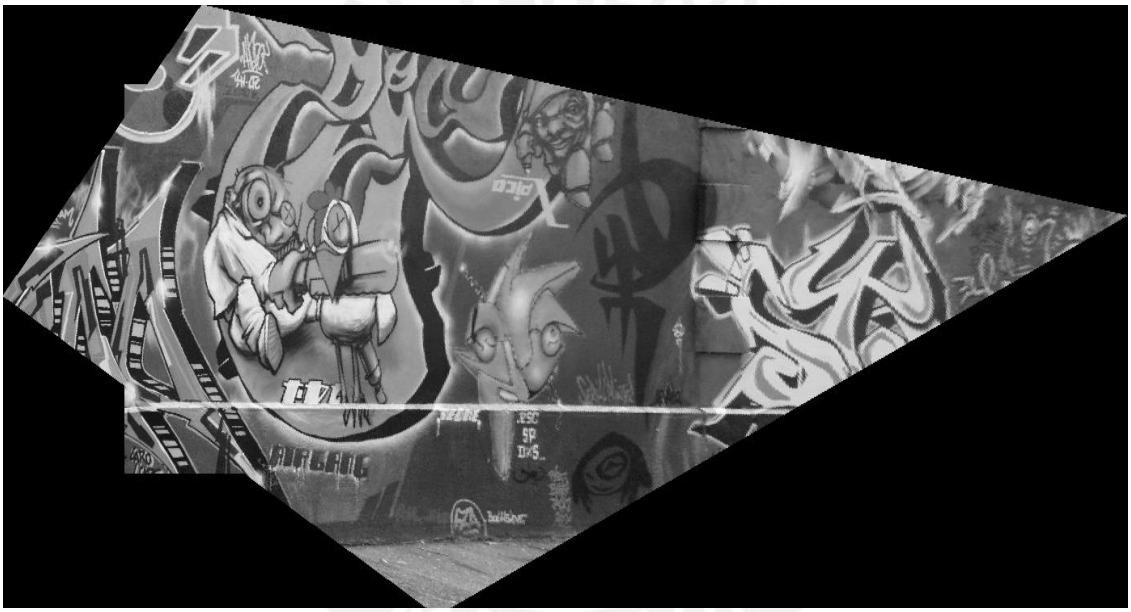
Figura 15: Mosaico generado a partir de imágenes 1 y 3. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 3 con puntos de correspondencia hallados. (c) Mosaico desarrollado.



(a)



(b)



(c)

Figura 16: Mosaico generado a partir de imágenes 1 y 4. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 4 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

1.4 Cambios de iluminación

Imagen 1

Imagen 2

Imagen 3

Imagen 4

Imagen 5

Imagen 6

Figura 17: Puntos de control seleccionados para el banco de imágenes con efectos de iluminación. El efecto de iluminación va creciendo desde la imagen 2 a la imagen 6. En la imagen 6 se puede ver la imagen más oscura con la que se trabaja.

(a)

(b)

(c)

Figura 18: Mosaico generado a partir de imágenes 1 y 2. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 2 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a) (b)

(c)

Figura 19: Mosaico generado a partir de imágenes 1 y 3. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 3 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a) (b)

(c)

Figura 20: Mosaico generado a partir de imágenes 1 y 4. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 4 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a) (b)

(c)

Figura 21: Mosaico generado a partir de imágenes 1 y 5. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 5 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

(a)

(b)

(c)

Figura 22: Mosaico generado a partir de imágenes 1 y 6. (a) Imagen 1 con puntos de correspondencia hallados. (b) Imagen 6 con puntos de correspondencia hallados. (c) Mosaico desarrollado.

Anexo 2: Resultados del banco de imágenes utilizando lenguaje C

2.1 Cambios de desenfoque

Tabla1: Diferencia entre los parámetros de la matriz hallada y la matriz real

	I1-I2	I1-I3	I1-I4	I1-I5	I1-I6
a	-0.013 E-01	-0.038 E-01	6.740 E-04	0.048 E-01	-0.027 E 01
b	-0.020 E-01	-0.017 E-01	-9.941 E-04	0.077 E-01	-0.036 E-01
c	-0.878 E00	-0.158 E01	-0.286 E00	-1.016 E00	9.147 E00
d	6.296 E-04	0.012 E-01	-7.141 E-05	0.020 E-01	-0.078 E-01
e	-0.027 E-01	-0.019 E-01	-0.021 E-01	0.016 E-01	0.080 E -01
f	9.633 E-01	8.406 E-01	-0.483 E00	3.810 E-01	3.230 E00
g	-6.420 E-08	-2.624 E-06	-2.069 E-07	-8.247 E-06	-2.401 E-05
h	-3.032 E-06	-1.305 E-06	2.246 E-06	1.021 E-05	1.217 E-06

Tabla2: Información del funcionamiento del algoritmo ante cambios de enfoque

	Descriptores		Descriptor en común	% de Aciertos	Error promedio	Tiempo de procesamiento
	Imagen 1	Imagen 2				
I1-I2	1053	732	597	71.5	5.26	2.44
I1-I3	1053	637	519	70.5	6.06	2.50
I1-I4	1053	451	362	72.1	7.5	2.38
I1-I5	1053	344	311	73.4	8.4	2.43
I1-I6	1053	233	180	75.2	9.8	2.35

2.2 Cambios de escala y rotación

Tabla3: Diferencia entre los parámetros de la matriz hallada y la matriz real

	I1-I2	I1-I3	I1-I4	I1-I5	I1-I6
a	-3.016 E-04	-0.036 E-01	-2.440 E-04	0.075 E-01	-0.071 E-01
b	-0.015 E-01	-0.102 E-01	-0.146 E-01	0.014 E-01	-1.83 E 00
c	-0.473 E00	1.939 E00	2.225 E 00	-0.314 E 01	562.7 E00
d	-6.992 E-04	0.042 E-01	0.050 E-01	0.025 E-01	-0.247 E00
e	1.975 E-04	-0.097 E-01	-0.024 E-01	0.158 E-01	0.323 E00
f	0.297 E00	-1.077 E00	-3.993 E00	-0.231 E 01	-327.3 E00
g	-1.442 E-06	-7.320 E-07	1.508 E-06	2.193 E-06	1.88 E-04
h	1.980 E-06	1.959 E-05	-2.203 E-05	3.223 E-05	-0.020 E-01

Tabla4: Información del funcionamiento del algoritmo ante cambios de escala y rotación

	Descriptores		Descriptor en común	% de Aciertos	Error promedio	Tiempo de procesamiento
	Imagen 1	Imagen 2				
I1-I2	2880	2860	941	70.7	14.2	4.04
I1-I3	2880	2105	649	72.5	15.4	4.9
I1-I4	2880	1390	479	65.8	23.88	5.45
I1-I5	2880	1259	386	67.4	22.26	7.4
I1-I6	2880	983	255	-	-	-

2.3 Cambios de perspectiva

Tabla5: Diferencia entre los parámetros de la matriz hallada y la matriz real

3	I1-I2	I1-I3	I1-I4	I1-I5	I1-I6
a	-0.011 E-01	-0.077 E-01	0.152 E-01	-0.836 E 00	-0.456 E 00
b	-0.021 E-01	-7.419 E-04	-0.251 E-01	-0.636 E00	0.115 E 00
c	0.432 E 00	0.084 E 00	5.362 E 00	113.22 E00	-230.5 E00
d	5.836 E-04	-0.041 E-01	0.021 E00	-0.322 E00	-0.522 E00
e	-0.045 E-01	-0.075 E-01	-0.420 E-01	-1.205 E00	-2.478 E00
f	0.271 E 00	1.396 E 00	5.991 E 00	112.25 E00	634.39 E00
g	-8.186 E-07	-1.505 E-05	7.655 E-05	-9.762 E-04	-6.569 E-04
h	-5601 E-06	-6.839 E-05	-8.147 E-05	-0.020 E-01	-0.024 E-01

Tabla6: Información del funcionamiento del algoritmo ante cambios de perspectiva

	Descriptores		Descriptor en común	% de Aciertos	Error promedio	Tiempo de procesamiento
	Imagen 1	Imagen 2				
I1-I2	1531	1783	584	70.3	12.33	3.23
I1-I3	1531	1847	275	75.06	18.8	3.8
I1-I4	1531	1657	159	72.43	33.8	5.35
I1-I5	1531	1910	136	-	-	-
I1-I6	1531	1497	120	-	-	-

2.4 Cambios de iluminación

Tabla7: Diferencia entre los parámetros de la matriz hallada y la matriz real

	I1-I2	I1-I3	I1-I4	I1-I5	I1-I6
a	-4.216 E-01	0.425 E 00	-4.231 E-01	1.580 E 00	0.418 E00
b	-2.450 E-04	0.017 E-01	0.017 E-01	0.179 E-01	0.031 E-01
c	-0.185 E 01	2.071 E00	3.393 E 00	0.550 E 00	0.871 E00
d	-0.017 E-01	-1.547 E-04	-9.879 E-04	-0.025 E-01	7.773 E-04
e	-4.225 E-01	4.260 E-01	-0.425 E 00	1.587 E00	0.421 E 00
f	0.131 E 01	-1.829 E00	-3.748 E00	-12.05 E00	-6.642 E00
g	1.043 E-06	-1.933 E-06	-3.430 E-06	-4.131 E-06	-2.736 E-07
h	-2.800 E-06	3.725 E-06	6.823 E-06	-2.358 E-05	6.558 E-06

Tabla8: Información del funcionamiento del algoritmo ante cambios de iluminación

	Descriptores		Descriptor en común	% de Aciertos	Error promedio	Tiempo de procesamiento
	Imagen 1	Imagen 2				
I1-I2	1146	833	594	47.1	29.78	2.1
I1-I3	1146	692	476	47.9	41.83	2.04
I1-I4	1146	572	397	50.9	51.42	2.06
I1-I5	1146	457	301	53.1	59.73	1.95
I1-I6	1146	363	227	55.4	66.06	2.02

Anexo 3: Programa en Matlab

3.1 Programa Principal

```
%% Ortorectificación y mosaico de imágenes
%
% Corrige la distorsión de las imágenes generado por el
% movimiento del UAV.
% Relaciona múltiples imágenes que tengan un espacio en común.
% Se siguen los siguientes pasos:
% - Obtención de puntos de control a partir de características
% invariantes  en las imágenes.
% - Extracción de descriptores mediante el algoritmo SURF
% -Correspondencia de dichos descriptores mediante el algoritmo
% NNDR
% - Estimación de la función de transformación: RANSAC.
% - Se construye el mosaico a partir de la función de
%  transformación y se repite el proceso, dependiendo del modo
% el proceso puede realizarse utilizando de referencia el
% mosaico o la imagen anterior.
%
% El programa necesita 3 variables de entrada, las cuales se
% especifican a continuación:
% Ruta del directorio donde se encuentran las imágenes para
% realizar el registro.
% Ruta del directorio donde se quiere guardar el mosaico final.
% nombre del archivo de texto donde se encuentran los ángulos de
% yaw, pitch y roll.
% modo del registro es un dato opcional usado para pruebas.
% modo = 1 significa registro con respecto a la imagen anterior
% como referencia.
% modo = 2 significa registro con respecto al mosaico total como
% referencia.

close all
clear all

% Nombre del directorio donde se ubica la carpeta con las
% imágenes

 directorio='./tesis/vuelo_pruebas2/';
% Modo de ejecucion del algoritmo
 mode=1;
%tipo de imagen (.bmp o .png)
data_type='.bmp';

% Lectura del archivo de texto con los ángulos y la hora a la
% que fueron tomados

[archivo,ruta]=uigetfile('*.txt','ABRIR ARCHIVO con angulos');
fid =fopen([ruta archivo],'r');
A=textscan(fid,'%s %10s %s \n %s %f %s %f %c %c %f %s %s %s %s
%s %s %s %s %s %s','headerlines',2);
A{1,1} = A{1,1}(1:end-3,1);
A{1,2} = A{1,2}(1:end-3,1);
```

```
[M,N]=size(A{1,1});
Angulo = zeros([M 3]);
hora_angulo = cellstr(A{1,2});
Angulo(:,1)=A{1,10}(1:M,1); % yaw
Angulo(:,2)=A{1,7}(1:M,1); % pitch
Angulo(:,3)=A{1,5}(1:M,1); % roll
fclose(fid);

% Lectura del archivo de texto con la hora a la que fueron
% tomados las imágenes.

[archivo,ruta]=uigetfile('*.txt','ABRIR ARCHIVO con imagenes');
fid =fopen([ruta archivo],'r');
A=textscan(fid,'%5s.%s %s %10s %s\n' ,'headerlines',1);
nombre_imagen = cellstr(A{1,1});
hora_imagen = cellstr(A{1,4});
fclose(fid);


% Lectura de las imágenes y el almacenamiento de las mismas para
% su posterior análisis

lee_archivos = dir(strcat(directorio,'*',data_type)); %el
formato de imagen puede ser
modificado.I=cell(1,length(lee_archivos));
name=cell(1,length(lee_archivos));
I=cell(1,length(lee_archivos));

Ang = zeros([length(lee_archivos) 3]);

for k = 1:length(lee_archivos) %recorre n?mero de archivos
guardados en el directorio
archivo = lee_archivos(k).name; %Obtiene el nombre de los
archivos
I{1,k} = imread(strcat(directorio,archivo));% lee la primera
imagen
 C = strsplit(archivo,'.');
 name=C{1,1};

% Referencia de las imágenes con la hora para obtener el ángulo
% al que fue tomada cada imagen.

 idx = strfind(nombre_imagen, name);
   [M,N]=size(idx);
   for i=1:1:M
        if (idx{i,1}==1)
            indice=i;
        end
   end
  hora = hora_imagen{indice,1};

  idx = strfind(hora_angulo, hora);

  [M,N]=size(idx);
```

```
 for i=1:1:M
        if (idx{i,1}==1)
            indice=i;
        end
  end

   Ang(k,:)=Angulo(indice,:);
end


figure; plot(Ang); % visualización de los ángulos

Ang(:,1)=Ang(:,1)-Ang(1,1);

% Iniciar variables

        error_cuad_prom=zeros(1,k-1);
        error_prom=zeros(1,k-1);
        good_pixel_perc=zeros(1,k-1);


% Tomar de la primera imagen como referencia

        [~,~,ColorChannel] = size(I{1,1});

% Transformación de la imagen a escala de grises

     if ColorChannel > 1
            img = rgb2gray(I{1,1});
        else
            img=I{1,1};
        end

% Corrección de la primera imagen

        yaw   = -Ang(1,1);
        pitch = Ang(1,2);
        roll  = Ang(1,3);

         img=imcorrect(img,yaw,roll,pitch);

% La referencia inicial siempre será la primera imagen
% independiente del modo.
% la primera imagen siempre será el inicio del mosaico también

        imref = img;
        mosaico=img;


%% Armado del mosaico a partir de la segunda imagen

for k = 2:length(lee_archivos)

% Transformación de imágenes a escala de grises

 [~,~,ColorChannel] = size(I{1,k});
```

```
        if ColorChannel > 1
            img = rgb2gray(I{1,k});
        else
            img=I{1,k};
        end

 % Asignación de ángulos

    yaw   =   -Ang(k,1);
    roll =    Ang(k,3);
    pitch  =  Ang(k,2);

% Corrección de la imagen con usando los ángulos obtenidos en el
% vuelo.

     img = imcorrect(img,yaw,roll,pitch);

    if mode==1

% Se limita la imagen al área de interés pero se almacenan las
% coordenadas originales para realizar la traslación de los
% puntos a su ubicación original.

    [row,col] = find(imref>0);
     a = min(col(:));
     b = min(row(:));
     c = max(col(:));
     d = max(row(:));

     st=imcrop(imref, [a b c-a d-b]);

% Detección de puntos de control y extracción de descriptores

points01 = detectSURFFeatures(st,'MetricThreshold',1000);
[features01, valid_points01]= extractFeatures(st,points01);
[M, N]=size(valid_points01.Location);
Traslacion=ones([M N]);
cp01 = zeros([M N]);
cp01(:,1)= valid_points01.Location(:,1)+ Traslacion(:,1)*(a-1);
cp01(:,2)= valid_points01.Location(:,2)+ Traslacion(:,2)*(b-1);

    else

% Detección de puntos de control y extracción de descriptores

points01 = detectSURFFeatures(imref,'MetricThreshold',1000);
[features01, valid_points01] = extractFeatures(imref, points01);
cp01=double(valid_points01.Location);

    end

% Detección de puntos de control y extracción de descriptores

points = detectSURFFeatures(img,'MetricThreshold',1000);
[features02, valid_points02] = extractFeatures(img, points);
cp02=double(valid_points02.Location);
```

```
% Correspondencia entre los descriptores encontrados NNDR=0.8

 [indexPairs,distancia]= nndr(features01, features02,0.8);
  matchedPoints1 = cp01(indexPairs(:, 1), :);
  matchedPoints2 = cp02(indexPairs(:, 2), :);

% Estimación de la función de transformación

[tform,inlierPtsOut,inlierPtsIn] =
estimateGeometricTransform(matchedPoints1,matchedPoints2,..
'projective');
 bestHomography=tform.T';

% Generación del mosaico, el error en cada pixel de la imagen y
% generación de la imagen transformada.

 [mosaico,imtransf,error_pixel] = stitch(mosaico,img,..
bestHomography);


% Analisis de error

error_promedio=norm(error_pixel,1)/size(error_pixel,2);
error_prom(1,k-1)=error_promedio;
good_pixel_perc(1,k-1)=size(find(error_pixel<=error_promedio)...
,2)/size(error_pixel,2);

 % Asignación de la referencia

        if mode == 1
            imref = imtransf;

        else
            imref = mosaico;

        end

end

%Mostrar resultado final

    figure; imshow(uint8(mosaico))
```

3.2 Correspondencia de puntos de control

```
% Nearest Neighbor Distance Ratio (NNDR)
%
% Análisis de correspondencia entre descriptores a partir de
% distancias euclidianas entre los 64 componentes del
% descriptor. Se define que existe la correspondencia si es que
% la relación entre la menor distancia del conjunto y la segunda
% distancia más cercana está por debajo de un valor umbral.
%
% Parámetros de entrada:
%
% desc_a: Descriptores de la imagen de referencia
% desc_b: Descriptores de la imagen de muestra
% dist_ratio: Valor umbral
%
% Parámetros de salida:
%
% indexPairs: índice donde se ubican los puntos de control con
% correspondencia
% distancia: Relación de distancias entre pares de
correspondencia
%-----------------------------------------------------------------
----------

function [indexPairs, distancia] = nndr(desc_a,

desc_b,dist_ratio)

% correspondencias muestra/referencia
nn= zeros(size(desc_b,1),1);

% se almacena distancia del punto correspondiente
dist= nn;

for k= 1:size(desc_b,1)

% distancia euclidiana entre el primer descriptor de una imagen
% con respecto a los demás de la otra imagen.

eucl= sqrt(sum((desc_a-
repmat(desc_b(k,:),size(desc_a,1),1)).^2,2));

[d01, idx]= min(eucl); % distancia con el vecino más cercano
eucl(idx)= max(eucl); % se descarta el menor valor para hallar
                      % el segundo más cercano.

d02= min(eucl); % distancia con el 2do más cercano
if d01/d02 < dist_ratio
nn(k)= idx;   % se asigna la ubicación donde se encontró la
              % menor distancia.

dist(k)= d01; % relación de distancias
end
end
```

```
ipts= find(nn~=0); % desc. sin correspondencia: nn= 0
indexPairs=[nn(ipts),ipts];
distancia = dist(ipts); % relación de distancias

% eliminando posible duplicidad de puntos de correspondencia

        [m, n]=size(indexPairs)


%arreglando una columna(borrando duplicado de puntos)

       [c,ia,ic]=unique(indexPairs(:,2),'stable');
         index = indexPairs(ia,:);
         dist = distancia;
         ind_repetidos = setdiff(1:m, ia);
         valores_repetidos = indexPairs(ind_repetidos,2);
         valores_repetidos = unique(valores_repetidos);
         [x,y]=size(valores_repetidos);
         for i=1:1:x
            ind=find(index(:,2)~=valores_repetidos(i));
            index=index(ind,:);
            dist = dist(ind);
         end
         indexPairs=index;
         distancia=dist;
         [m, n]= size(indexPairs);

%arreglando la otra columna (borrando duplicado de puntos)

         [c,ia,ic]=unique(indexPairs(:,1),'stable');
         index = indexPairs(ia,:);
         dist = distancia;
         ind_repetidos = setdiff(1:m, ia);
         valores_repetidos = indexPairs(ind_repetidos,1);
         valores_repetidos = unique(valores_repetidos);
         [x,y]=size(valores_repetidos);
         for i=1:1:x
            ind=find(index(:,1)~=valores_repetidos(i));
            index=index(ind,:);
            dist = dist(ind);
         end
         indexPairs=index;
         distancia=dist;
end
```

3.3 Transformación de imagen

```
%Función que se encarga de la generación del mosaico a partir de
% la función de homografía. Además también devuelve la imagen
% transformada sin la generación del mosaico y la diferencia de
% intensidades de los pixeles superpuestos como medida de error.
% Entradas: im1= imagen referencia
%           im2= imagen superpuesta
%           homography= matriz de transformacion(3*3 projective)
% Salidas:  stitchedImage= mosaico (uint8)
%           imref = imagen superpuesta transformada (uint8)
%           diffpixel= vector que almacena la diferencia de
%                      intensidades de los pixeles superpuestos.

function [stitchedImage, imref, diffpixel]= stitch( im1, im2,...
homography)

stitchedImage = im1;

stitchedImage = padarray(stitchedImage,4*size(im2));

% Permite armar un mosaico de hasta el cuádruple del tamaño de
% la imagen a superponer

commompixel=1;
[M, N]=size(stitchedImage);
diffpixel=zeros(1,M*N);
imref=zeros(size(stitchedImage));

% Hallando vértices de la imagen para usarlos como referencia
% es una aproximación para reducir el número de iteraciones

vert_y=[0 size(im2,2)];
vert_x=[0 size(im2,1)];
puntos_i=zeros([1 4]);
puntos_j=zeros([1 4]);
h=homography;

for a=1:2
    for b=1:2

num1=h(1,1)*4*(size(im2, 2))+ h(1,2)*4*(size(im2, 1))-h(1,3)-...
vert_y(b)*h(3,1)*4*(size(im2, 2))-vert_y(b)*h(3,2)*...
(4*size(im2, 1))+h(3,3)*vert_y(b);

num2=h(2,1)*4*(size(im2, 2))+ h(2,2)*4*(size(im2, 1))-h(2,3)-...
vert_x(a)*h(3,1)*4*(size(im2, 2))-vert_x(a)*h(3,2)*...
(4*size(im2, 1))+h(3,3)*vert_x(a);

coef_1 = h(1,1)-vert_y(b)*h(3,1);
coef_2 = h(1,2)-vert_y(b)*h(3,2);
coef_3 = h(2,1)-vert_x(a)*h(3,1);
coef_4 = h(2,2)-vert_x(a)*h(3,2);

j = (coef_1*num2-coef_3*num1)/(coef_1*coef_4-coef_3*coef_2);
i = (num1-coef_2*j)/coef_1;
```

```
puntos_i(2*(a-1)+b)=i;
puntos_j(2*(a-1)+b)=j;
    end
end

% índices sobre los cuales iterar. La imagen transformada debe
% encontrarse entre esas coordenadas.

index_i_ini=floor(min(puntos_i));
index_j_ini=floor(min(puntos_j));
index_i_fin=round(max(puntos_i));
index_j_fin=round(max(puntos_j));


for i = index_i_ini:1:index_i_fin+1
    for j = index_j_ini:1:index_j_fin+1

    %resolución de la matriz de transformación

     p2 = homography * [i-4*size(im2, 2); j-4*size(im2, 1); 1];
     p2 = p2 / p2(3);

     x2 = round(p2(1));
     y2 = round(p2(2));


  if  x2>0 && x2<= size(im2, 2) && y2>0 && y2 <= size(im2, 1)


  imref(j,i)=im2(y2,x2);  % asignación de la imagen transformada


  valor_ini = stitchedImage(j, i);
  valor_next = imref(j, i);

  if  valor_next==0        % si el punto a superponer es negro %
                           % (superficie negra) se salta y se
                           % deja el que esta abajo(mosaico)

   stitchedImage(j, i) = valor_ini;

  else
   stitchedImage(j,i)= valor_next;
   end



if valor_ini> 0 && valor_next>0   % si ambos valores son mayores
                                  % que cero se asume traslape.

diffpixel(1,commompixel) = abs(double(valor_ini)-...
double(stitchedImage(j,i)));
commompixel=commompixel+1;
   end
```

```
        end

    end


end

diffpixel=diffpixel(1,1:commompixel-1);

%Limitar mosaico e imref al mismo tamaño. se limita a las mismas
%dimensiones para conservar los puntos de traslación en la
%imagen transformada.

[row,col] = find(stitchedImage>0);
a = min(col(:));
b = min(row(:));
c = max(col(:));
d = max(row(:));

st=imcrop(stitchedImage, [a b c-a d-b]);
st2=imcrop(imref, [a b c-a d-b]);

stitchedImage = st;
imref = uint8(st2);

end
```

3.4 Corrección de imágenes

```
% Adaptación de la implementación del algoritmo de rectificación geométrica diseñado
% y desarrollado por: Roberto Tupac Yupanqui Fernandez
% Datos de entrada: Imagen a rectificar (img) y ángulos de rotación de
% la cámara (yaw, pitch,roll)
%
% Datos de salida: Imagen geométricamente rectificada (img_correct)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function img_correct = imcorrect(img,yaw,roll,pitch)

% Dimensiones de la imagen fuente

width=size(img,2);
length=size(img,1);
%puntos de la imagen sin transfomar
C=[ 0 0;width 0;width length;0 length ];
D=[ 0 0;0 0;0 0;0 0];

% Definición de parámetros con respecto a los ejes

% Distancia aeroplano a la superficie
f=1333.333;
%Angulo del plano imaginario respecto a la superficie
ang=deg2rad(pitch);
```

```
%Hallando los puntos de la imagen transformada
%Realizando las proyecciones de los 4 puntos de la imagen
%Angulo eje Y
angy=deg2rad(roll);
%distancia del plano proyectado imaginario a la superficie
%en eje ZY
z=((width)/2)*sin(ang);
%distancia del plano proyectado imaginario a la superficie
%en eje ZX
z1=((length)/2)*sin(angy);
%angulo del vertice (1,1) cuando angy=0 en el plano Z-Y
angulo=atan(z/(length/2));
%angulo del vertice (1,1) cuando angy=0 en el plano Z-X
anguloy=atan(z1/(width/2));
%longitud del vertice(1,1) del rectangulo al punto central visto
desde el plano
%Z-Y
if ang==0
l=length/2;
angulo=0;
else
l=sqrt(z^2 +(length/2)^2 );
end
%longitud del vertice(1,1) del rectangulo al punto central visto
desde el plano
%Z-X
if angy==0
la=width/2;
else
la=sqrt(z1^2 +(width/2)^2 );
end
%longitud proyectada paralela al plano
%lon=((width)/2)*cos(ang);
teta1= atan((l*cos(angy+angulo))/(f-l*sin(angy+angulo)));
D(1,1)= tan(teta1)*f;
teta1a= atan((-1)*l*cos(pi-angulo+angy)/(f-l*sin(pi-
angulo+angy)));
D(4,2)= tan(teta1a)*f;
D(4,2)= (length/2) + D(4,2);%sin(pi-anguno+angy)
D(1,2)=(length/2)-D(1,1);
teta2= atan((l*cos(angy-angulo))/(f-l*sin(angy-angulo)));
D(2,2)= (f)*tan(teta2);
teta2a= atan(((-l)*cos(pi+angulo+angy))/(f-
l*sin(pi+angulo+angy)));
D(3,2)=(length/2) + (f)*tan(teta2a);
D(2,2)=(length/2)-D(2,2);
teta3= atan((la*cos(ang+anguloy))/(f-la*sin(ang+anguloy)));
D(1,1)=(width/2)-f*tan(teta3);
%-----------------------------------------------------
%
%procedimiento para hallar la coordenada x en el punto 1
%primero en el plano ZY obtenemos los lados del triangulo
lc1=l*sin(angy+angulo)-(length/2)*sin(angy);
lc2=abs(l*cos(angy+angulo))-abs((length/2)*cos(angy));
lc3=sqrt(lc1^2 +lc2^2);
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%formas otro triangulo
lc4=sqrt((width/2)^2 -lc3^2);
%%%%%%%%%%%%%%%%%%%%
%por ultimo formamos un triangulo en el plano XY
% hacemos la proyeccion al final en el plano XZ
tet11= atan((lc4)/(f-l*sin(angy+angulo)));
D(1,1)=(width/2)-f*tan(tet11);
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------
%
%procedimiento para hallar la coordenada x en el punto 4
%primero en el plano ZY obtenemos los lados del triangulo
lb1=l*sin(pi-angulo+angy)-(length/2)*sin(pi+angy);
lb2=abs(l*cos(pi-angulo+angy))-abs((length/2)*cos(pi+angy));
lb3=sqrt(lb1^2 +lb2^2);
%%%%%%%%%%%%%%%%%%%%%%%%
%formas otro triangulo
lb4=sqrt((width/2)^2 -lb3^2);
%%%%%%%%%%%%%%%%%%%%
%por ultimo formamos un triangulo en el plano XY
% hacemos la proyeccion al final en el plano XZ
tet4aa= atan((lb4)/(f-l*sin(pi-angulo+angy)));
D(4,1)=(width/2)-f*tan(tet4aa);
%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------
%
%procedimiento para hallar la coordenada x en el punto 2
%primero en el plano ZY obtenemos los lados del triangulo
la1=z1-l*sin(angy-angulo);
la2=l*cos(angy-angulo)-(length/2)*cos(angy);
la3=sqrt(la1^2 +la2^2);
%%%%%%%%%%%%%%%%%%%%%%%%
%formas otro triangulo
la4=sqrt((width/2)^2 -la3^2);
%%%%%%%%%%%%%%%%%%%%
%por ultimo formamos un triangulo en el plano XY
% hacemos la proyeccion al final en el plano XZ
tet4= atan((la4)/(f-l*sin(angy-angulo)));
D(2,1)=(width/2)+f*tan(tet4);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%-----------------------------------------------------------
%
%procedimiento para hallar la coordenada x en el punto 3
%primero en el plano ZY obtenemos los lados del triangulo
l1= l*sin(angulo+angy)-z1;
l2=(length/2)*cos(angy)-l*cos(angulo+angy);
l3=sqrt(l1^2 +l2^2);
%%%%%%%%%%%%%%%%%%%%%%%%
%formas otro triangulo
l4=sqrt((width/2)^2 -l3^2);
%%%%%%%%%%%%%%%%%%%%
%por ultimo formamos un triangulo en el plano XY
% hacemos la proyeccion al final en el plano XZ
tet31= atan((l4)/(f-l*sin(pi+angulo+angy)));
D(3,1)=(width/2)+f*tan(tet31);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
T = maketform('projective',D,C);

J = imtransform(img,T,'nearest',...
'XData', [1 (size(img,2)+0)],...
'YData', [1 (size(img,1)+0)],...
'FillValues', 0);

%%%%%%%%%%%%%%%%%%%%%%%%%%5
%Rotacion
% Rotation: jaw
img_correct=imrotate(J,yaw);
[row,col] = find(img_correct>0);
a = min(col(:));
b = min(row(:));
c = max(col(:));
d = max(row(:));
img_correct=imcrop(img_correct, [a b c-a d-b]);

end
```

Anexo 4: Programa en C

```
// Relaciona multiples imágenes con una área en común.
// El parámetro de entrada es la carpeta donde se ubican las
// las imágenes que generarían el mosaico. No se logró realizar
// la corrección geométrica conjuntamente con la generación de
// mosaico. El modo por defecto es el de usar la imagen anterior
// como referencia

#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <opencv2/opencv.hpp>
#include <opencv2/nonfree/nonfree.hpp>
#include <opencv2/legacy/legacy.hpp>
#include <opencv2/nonfree/features2d.hpp>
#include <opencv2/features2d/features2d.hpp>
#include <dirent.h>

using namespace cv;
using std::cout;
using std::string;

// Función que realizará la unión  de las imágenes  devuelve
// también el error promedio por pixel.

void stitch(Mat& img1,Mat& img2,const Mat& Homography, Mat&...
StitchedImage , Mat& Imref)

{

// se define un tamaño mayor al original (5 veces más grande)
// para ubicar la el mosaico generado.

 Size s=img2.size();
 double  row2=s.height;
 double  col2=s.width;
 int top=round(row2*5);
 int bot=round(row2*5);
 int left=round(col2*5);
 int right=round(col2*5);

copyMakeBorder(img1,StitchedImage,top,bot,left,right,...
BORDER_CONSTANT,Scalar(0));

 s=StitchedImage.size();
 int row=s.height;
 int col=s.width;

Imref = Mat::zeros(row,col,CV_8U);

double num_pixel=0;
int jmin=row2*5;
```

```
int imin=col2*5;
      int jmax=row-row2*5;
      int imax=col-col2*5;
      double error;
      double suma_error=0;
      double promedio;

// Este procedimiento es una estimación para obtener los
// vértices de la imagen transformada para evitar repeticiones
// innecesarias.

Mat vert_y = (Mat_<double>(2,1) << -5, col2+5);
Mat vert_x = (Mat_<double>(2,1) << -5, row2+5);
Mat puntos_i = (Mat_<double>(4,1) << 0,0,0,0);
Mat puntos_j =(Mat_<double>(4,1) << 0,0,0,0);

Mat h=Homography;

for (double a = 0; a < 2; a++ )
    {
     for (double b = 0; b < 2; b++ )
     {
     double num1 = h.at<double>(0,0)*col2*5+...
     h.at<double>(0,1)*row2*5 - h.at<double>(0,2) -...
     vert_y.at<double>(b,0)*h.at<double>(2,0)*col2*5 -...
     vert_y.at<double>(b,0)*h.at<double>(2,1)*row2*5 +...
     h.at<double>(2,2)*vert_y.at<double>(b,0);

     double num2 = h.at<double>(1,0)*col2*5 +...
     h.at<double>(1,1)*row2*5 - h.at<double>(1,2)-...
     vert_x.at<double>(a,0)*h.at<double>(2,0)*col2*5 -...
     vert_x.at<double>(a,0)*h.at<double>(2,1)*row2*5 +...
     h.at<double>(2,2)*vert_x.at<double>(a,0);

     double coef_1 = h.at<double>(0,0)-...
     vert_y.at<double>(b,0)*h.at<double>(2,0);
     double coef_2 = h.at<double>(0,1)-...
     vert_y.at<double>(b,0)*h.at<double>(2,1);
     double coef_3 = h.at<double>(1,0)-...
     vert_x.at<double>(a,0)*h.at<double>(2,0);
     double coef_4 = h.at<double>(1,1)-...
     vert_x.at<double>(a,0)*h.at<double>(2,1);

double j = round((coef_1*num2-coef_3*num1)/(coef_1*coef_4-...
coef_3*coef_2));
double i = round((num1-coef_2*j)/coef_1);
puntos_i.at<double>(2*(a)+b,0)=i;
puntos_j.at<double>(2*(a)+b,0)=j;
          }
    }
          double index_y_ini, index_y_max;
          cv::minMaxLoc(puntos_i, &index_y_ini, &index_y_max);
          double index_x_ini, index_x_max;
          cv::minMaxLoc(puntos_j, &index_x_ini, &index_x_max);
```

```
for (double i = index_y_ini; i <= index_y_max; i++ )
    {
   for (double j = index_x_ini; j <= index_x_max; j++ )
     {
// %resolución de la matriz de transformación

Mat p = (Mat_<double>(3,1) <<i-round(col2*5),j-round(row2*5),1);
      Mat q=Homography*p;
      q=q/q.at<double>(2,0);

     double xt= round(q.at<double>(0,0));
     double yt= round(q.at<double>(1,0));


 if (xt>=0 && xt < col2 && yt>=0 && yt<row2)
      {

     Imref.at<uchar>(j,i)=img2.at<uchar>(yt,xt);
     double  valor_ini = StitchedImage.at<uchar>(j,i);
     double  valor_next = Imref.at<uchar>(j,i);

     if (valor_next == 0) //si lo que se superpone es negro no
                          // interesa

     {StitchedImage.at<uchar>(j,i) = valor_ini;}
     else
     {StitchedImage.at<uchar>(j,i) = valor_next;}

     if (valor_ini> 0 && valor_next>0 )
      {
      num_pixel=num_pixel+1; // si hay "color" en ambas imágenes
                             // están superpuestas

error = double (abs(StitchedImage.at<uchar>(j,i)-valor_ini));
          suma_error = suma_error+error;


// Permite obtener las coordenadas que ocupa la imagen
     if (StitchedImage.at<uchar>(j,i)>0 )
     {
      if (i<imin)
     { imin=i; }

     if (j<jmin)      //define el inicio de la imagen cuando la
                      // imagen referencia este superior

     { jmin=j; }

      if (j>jmax)     //define el fin de la imagen
     {jmax=j;}

     if (i>imax)
     {imax=i;}
     }
   }
}
```

```
//Limitación  de la imagen (mosaico y la nueva referencia)

      Rect myROI(imin,jmin,imax-imin,jmax-jmin);
      StitchedImage=StitchedImage(myROI);
      Imref=Imref(myROI);
      promedio = suma_error/num_pixel;
      printf("-- Num_pixeles_totales");


printf("Num_pixeles_totales=%f;\n promedio_error = %f\n ",...
num_pixel, ,promedio);
           }


int main( )
{
      clock_t startC, finishC;

      startC = clock();


cv::initModule_nonfree();  // declaración necesaria para
                           // habilitar el surf detector

// Inicializar directorio de donde se leerán las imágenes

char imageDirectory[] = "Images/vuelo3/00000";
char fullImagePath[1000];
Mat image_array[1000];
char data_type[] = ".bmp";
int total_img=200; //total de imagenes

// Lectura de las imágenes que tendrán el formato 00001.bmp; es
// el número de la imagen acompañada de ceros al inicio.

for  (int i=0;i<=total_img;i++)
 {
      if (i<10)
      {
           image_array[i-25]=imread(fullImagePath,...
CV_LOAD_IMAGE_GRAYSCALE);
           }
           if (i>9 && i<100)
           {
           char imageDirectory[] = "Images/vuelo3/000";
          image_array[i-25]=imread(fullImagePath,...
CV_LOAD_IMAGE_GRAYSCALE);
           }
           if(i>100 && i <1000)
           {
           char imageDirectory[] = "Images/vuelo3/00";
           image_array[i-25]=imread(fullImagePath,...
CV_LOAD_IMAGE_GRAYSCALE);
           }
      }
```

```
//Armado del mosaico

Mat mosaico = image_array[0]; //La primera imagen es la
                              //referencia de todo el mosaico.
Mat imref = image_array[0];

for (int k=1;k<total_img;k++)
{
     Mat new_image = image_array[k];

          if(imref.empty() || new_image.empty())
               {
                  printf("No se puede leer alguna imagen\n");
                   continue;
               }

// Detección de puntos de control.

SurfFeatureDetector detector(1000);
std::vector<KeyPoint> keypoints_reference, keypoints_moving;
detector.detect(imref, keypoints_reference);
detector.detect(new_image, keypoints_moving);

// Extraccion de descriptores

SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute(imref, keypoints_reference, descriptors1);
extractor.compute(new_image, keypoints_moving, descriptors2);

// Analisis de correspondencia NNDR = 0.8

FlannBasedMatcher matcher;
double ratio = 0.8;
std::vector< vector<DMatch > > nnMatches;
std::vector< DMatch > good_matches;
matcher.knnMatch(descriptors1, descriptors2, nnMatches, 2 );

for(int k = 0; k < nnMatches.size(); k++)
   {
if(nnMatches[k][0].distance / nnMatches[k][1].distance < ratio)
     {
     good_matches.push_back(nnMatches[k][0]);
     }
   }

if (good_matches.size()<5)
{ printf("No hay relacion entre las imagenes\n");
  continue;
}

// Asignacion de los puntos correspondientes hallados. Obj=
//puntos en referencia, scene =puntos en imagen a superponer.

std::vector< Point2f > obj;
std::vector< Point2f > scene;
```

```
for( int i = 0; i < good_matches.size(); i++ )
  {

obj.push_back( keypoints_reference[ good_matches[i].queryIdx...
].pt);
scene.push_back( keypoints_moving[ good_matches[i].trainIdx...
].pt);
   }

// Estimacion de la funcion de transformacion

Mat bestHomography = findHomography( obj, scene, CV_RANSAC );

// Transformación de la imagen y generación del mosaico.

 Mat StitchedImage;
 Mat im_ref;

stitch(mosaico,new_image,bestHomography,StitchedImage,im_ref);
imref=im_ref;
   }

// Mostrar resultados tiempo y mosaico generado.

  namedWindow("mosaico", 1);
  imshow("mosaico", mosaico);
  finishC = clock();

cout << "Time (clock): "    << (finishC - ...
startC)*1.0/CLOCKS_PER_SEC << std::endl;
        waitKey(0);
        return 0;
}
```