PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERIA

RECTIFICACIÓN GEOMÉTRICA DE IMÁGENES MEDIANTE EL EMPLEO DE TRANSFORMACIONES PROYECTIVAS Y UN SISTEMA DE MEDICIÓN INERCIAL

ANEXOS

ÍNDICE

ANEXO 1: Parámetros intrínsecos y extrínsecos de una cámara e implementación de los movimientos de alabeo, cabeceo, y guiñada

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%%% Fuente:<http://www.cs.toronto.edu/~jepson/csc2503/index01.html>
%%% Archivo: warpDemo.m
%%% Consultado: 05/05/2013
%%% Implementación de los parámetros extrínseco e intrínsecos de una
%%% cámara, y cambios de orientación en la camara
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%

%% Parámetros intrínsecos


% punto principal
o = [320 240];
f = 1.65;  % Distancia focal

% Matriz de parámetros intrínsecos
M_im = [f 0 0;
       0 f 0;
       0 0 1];

M_pix = [320  0     o(1);
     0     320    o(2);
     0     0      1];

M_int = M_pix;

%% ----------------------------------------------
%% Parámetros extrínsecos

%%Se definen los angulos de orientacion inicial
%%apuntando a la camara en la matriz de rotación R

thetaYZ = 0;
ct = cos(thetaYZ);
st = sin(thetaYZ);
R1 = [1 0 0; 0 ct -st; 0 st ct];

thetaXZ = pi;
ct = cos(thetaXZ);
st = sin(thetaXZ);
R2 = [ct 0 -st; 0 1 0; st 0 ct];

thetaXY = 0;
ct = cos(thetaXY);
```

```
st = sin(thetaXY);
R3 = [ct -st 0; st ct 0; 0 0 1];
R = R1*R2*R3;

%% La matriz T define la traslación del sistema de coordenadas de la    %% cámara
respecto al sistema de coordenadas universal
T = [320 240 320]';
D = -R*T;

M_cam = [R D];


% ----------------------------------------------
% De las matriz de los parámetros intrínsecos y extrínsecos de obtiene
% la matriz fundamental H

H = M_int*M_cam

% Por redundancia se elimina la 3ra columna de la matriz H
F = H(:,[1,2,4]);
A = pinv(F); % versión estable de la inversión de F

% dimensiones  de la imagen
szIm = [480 640];
% relicacion de los pixeles
[x,y] = meshgrid(1:szIm(2),1:szIm(1));
pix   = [x(:)'; y(:)'];

hPixels = [ pix; ones(1,szIm(1)*szIm(2)) ];

% Correspondencia entre puntos respecto ala estructura de la imagen %inicial
Scene  = A*hPixels;

% coordenadas de los pixeles en la imagen inicial
xprime=(hScene(1,:)./(hScene(3,:)))';
yprime=(hScene(2,:)./(hScene(3,:)))';
xprime = reshape(xprime, szIm);
yprime = reshape(yprime, szIm);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
% Obtención de la imagen vista por la cámara, empleando la matriz de     %
homografía antes descrita
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%

% Lectura de la imagen
im = imread('f9.bmp');
im=rgb2gray(im);
im = im2double(im);
im = fliplr(im);
```

```
% Proceso de interpolación
result = interp2(x,y,im,xprime,yprime, '*linear');
skyGray = 0.8 * 256;
result(hScene(3,:) < 0) = skyGray;
result(isnan(result)) = 0;
result = reshape(result,szIm);

figure(1); clf; imshow(result);
pause;


%%% Simulación de movimiento de alabeo de la cámara respecto a la imagen %%%
de referencia

for thetaYZ = 0.2*pi-0.2:-0.2:-0.3*pi

  ct = cos(thetaYZ);
  st = sin(thetaYZ);
  R1 = [1 0 0; 0 ct -st; 0 st ct];

  % The rest is the same as above...
  R = R1*R2*R3;
  T = [320 240 320]';
  D = -R*T;
  M_cam = [R D];
  H = M_int*M_cam;

  F = H(:,[1,2,4]);
  A = pinv(F); % stable version of inverse

  % pixel coordinates
  [x,y] = meshgrid(1:szIm(2),1:szIm(1));
  pix   = [x(:)'; y(:)'];

  % homogeneous pixels
  hPixels = [ pix; ones(1,szIm(1)*szIm(2)) ];
  % corresponding warped points
  hScene  = A*hPixels;

  xprime=(hScene(1,:)./(hScene(3,:)))';
  yprime=(hScene(2,:)./(hScene(3,:)))';

  xprime = reshape(xprime, szIm);
  yprime = reshape(yprime, szIm);

  result = interp2(x,y,im,xprime,yprime, '*linear');
  skyGray = 0.8 * 256;
  result(hScene(3,:) < 0) = skyGray;
  result(isnan(result)) = 0;
  result = reshape(result,szIm);
```

```
  figure(1); clf; imshow(result);
  pause;

end  % End of loop over camera rotation
thetaYZ = 0;
ct = cos(thetaYZ);
st = sin(thetaYZ);
R1 = [1 0 0; 0 ct -st; 0 st ct];

%%% Simulación de movimiento de cabeceo de la cámara respecto a la     %%%
imagen de referencia


for thetaXZ = pi*0.6:0.41:pi*1.4
thetaXZ
    % Almost looking backwards along W_3, or Z, axis
ct = cos(thetaXZ);
st = sin(thetaXZ);
R2 = [ct 0 -st; 0 1 0; st 0 ct];



  % The rest is the same as above...
  R = R1*R2*R3;
  T = [320 240 320]';
  D = -R*T;
  M_cam = [R D];
  H = M_int*M_cam;

  F = H(:,[1,2,4]);
  A = pinv(F); % stable version of inverse

  % pixel coordinates
  [x,y] = meshgrid(1:szIm(2),1:szIm(1));
  pix   = [x(:)'; y(:)'];

  % homogeneous pixels
  hPixels = [ pix; ones(1,szIm(1)*szIm(2)) ];
  % corresponding warped points
  hScene  = A*hPixels;

  xprime=(hScene(1,:)./(hScene(3,:)))';
  yprime=(hScene(2,:)./(hScene(3,:)))';

  xprime = reshape(xprime, szIm);
  yprime = reshape(yprime, szIm);

  result = interp2(x,y,im,xprime,yprime, '*linear');
  skyGray = 0.8 * 256;
  result(hScene(3,:) < 0) = skyGray;
```

```
  result(isnan(result)) = 0;
  result = reshape(result,szIm);

  figure(1); clf; imshow(result);
  pause;

end  % End of loop over camera rotation
thetaXZ = pi;  % Almost looking backwards along W_3, or Z, axis
ct = cos(thetaXZ);
st = sin(thetaXZ);
R2 = [ct 0 -st; 0 1 0; st 0 ct];

%% Simulación de movimiento de guiñada de la cámara respecto a la imagen
%% de referencia
for thetaXY = pi*0.6:0.41:pi*1.4
thetaXZ
    % Almost looking backwards along W_3, or Z, axis

ct = cos(thetaXY);
st = sin(thetaXY);
R3 = [ct -st 0; st ct 0; 0 0 1]



  % The rest is the same as above...
  R = R1*R2*R3;
  T = [320 240 320]';
  D = -R*T;
  M_cam = [R D];
  H = M_int*M_cam;

  F = H(:,[1,2,4]);
  A = pinv(F); % stable version of inverse

  % pixel coordinates
  [x,y] = meshgrid(1:szIm(2),1:szIm(1));
  pix   = [x(:)'; y(:)'];

  % homogeneous pixels
  hPixels = [ pix; ones(1,szIm(1)*szIm(2)) ];
  % corresponding warped points
  hScene  = A*hPixels;

  xprime=(hScene(1,:)./(hScene(3,:)))';
  yprime=(hScene(2,:)./(hScene(3,:)))';

  xprime = reshape(xprime, szIm);
  yprime = reshape(yprime, szIm);
```

```
  result = interp2(x,y,im,xprime,yprime, '*linear');
  skyGray = 0.8 * 256;
  result(hScene(3,:) < 0) = skyGray;
  result(isnan(result)) = 0;
  result = reshape(result,szIm);

  figure(1); clf; imshow(result);
  pause;

end
```

ANEXO 2: Código del algoritmo de rectificación geométrica para Matlab

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
%%%
%%% Implementación del algoritmo de rectificación geométrica diseñado
%%% Desarrollado por: Roberto Tupac Yupanqui Fernandez
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
%%% Datos de entrada: Imagen a rectificar (I) y ángulos de rotación de
%%% la cámara(yaw, pitch,roll)
%%%
%%% Datos de salida: Imagen geométricamente rectificada(J)
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%

%Leemos la imagen
I = imread('f44.bmp');
K=imread('f22.bmp');
I = imresize(I, 1);
%escalamos la imagen pues es muy grande de tamaño
I = imresize(I,1);
V=I;
%Definimos los angulos de rotacion yaw,pitch y roll en grados sexagesimales
yaw =-2;%0
pitch=6;%27 f21
roll=28;%18

%dimensiones de la imagen fuente
width=size(I,2);
length=size(I,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];

%Angulo del aeroplano en grados
angu=deg2rad(90);
%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
anguno=atan(z/(length/2));

%angulo del vertice (1,1) cuando angy=0 en el plano Z-X
angunoy=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;
   anguno=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+anguno))/(f-l*sin(angy+anguno)));
D(1,1)= tan(teta1)*f;
teta1a= atan((-1)*l*cos(pi-anguno+angy)/(f-l*sin(pi-anguno+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-anguno))/(f-l*sin(angy-anguno)));
D(2,2)= (f)*tan(teta2);
teta2a= atan(((-l)*cos(pi+anguno+angy))/(f-l*sin(pi+anguno+angy)));

D(3,2)=(length/2) + (f)*tan(teta2a);

D(2,2)=(length/2)-D(2,2);
```

```
teta3= atan((la*cos(ang+angunoy))/(f-la*sin(ang+angunoy)));
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+anguno)-(length/2)*sin(angy);
lc2=abs(l*cos(angy+anguno))-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+anguno)));
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-anguno+angy)-(length/2)*sin(pi+angy);
lb2=abs(l*cos(pi-anguno+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-anguno+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-anguno);
la2=l*cos(angy-anguno)-(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-anguno)));
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(anguno+angy)-z1;
l2=(length/2)*cos(angy)-l*cos(anguno+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+anguno+angy)));
D(3,1)=(width/2)+f*tan(tet31);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%


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

J = imtransform(I,T,...
                'XData', [1 (size(I,2)+0)],...
                'YData', [1 (size(I,1)+0)],...
                'FillValues', [0;0;255]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
%Rotacion
% Rotation: theta
theta = deg2rad(yaw);

T_r = [ cos(theta)  -sin(theta)  0; ...
        sin(theta)   cos(theta)  0; ...
            0            0      1];

tform = maketform('affine',T_r);


J = imtransform(J,tform,...
                'XData', [1 (size(I,2)+0)],...
                'YData', [1 (size(I,1)+0)],...
                'FillValues', [0;0;255]);

imwrite(J, 'prueba88.bmp', 'bmp');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 imshow(V), figure,imshow(J),figure,imshow(K)
```

ANEXO 3: Código algoritmo de rectificación geométrica en OPENCV

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%%
%%% Implementación del algoritmo de rectificación geométrica diseñado
%%% Desarrollado por: Roberto Tupac Yupanqui Fernandez
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%
%%% Datos de entrada: Imagen a rectificar (imag) y ángulos de rotación
%%% de la cámara(yaw, pitch,roll)
%%%
%%% Datos de salida: Imagen geométricamente rectificada(J)
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%

#include <opencv\cv.h>
#include<opencv\cxcore.h>
#include <opencv\highgui.h>
#include <iostream>
#include <cmath>
#define PI 3.14159265358979323846264338327950

using namespace std;
using namespace cv;

int main()
{
	float roll,pitch,yaw,
focal=1333.333,z,z1,anguno,angunoy,l,la,teta1,teta1a,angy,ang,teta2,teta2a,teta3;
	float lc1,lc2,lc3,lc4,tet11,lb1,lb2,lb3,lb4,tet4aa,la1,la2,la3,la4,tet4,l1,l2,l3,l4,tet31;
	int h;
	//Obtencion de la imagen
	Mat imag = imread("f44.bmp");
	//Obtencion de los angulos de rotacion
	roll=28;
	yaw=-2;
	pitch=6;
	//Normalizacion de los grados a radianes
	angy = (roll * PI)/180;
	ang = (pitch * PI)/180;
	//distancia del plano proyectado imaginario a la superficie
	//en eje ZY
	z=((imag.cols)/2)*sin(ang);
	cout << z << endl;

	//distancia del plano proyectado imaginario a la superficie
	//en eje ZX
	z1=((imag.rows)/2)*sin(angy);
```

```
        //angulo del vertice (1,1) cuando angy=0 en el plano Z-Y
        anguno=atan(z/(imag.rows/2));

        //angulo del vertice (1,1) cuando angy=0 en el plano Z-X
        angunoy=atan(z1/(imag.cols/2));

        //longitud del vertice(1,1) del rectangulo al punto central visto desde el plano
        //Z-Y
        if (pitch==0)
                {l=imag.rows/2;
                anguno=0;}
        else
                {
                l=sqrt(z*z +(imag.rows/2)*(imag.rows/2));}

        //longitud del vertice(1,1) del rectangulo al punto central visto desde el plano
        //Z-X

        if (roll==0)
                {la=imag.cols/2;
                }
        else
                {
                la=sqrt(z1*z1 +(imag.cols/2)*(imag.cols/2));}

        cout << l << endl;
        cout << imag.rows << endl;
        cout << imag.cols << endl;
        cout << roll << endl;
        //Definicion de matriz C
        vector<Point2f> matriz_c;
matriz_c.push_back(Point(0, 0));
matriz_c.push_back(Point(imag.cols, 0));
matriz_c.push_back(Point(imag.cols, imag.rows));
matriz_c.push_back(Point(0, imag.rows));
        //Definicion de matriz D
        vector<Point2f> matriz_d;
matriz_d.push_back(Point(0, 0));
matriz_d.push_back(Point(0, 0));
matriz_d.push_back(Point(0, 0));
matriz_d.push_back(Point(0, 0));

        teta1= atan((l*cos(angy+anguno))/(focal-l*sin(angy+anguno)));
        matriz_d[0].x= tan(teta1)*focal;
        teta1a= atan((-1)*l*cos(PI-anguno+angy)/(focal-l*sin(PI-anguno+angy)));
        matriz_d[3].y= tan(teta1a)*focal;
        matriz_d[3].y= (imag.rows/2) + matriz_d[3].y;//sin(pi-anguno+angy)
        matriz_d[0].y=(imag.rows/2)-matriz_d[0].x;

        teta2= atan((l*cos(angy-anguno))/(focal-l*sin(angy-anguno)));
```

```
matriz_d[1].y= (focal)*tan(teta2);
teta2a= atan(((-l)*cos(PI+anguno+angy))/(focal-l*sin(PI+anguno+angy)));

matriz_d[2].y=(imag.rows/2) + (focal)*tan(teta2a);

matriz_d[1].y=(imag.rows/2)-matriz_d[1].y;

teta3= atan((la*cos(ang+angunoy))/(focal-la*sin(ang+angunoy)));
matriz_d[0].x=(imag.cols/2)-focal*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+anguno)-(imag.rows/2)*sin(angy);
lc2=abs(l*cos(angy+anguno))-abs((imag.rows/2)*cos(angy));
lc3=sqrt(lc1*lc1 +lc2*lc2);
//
//formas otro triangulo
lc4=sqrt((imag.cols/2)*(imag.cols/2) -(lc3*lc3));
//
//por ultimo formamos un triangulo en el plano XY
// hacemos la proyeccion al final en el plano XZ
tet11= atan((lc4)/(focal-l*sin(angy+anguno)));
matriz_d[0].x=(imag.cols/2)-focal*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-anguno+angy)-(imag.rows/2)*sin(PI+angy);
lb2=abs(l*cos(PI-anguno+angy))-abs((imag.rows/2)*cos(PI+angy));
lb3=sqrt(lb1*lb1 +lb2*lb2);
//
//formas otro triangulo
lb4=sqrt((imag.cols/2)*imag.cols/2 -lb3*lb3);
//
//por ultimo formamos un triangulo en el plano XY
// hacemos la proyeccion al final en el plano XZ
tet4aa= atan((lb4)/(focal-l*sin(PI-anguno+angy)));
matriz_d[3].x=(imag.cols/2)-focal*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-anguno);
la2=l*cos(angy-anguno)-(imag.rows/2)*cos(angy);
```

```
        la3=sqrt(la1*la1 +la2*la2);
        //
        //        formas otro triangulo
        la4=sqrt((imag.cols/2)*(imag.cols/2) -la3*la3);
        //
        //%por ultimo formamos un triangulo en el plano XY
        //% hacemos la proyeccion al final en el plano XZ
        tet4= atan((la4)/(focal-l*sin(angy-anguno)));
        matriz_d[1].x=(imag.cols/2)+focal*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(anguno+angy)-z1;
        l2=(imag.rows/2)*cos(angy)-l*cos(anguno+angy);
        l3=sqrt(l1*l1 +l2*l2);
        //
        //%formas otro triangulo
        l4=sqrt((imag.cols/2)*(imag.cols/2) -l3*l3);
        //
        //%por ultimo formamos un triangulo en el plano XY
        //% hacemos la proyeccion al final en el plano XZ
        tet31= atan((l4)/(focal-l*sin(PI+anguno+angy)));
        matriz_d[2].x=(imag.cols/2)+focal*tan(tet31);

        Mat warpPro = getPerspectiveTransform(matriz_d, matriz_c);
        Mat final;

        warpPerspective(imag,final,warpPro,(imag.size()));
        //
        cout << imag.size() << endl;
        cout << matriz_d[0] << endl;
        cout << matriz_d[1] << endl;
        cout << matriz_d[2] << endl;
        cout << warpPro << endl;

        Point2f src_center(imag.cols/2.0F, imag.rows/2.0F);
    Mat rot_mat = getRotationMatrix2D(src_center, yaw, 1.0);
        Mat dst;
    warpAffine(final, dst, rot_mat, final.size());

        cout << yaw << endl;
        namedWindow( "Display window");// Create a window for display.
        imshow( "Display window", imag );
        namedWindow( "Display");// Create a window for display.
        imshow( "Display", final );
        namedWindow( "Rectificacion");// Create a window for display.
        imshow( "Rectificacion", dst);
        waitKey(0);
return 0;}
```

ANEXO 4: Banco de imágenes

En este anexo se presentan el banco de imágenes empleado en el capítulo 4, con los cuales se evalúa la eficiencia del algoritmo de rectificación diseñado. Cada una de las imágenes se presentan rectificadas de acuerdo a los ángulos de rotación y los datos de media aritmética y desviación estándar obtenidos de la relación con similitud con la imagen de referencia.

Imagen de referencia

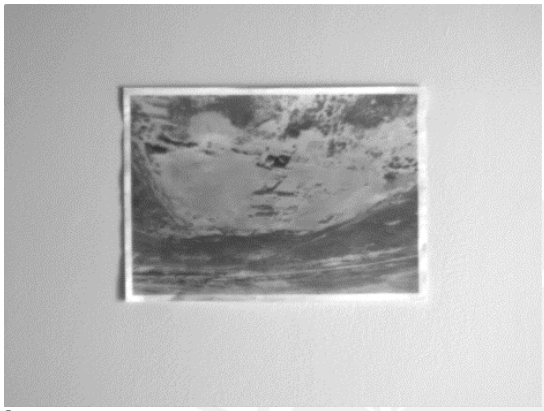
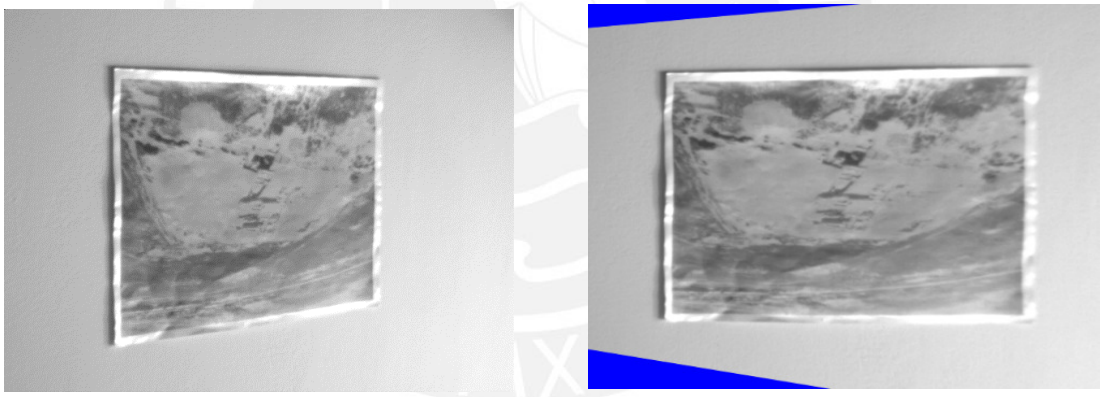
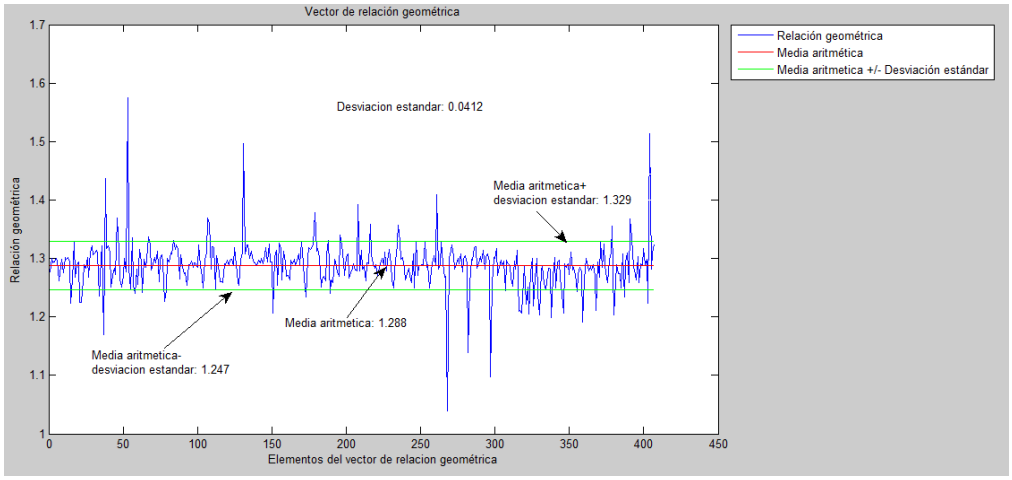


Imagen 1

Ángulos de rotación: yaw=0 pitch=44, roll=3



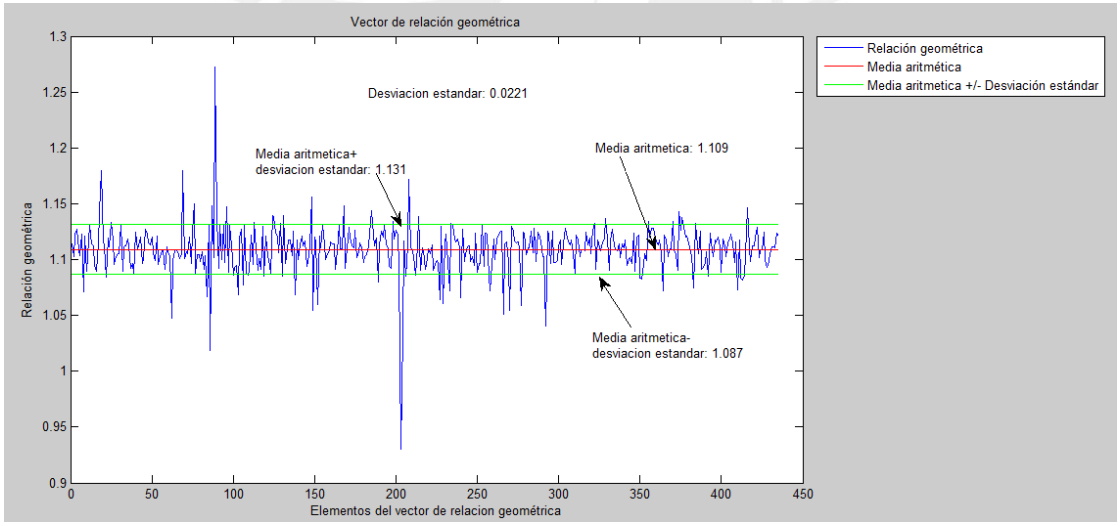


Desviación estándar: 0.0412

Media aritmética: 1.288

Imagen 2

Ángulos de rotación: yaw=26 pitch=17, roll=0

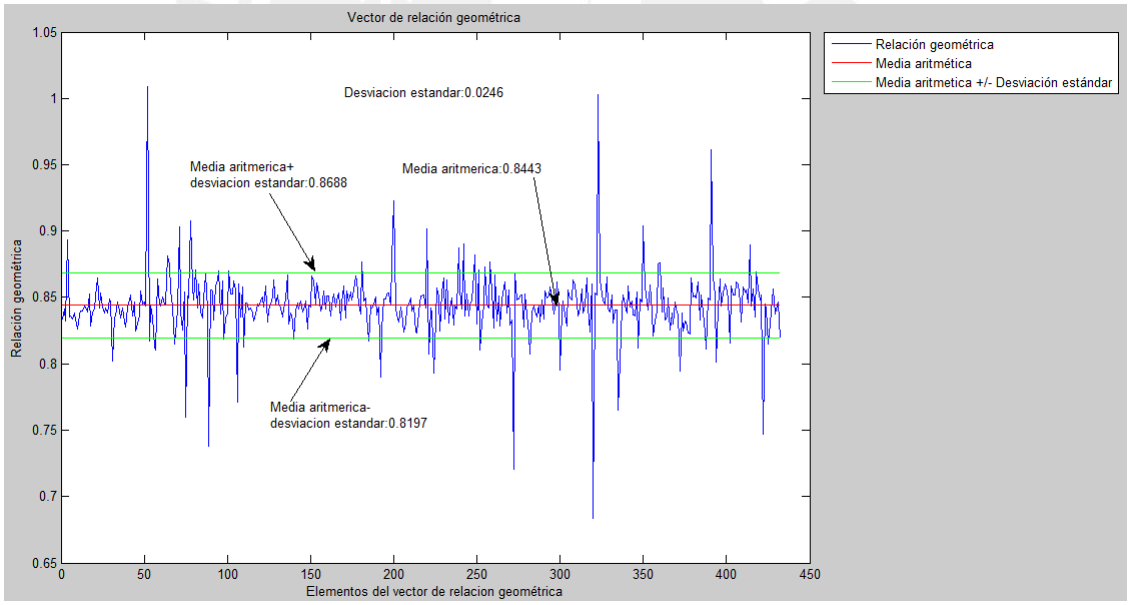


Desviación estándar: 0.0221

Media aritmética: 1.109

Imagen 3

Ángulos de rotación: yaw=0 pitch=16, roll=0

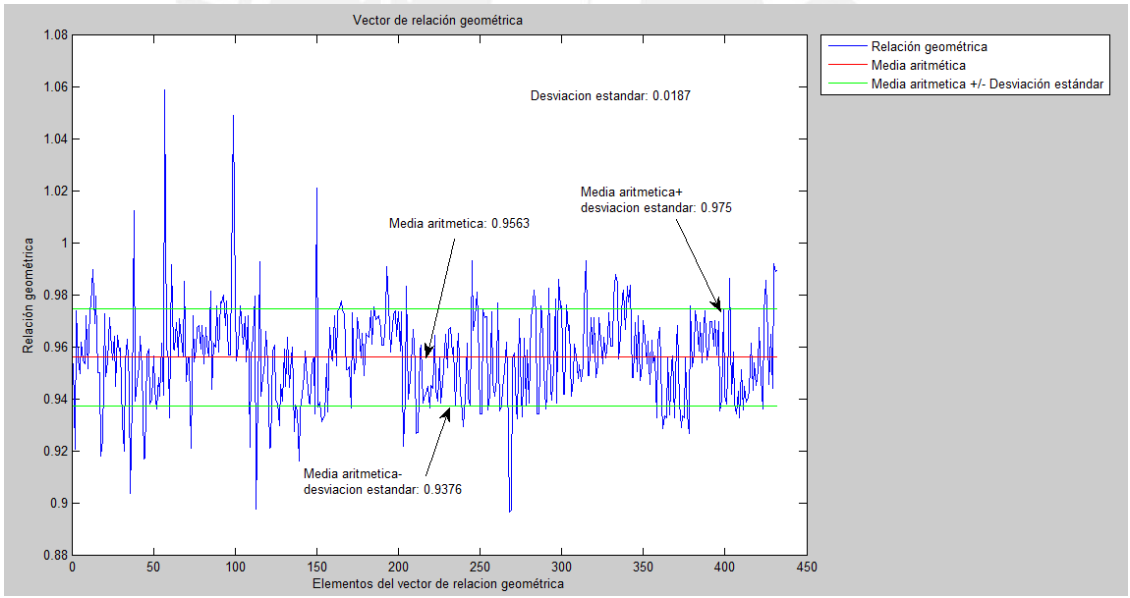


Desviación estándar: 0.0246

Media aritmética: 0.8443

Imagen 4

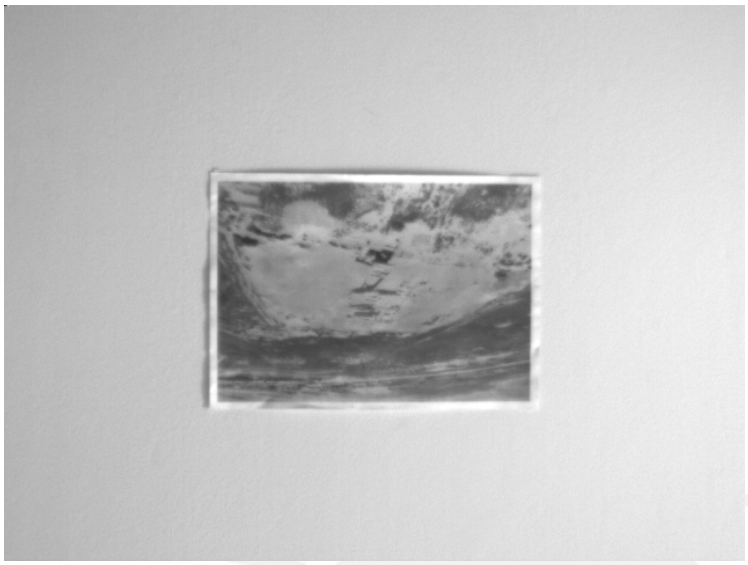
Ángulos de rotación: yaw=2 pitch=17, roll=4

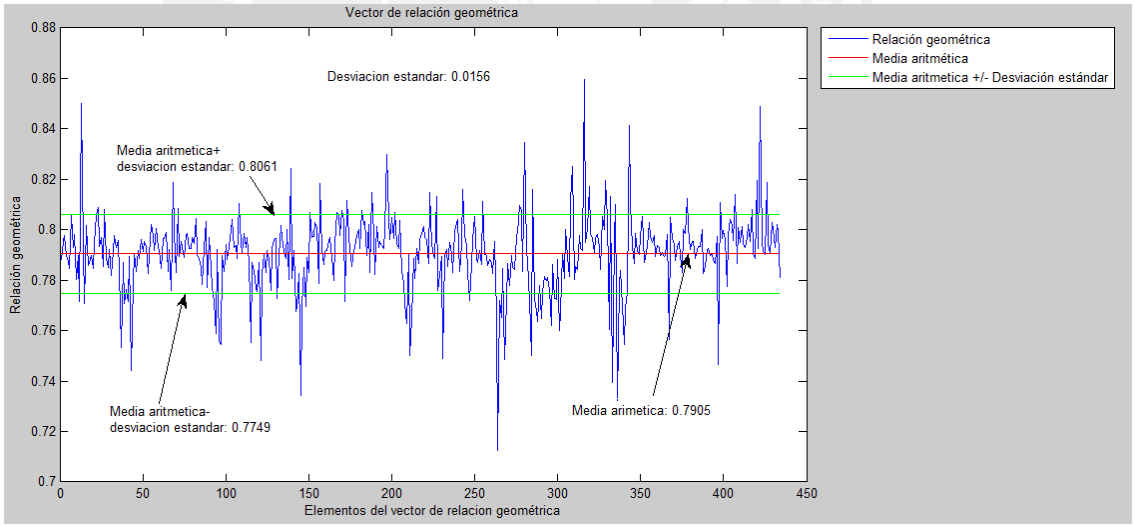


Desviación estándar: 0.0187

Media aritmética: 0.9563

Imagen 5

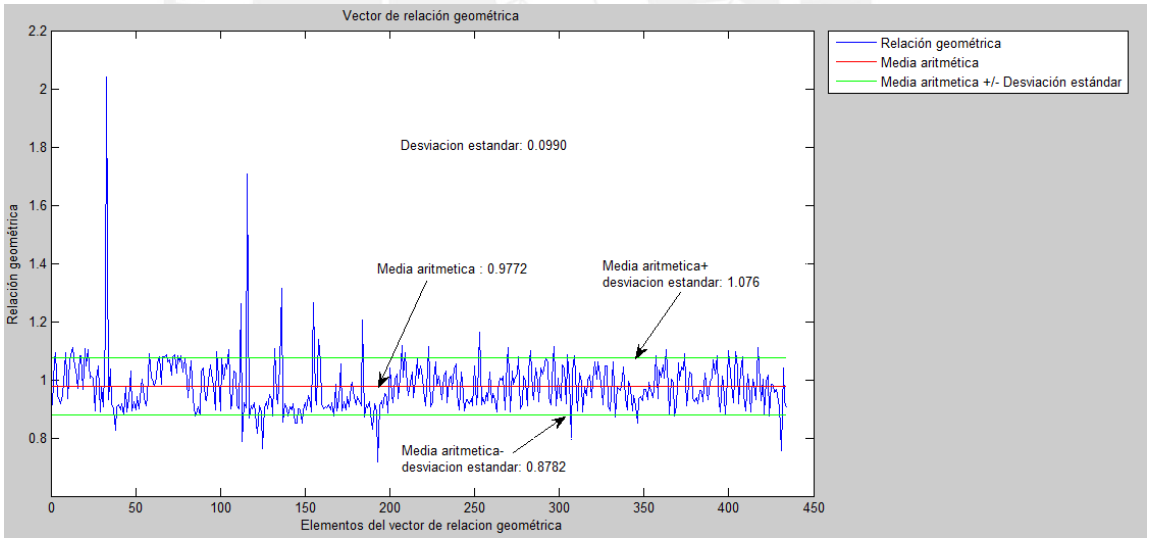




Desviación estándar: 0.0156

Media aritmética: 0.7905

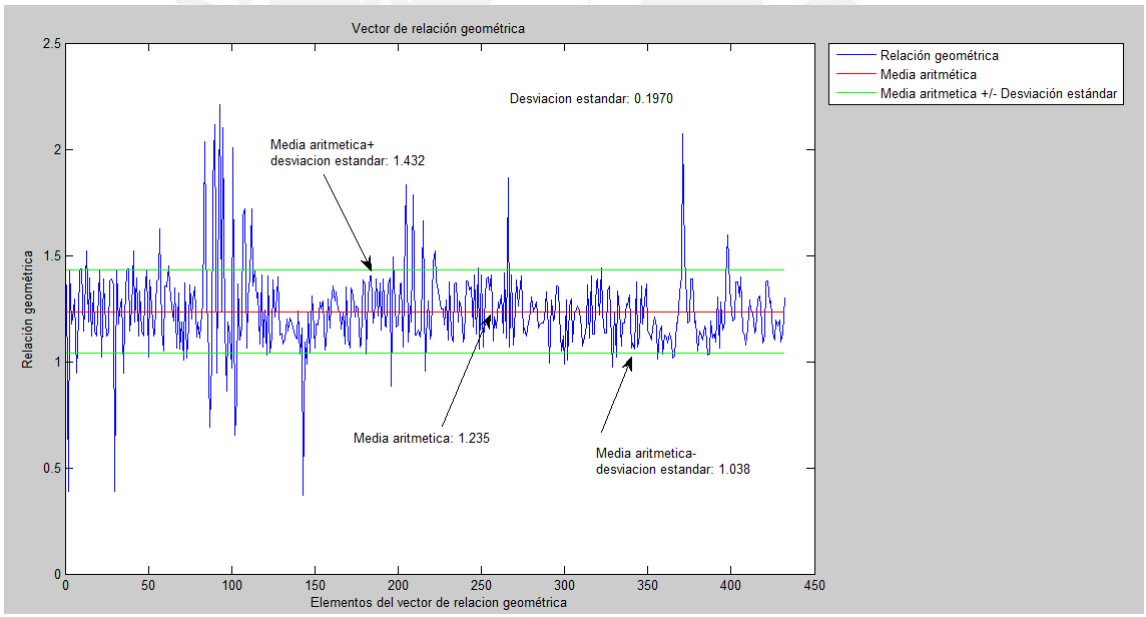
Imagen 6



Desviación estándar: 0.099

Media aritmética: 0.9772

Imagen 7



Desviación estándar: 0.197

Media aritmética: 1.235

ANEXO 5: Imágenes aéreas

En este anexo se presentan algunas de las imágenes aéreas obtenidas y rectificadas, cada una de las imágenes se presentan rectificadas de acuerdo a los ángulos de rotación, datos proporcionados por el sensor. De las 52 imágenes procesadas se obtuvieron 32 imágenes correctamente rectificadas, las cuales se califican como "correcta". Luego de mostrarse las 52 imágenes antes mencionadas, se presentan 6 imágenes que no necesitan rectificación.

Imagen 1

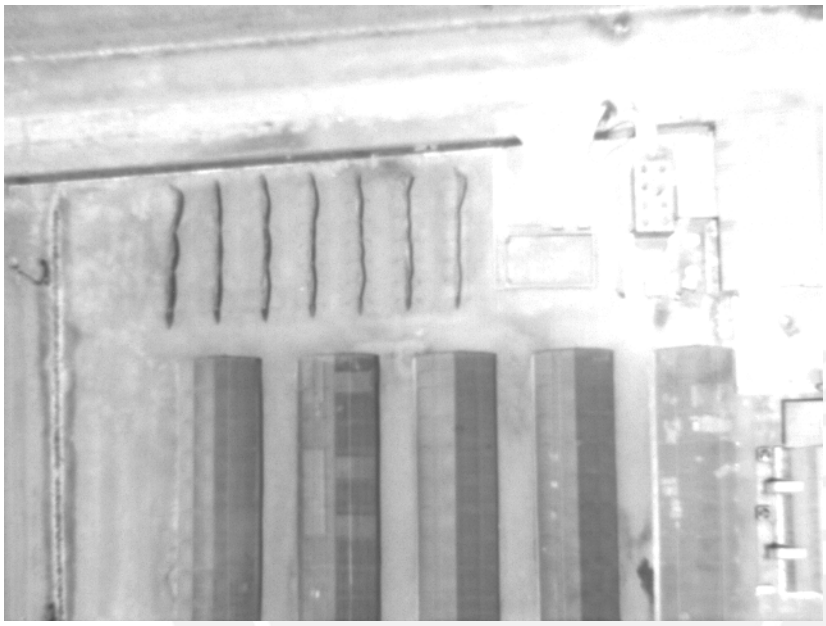
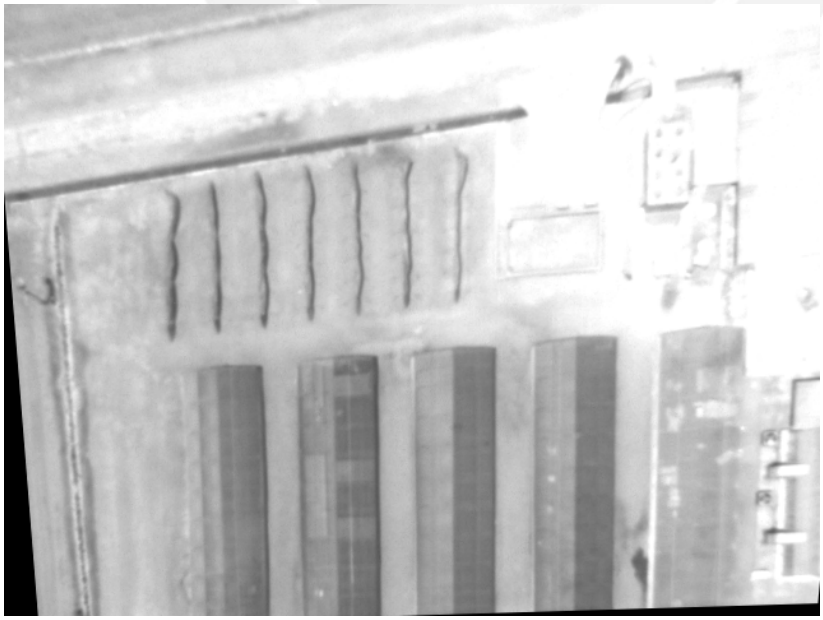


Imagen rectificada



r: -12 p: -5 y:0

Imagen 2

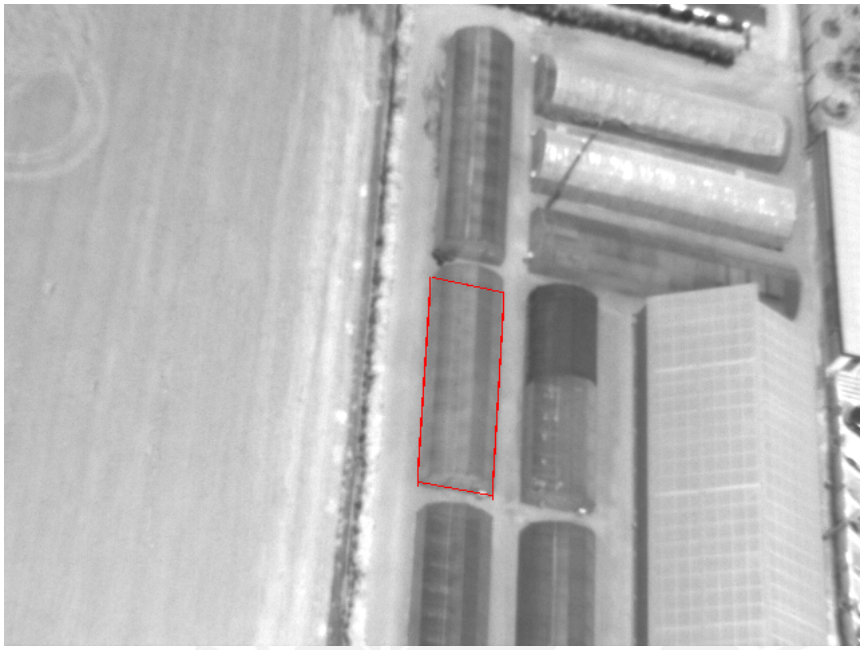
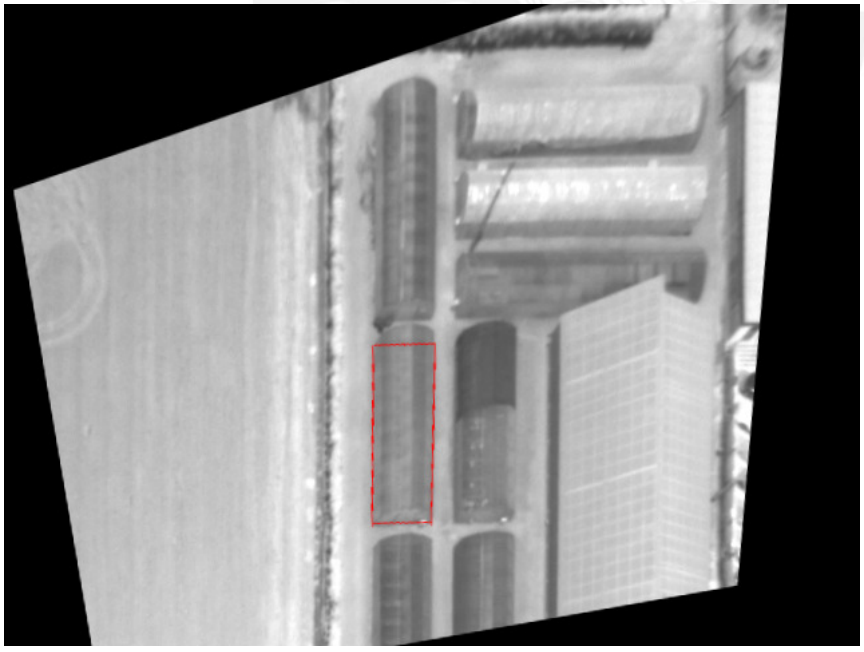


Imagen rectificada (correcta)



r: -30 p: 20 y:3

Imagen 3

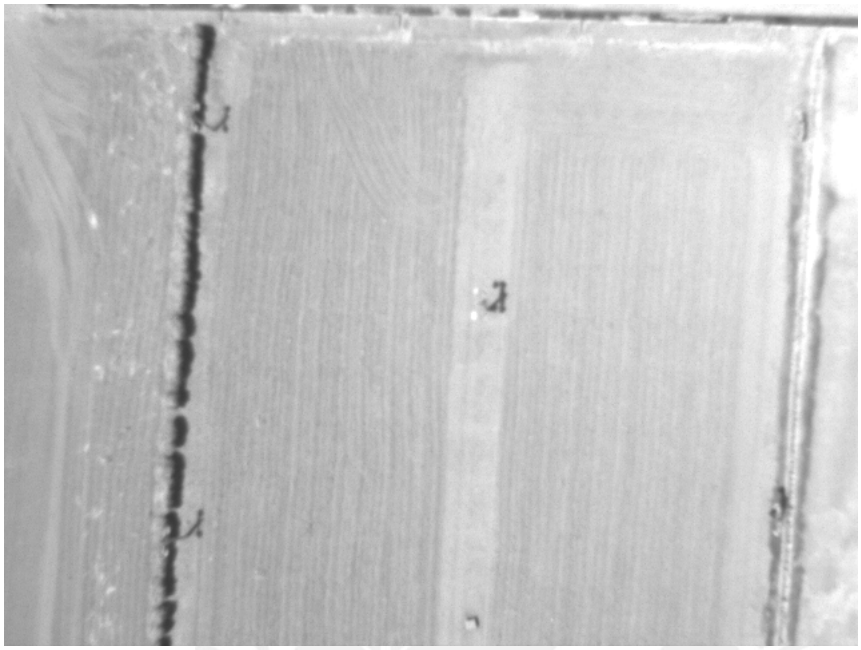
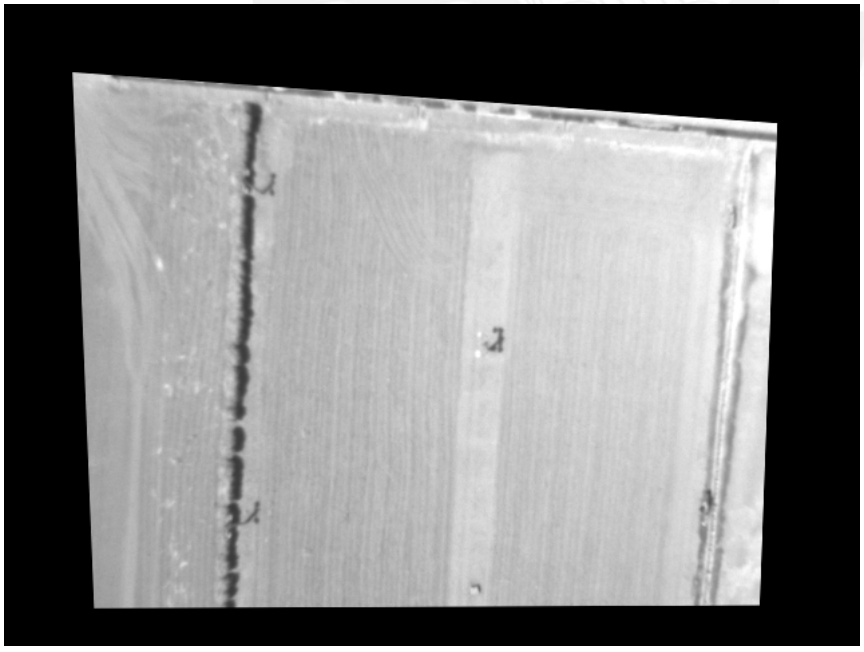


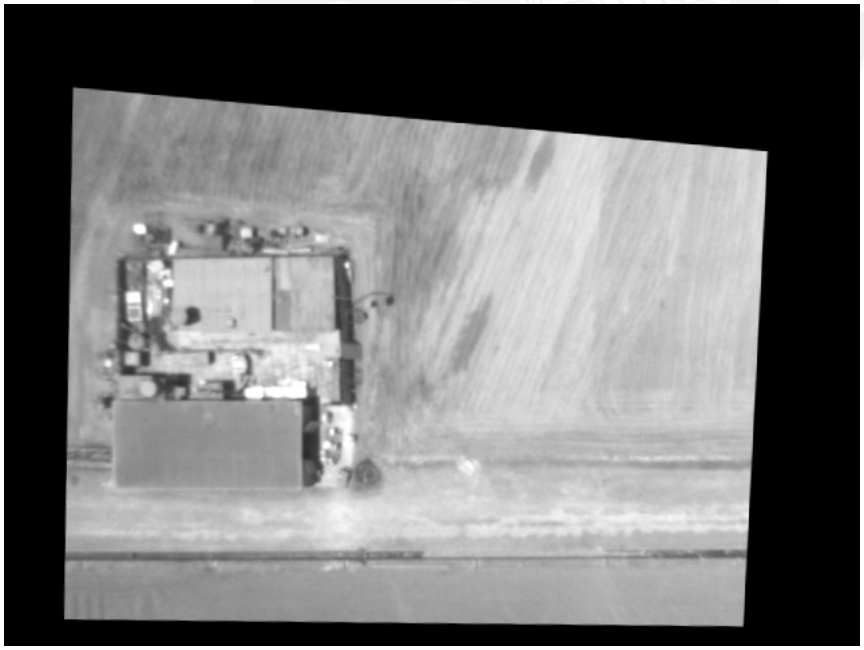
Imagen rectificada (correcta)



r: -9 p: 12 y:0

Imagen 4

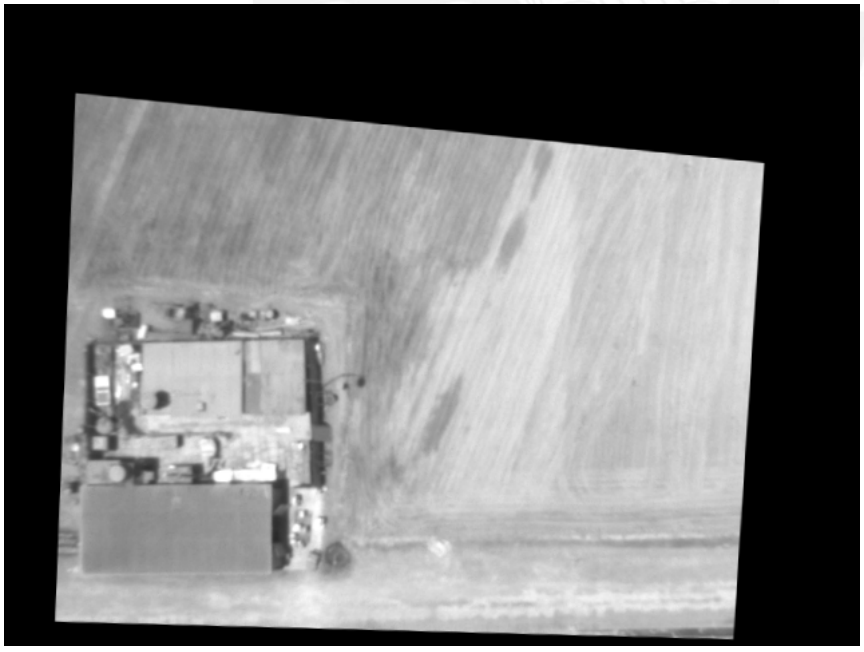
Imagen rectificada (correcta)



r: -4 p: 14 y:0

Imagen 5

Imagen rectificada



r: -3 p: -12 y:-3

Imagen 6

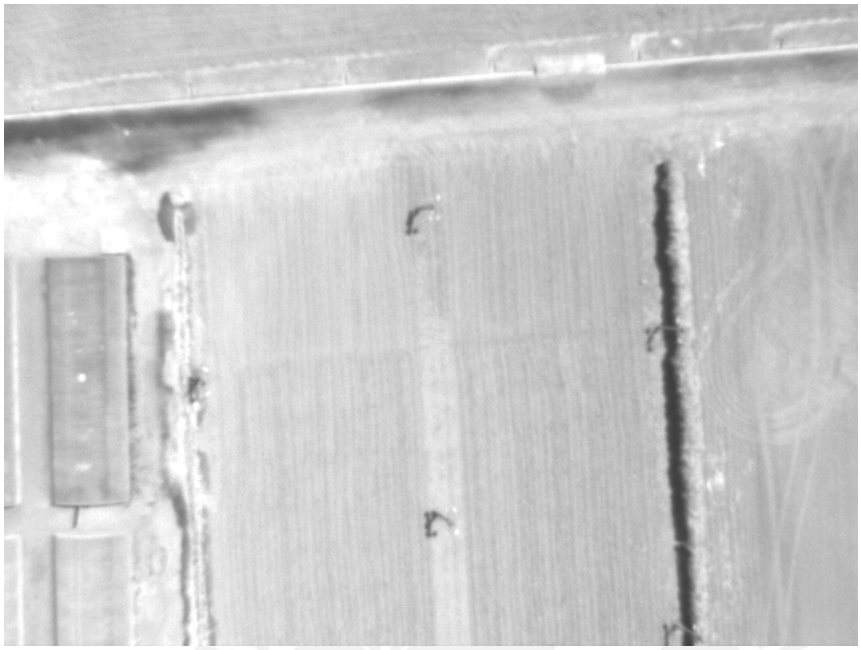
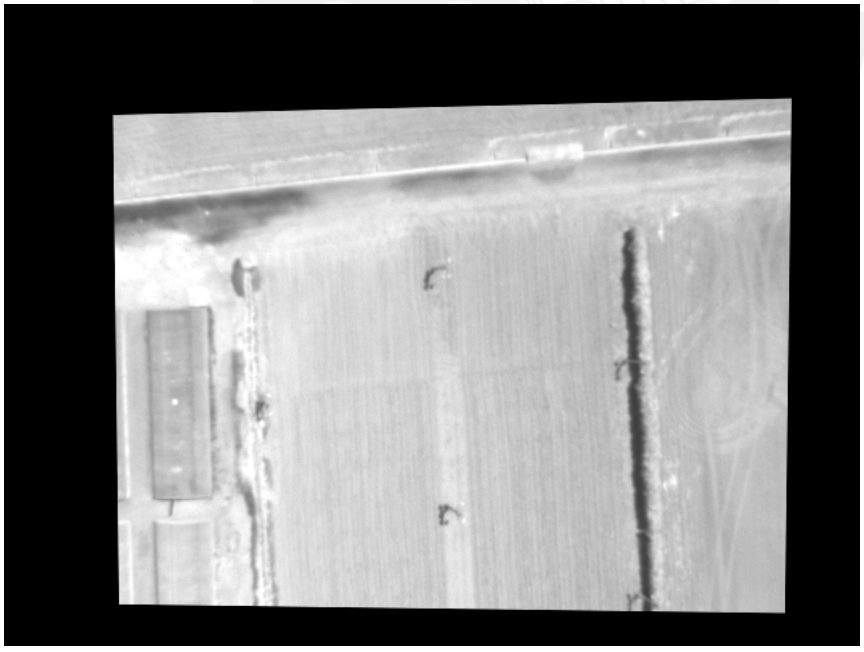


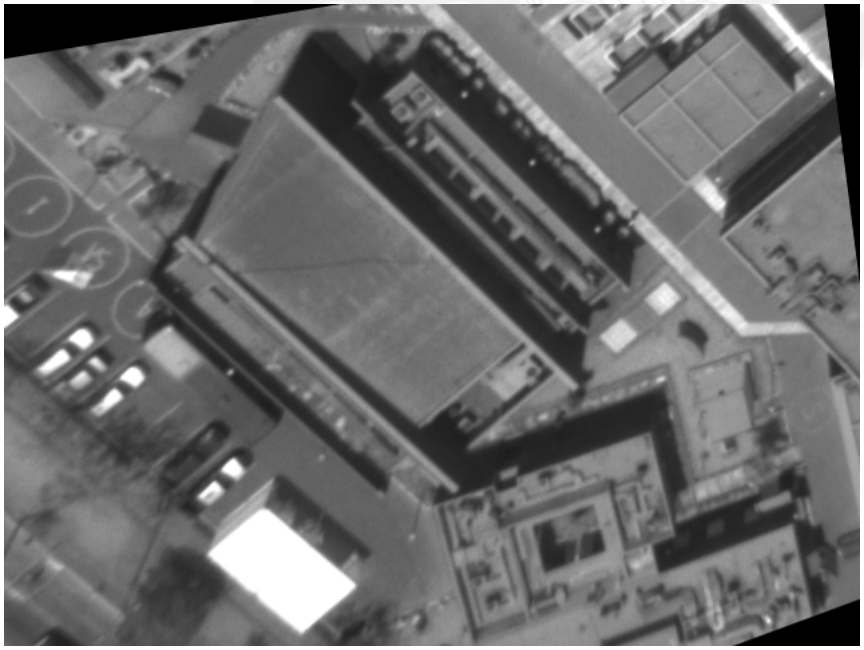
Imagen rectificada (correcta)



r: -3 p: 7 y: 0

Imagen 7

Imagen rectificada (correcta)



r: 32 p: -24 y:0

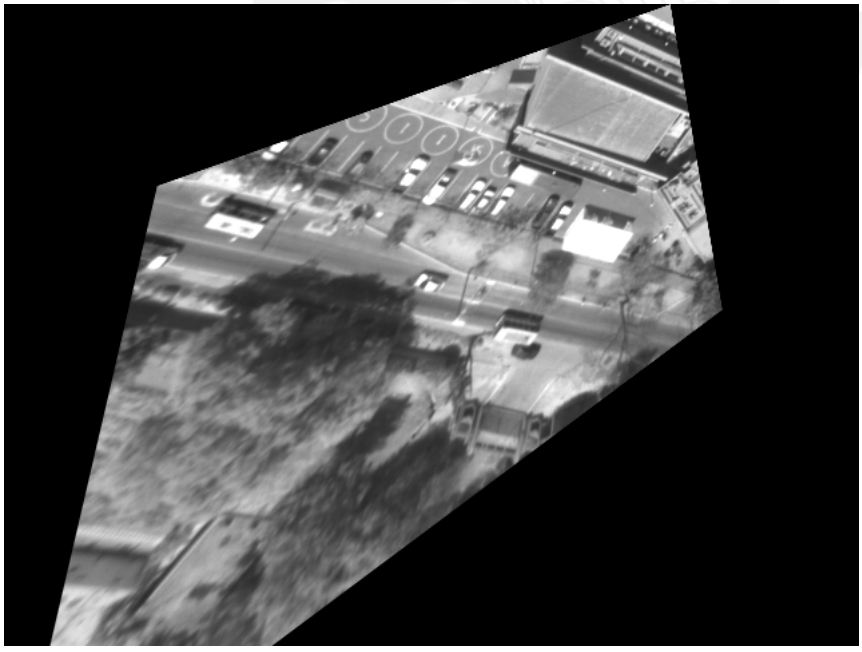
Imagen 8

Imagen rectificada

r: -13 p: -34 y:0

Imagen 9

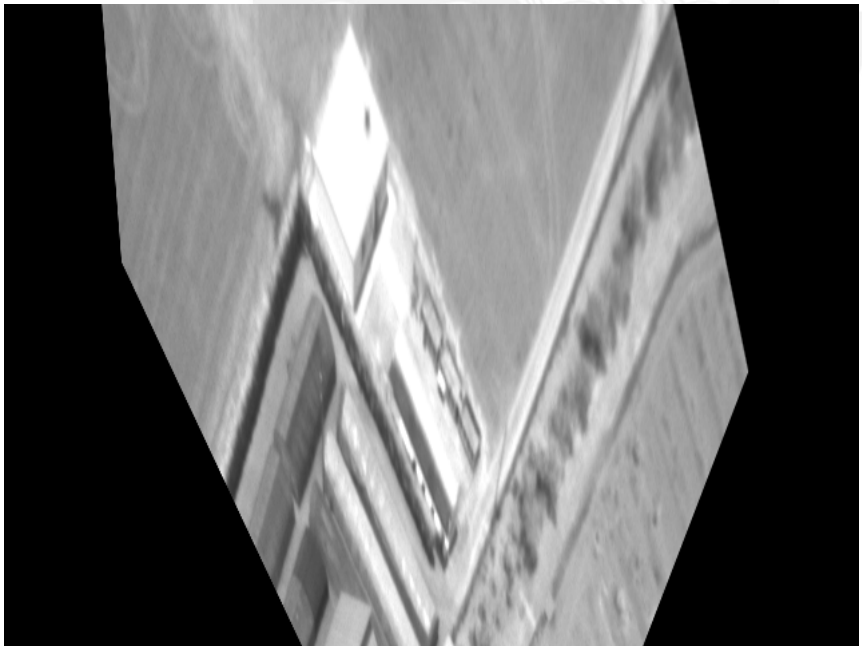
Imagen rectificada



r: 39 p: -40 y:0

Imagen 10

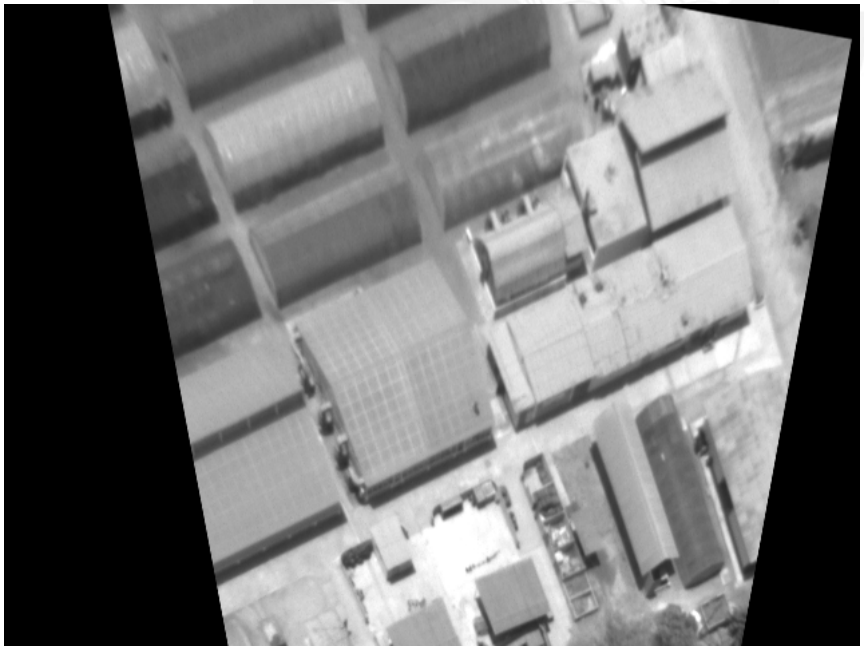
Imagen rectificada



r: -74 p: -41 y:0

Imagen 11

Imagen rectificada (correcta)



r: -50 p: -6 y:0

Imagen 12

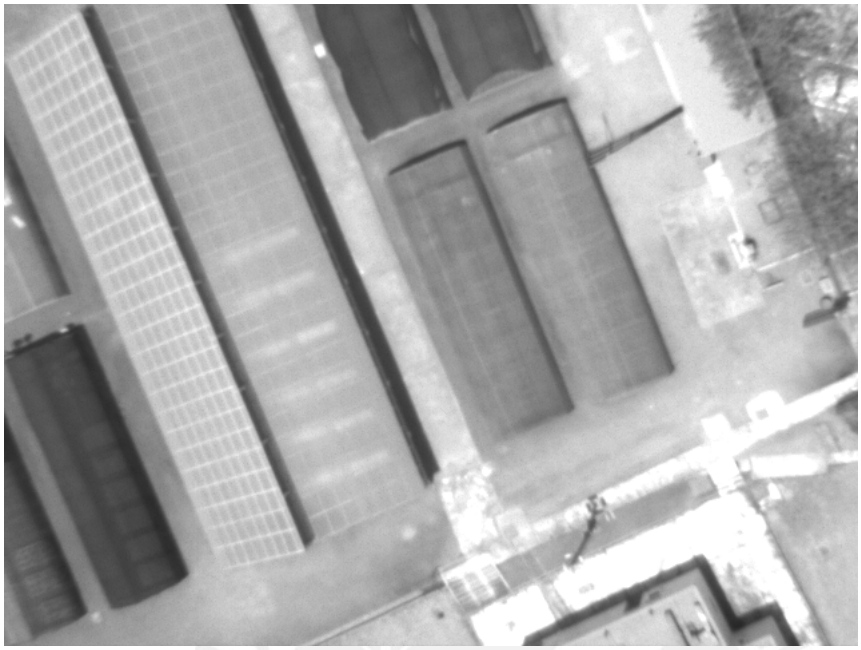
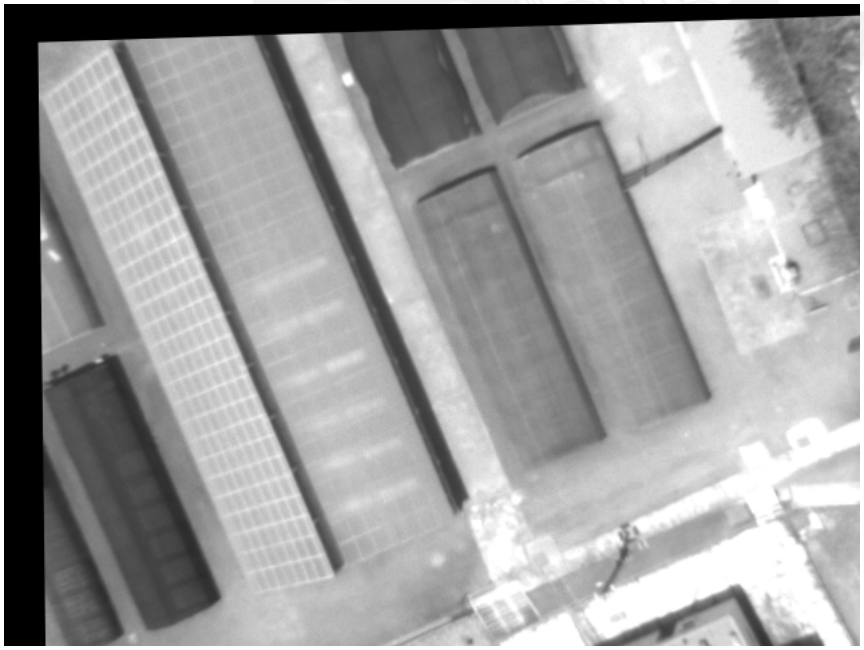


Imagen rectificada (correcta)



r: 3 p: 8 y:0

Imagen 13

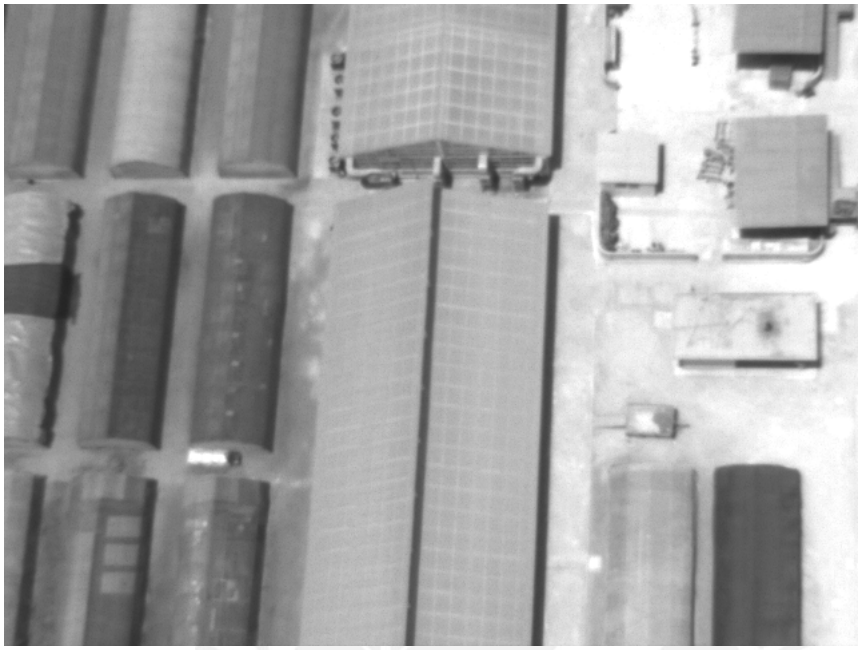
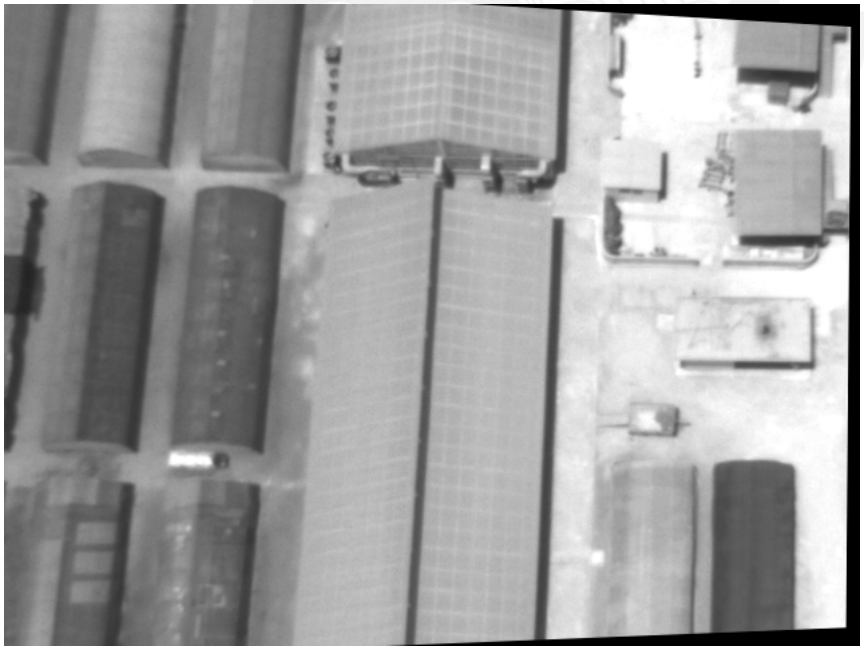


Imagen rectificada (correcta)



r: -3 p: -16 y:0

Imagen 14

Imagen rectificada

r: 1 p: -12 y:-4

Imagen 15

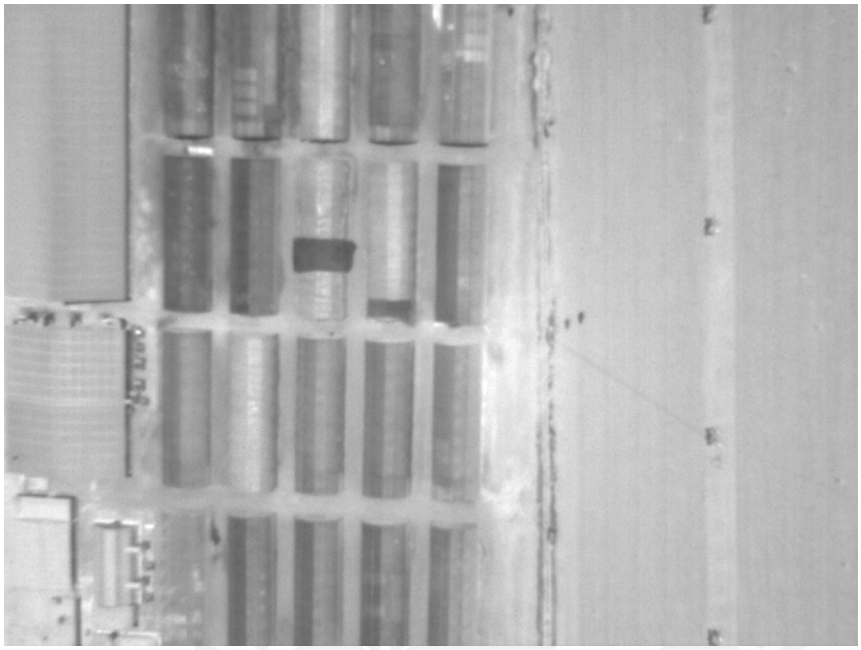
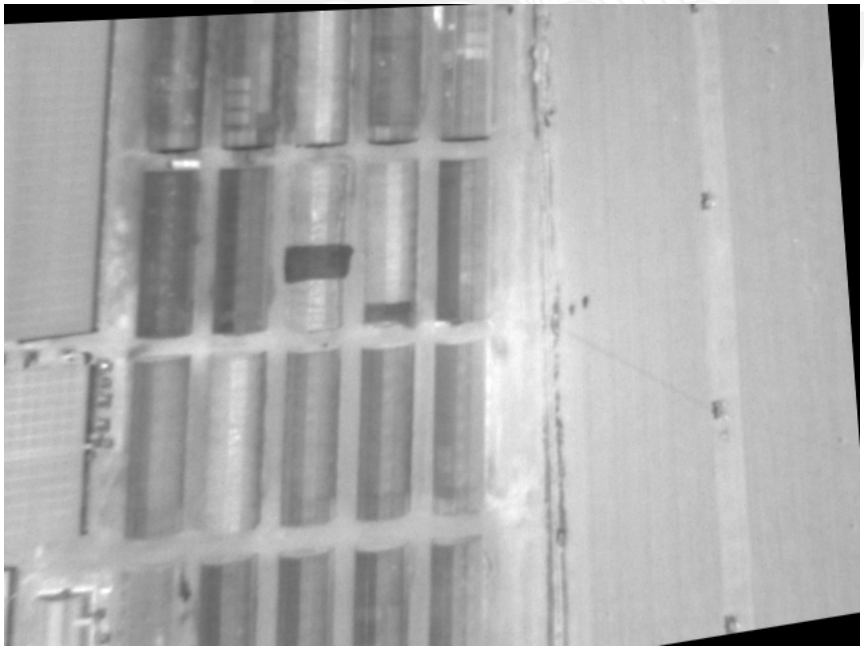


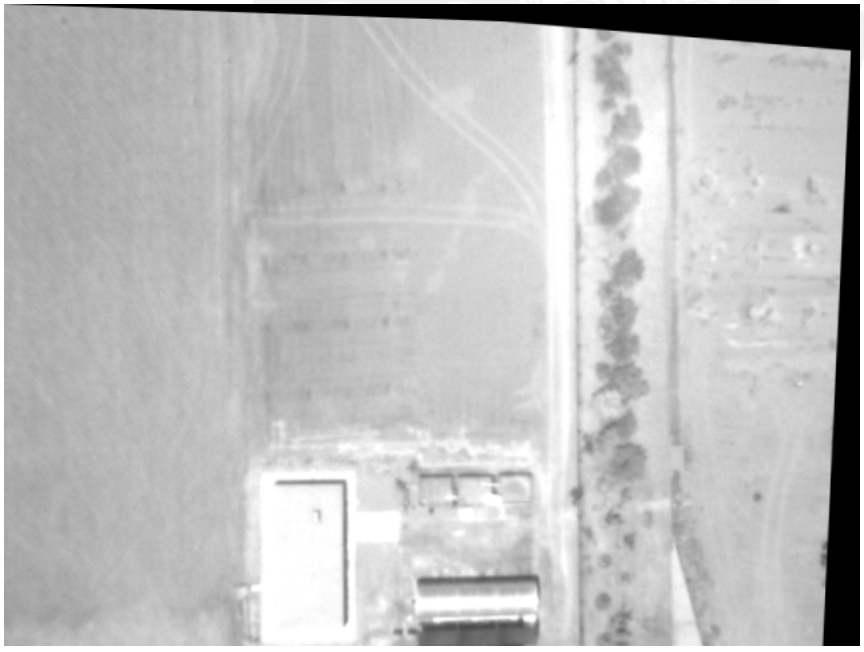
Imagen rectificada (correcta)



r: 18 p: -18 y: 5

Imagen 16

Imagen rectificada (correcta)



r: -4 p: -13 y=-2

Imagen 17

Imagen rectificada (correcta)

r: 19 p: 3 y:4

Imagen 18

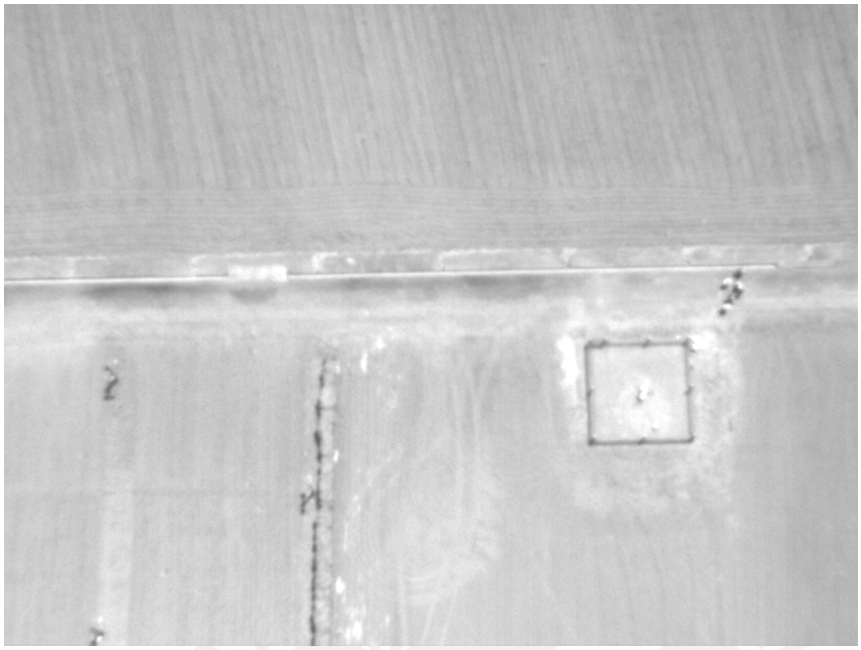
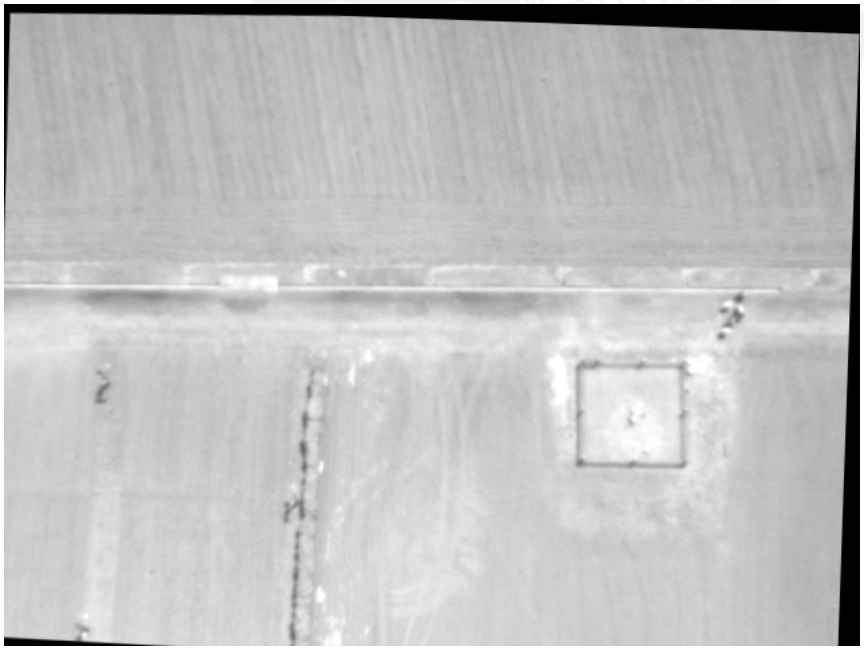


Imagen rectificada (correcta)



r: -2 p: 7 y: -2

Imagen 19

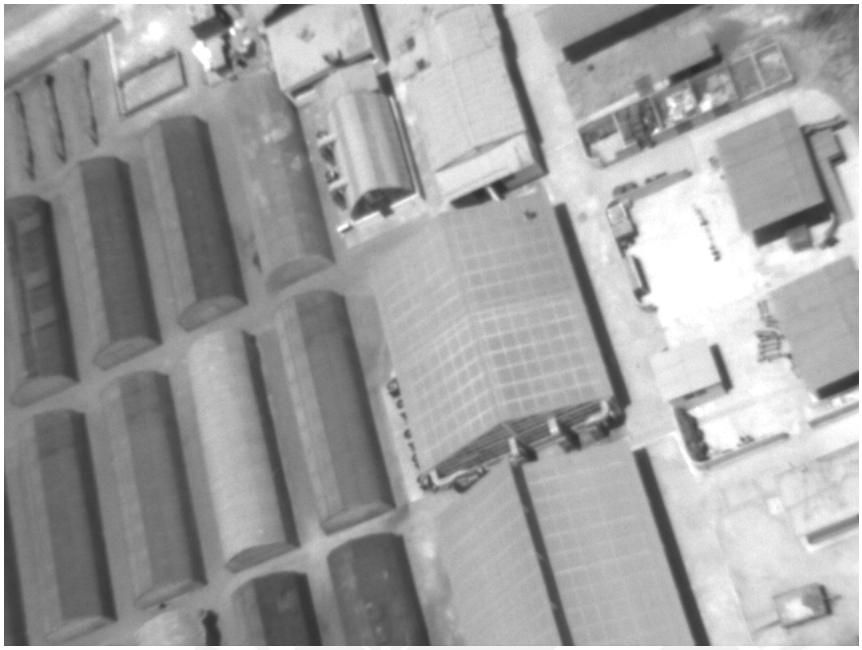
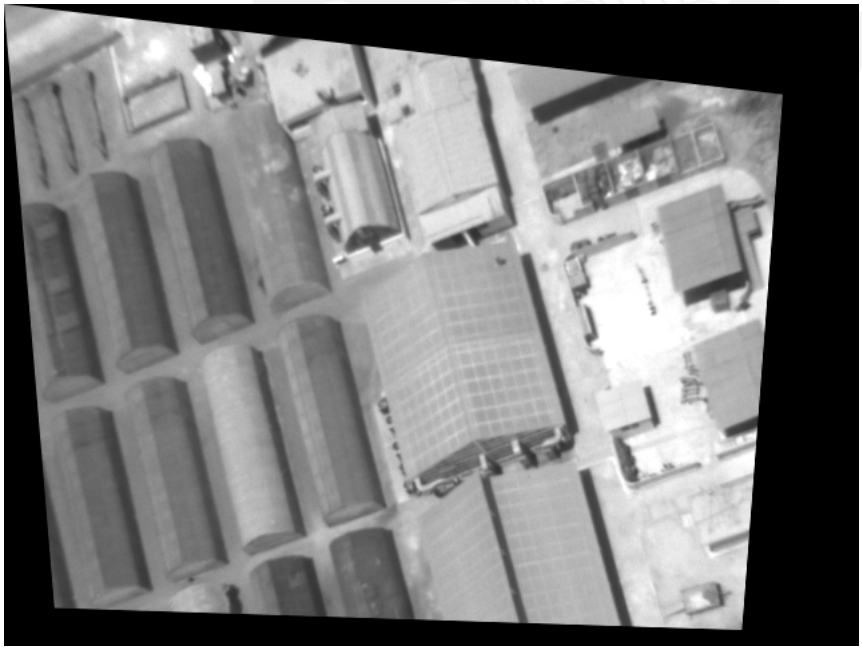


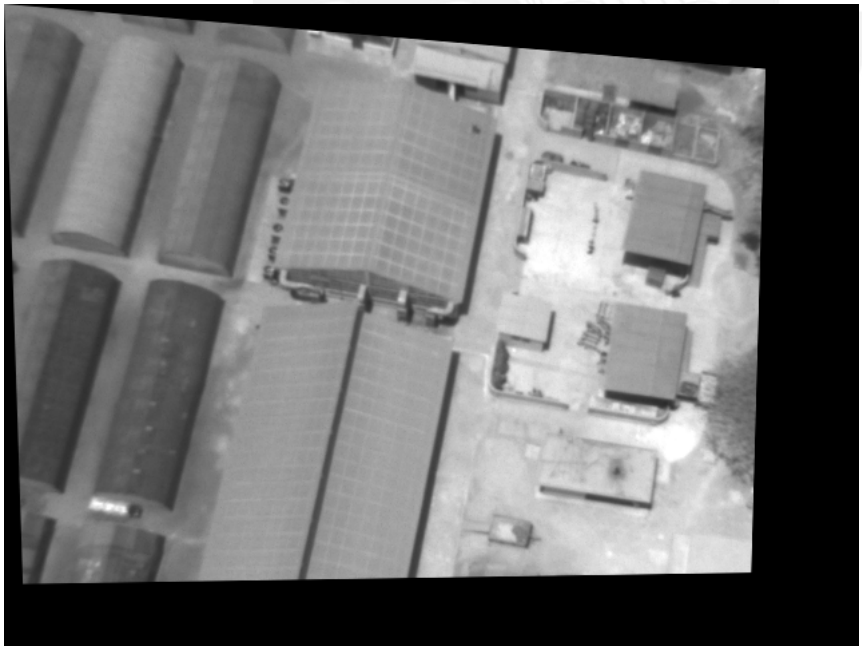
Imagen rectificada



r: -19 p: -12 y:0

Imagen 20

Imagen rectificada (correcta)



r: -7 p: -16 y:0

Imagen 21

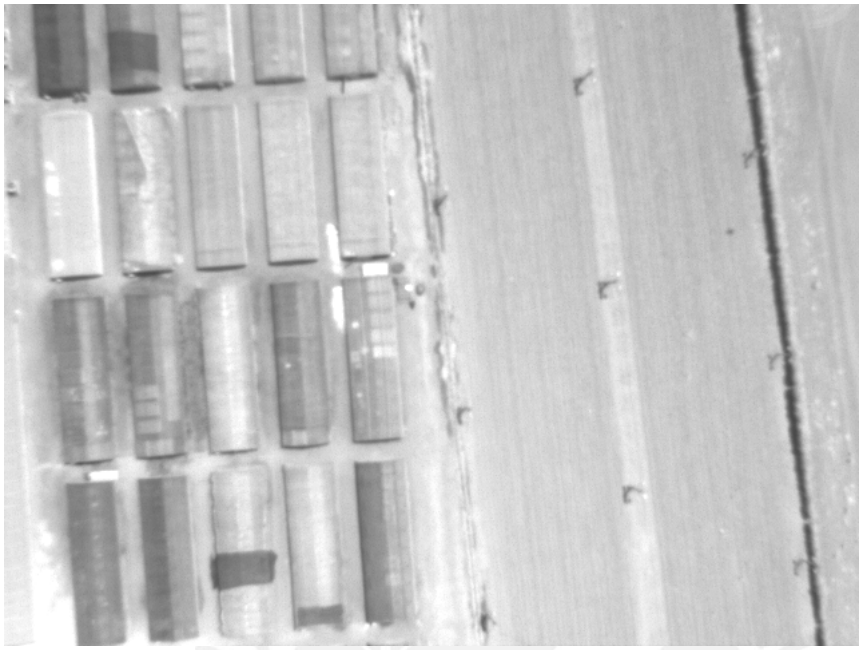
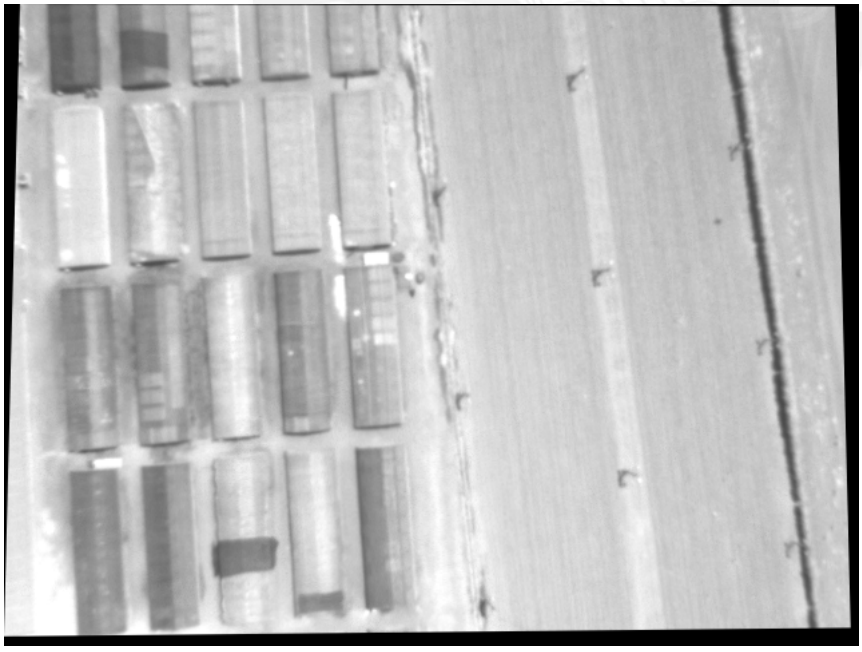


Imagen rectificada (correcta)



r: 6 p: -2 y: 0

Imagen 22

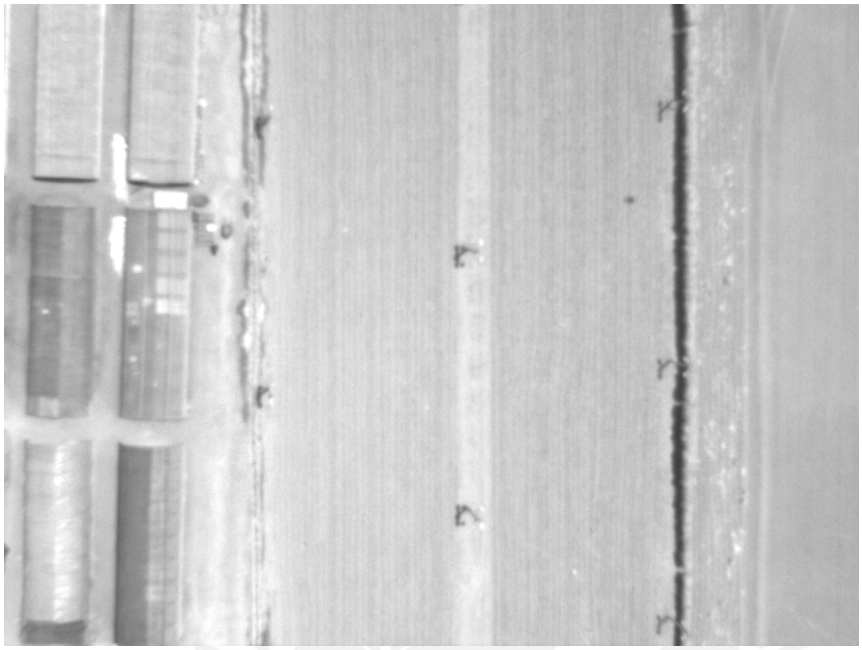
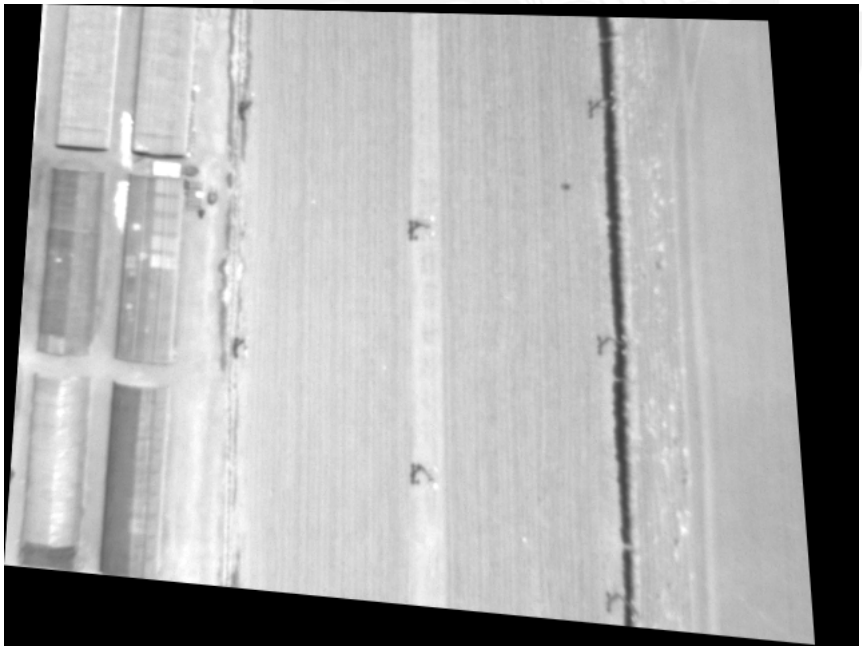


Imagen rectificada (correcta)



r: 17 p: 11 y: 0

Imagen 23

Imagen rectificada (correcta)

r: 38 p: 8 y:0

Imagen 24

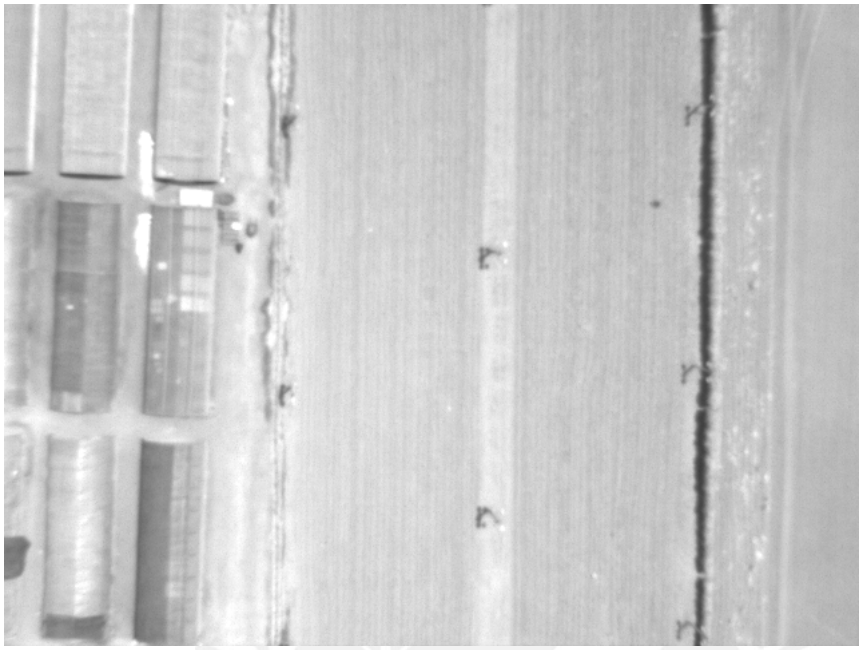
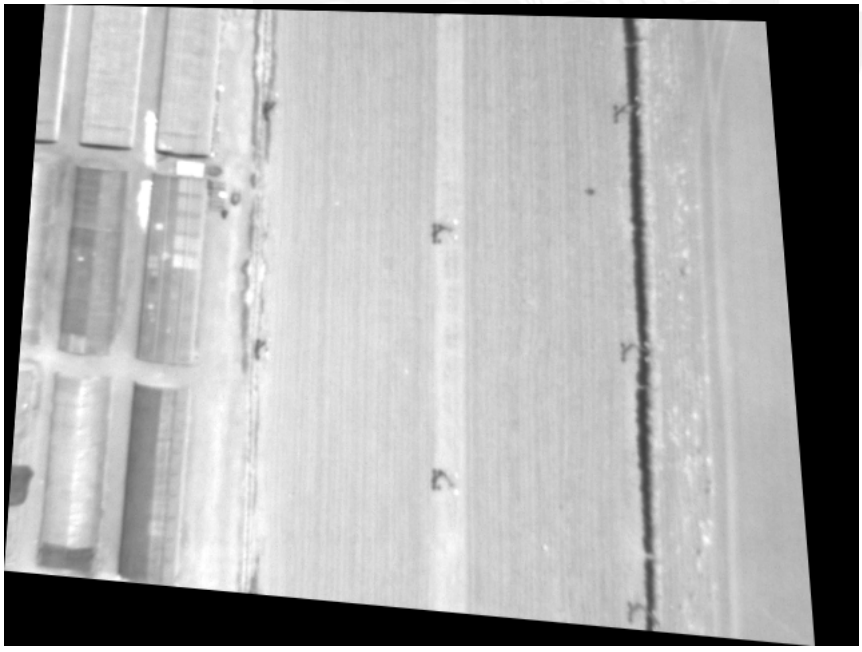


Imagen rectificada (correcta)



r: 18 p: 10 y: 0

Imagen 25

Imagen rectificada (correcta)

r: 29 p: -17 y:0

Imagen 26

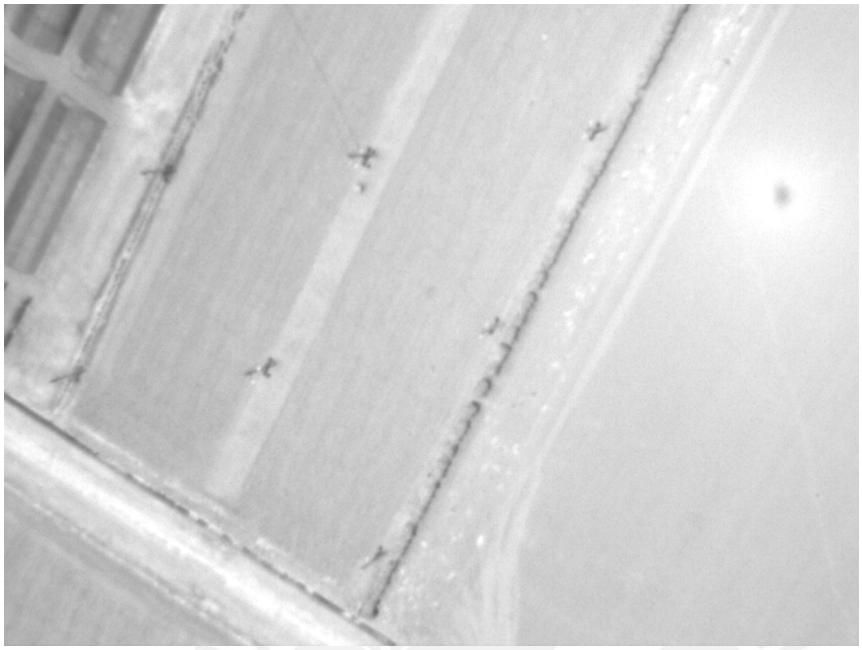
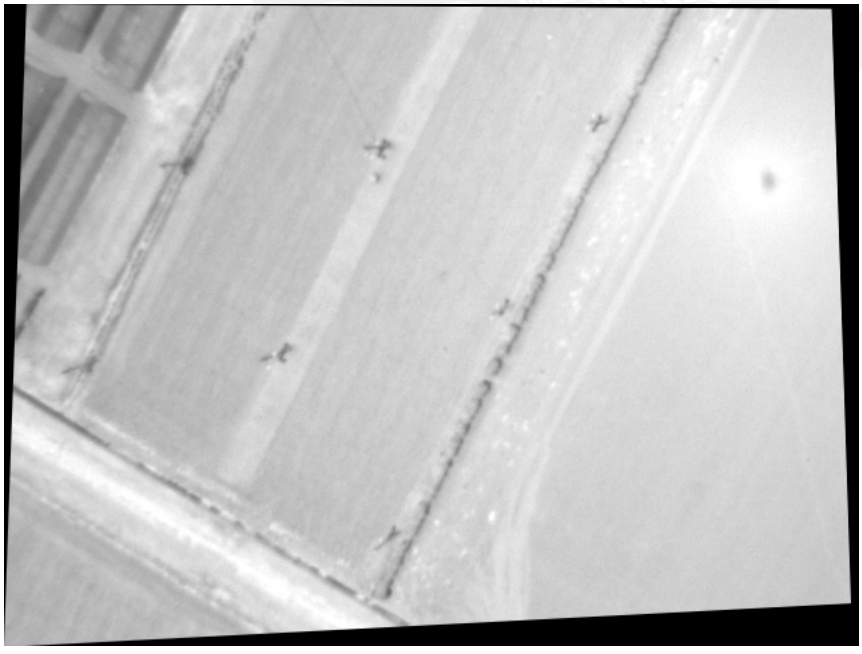


Imagen rectificad (correcta)



r: 8 p: -9 y: 0

Imagen 27

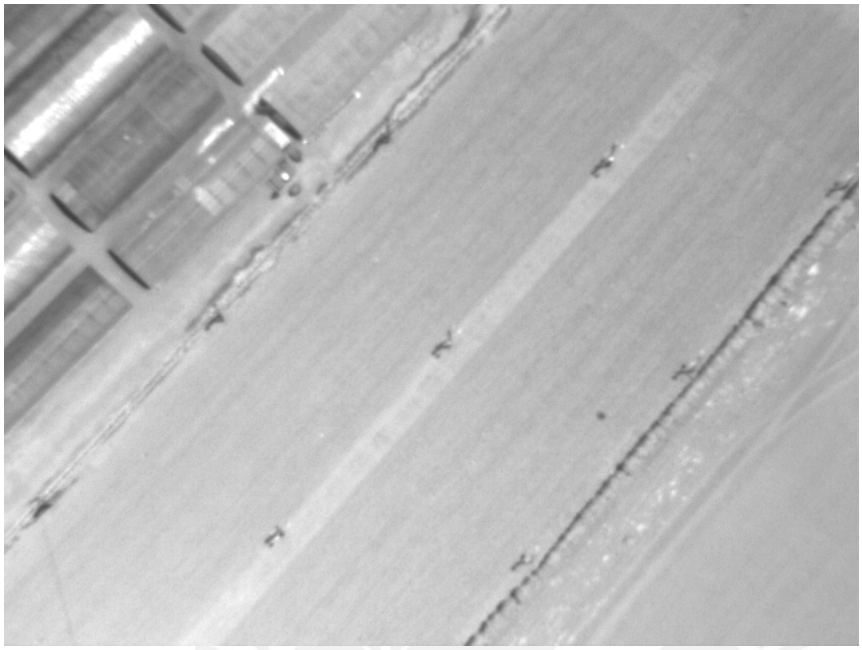
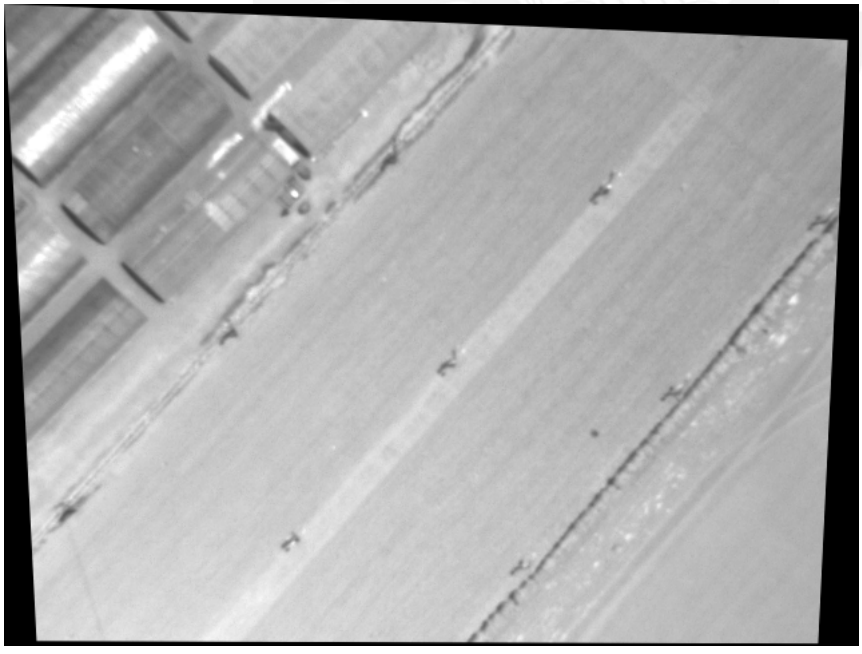


Imagen rectificada (correcta)



r: -12 p: -6 y: 0

Imagen 28

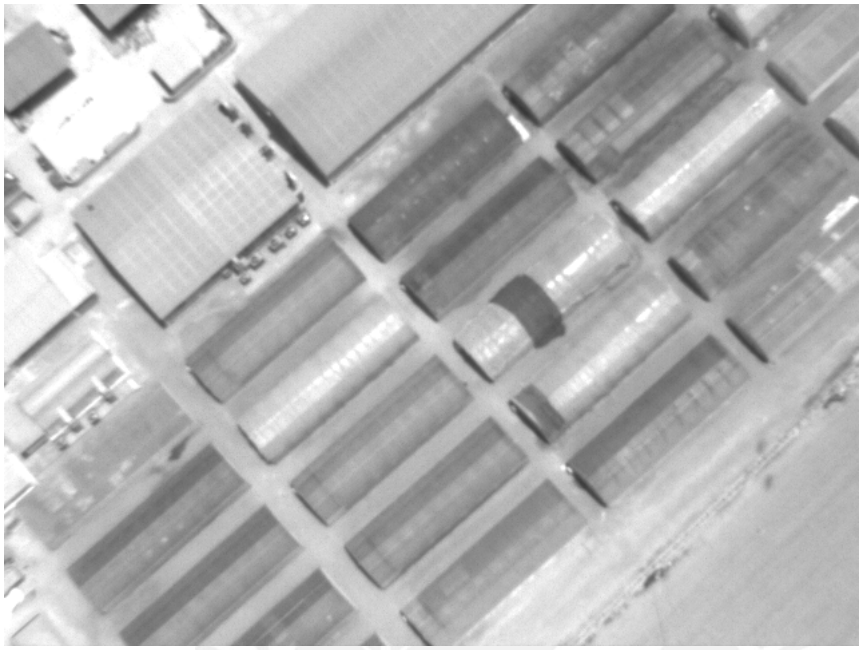
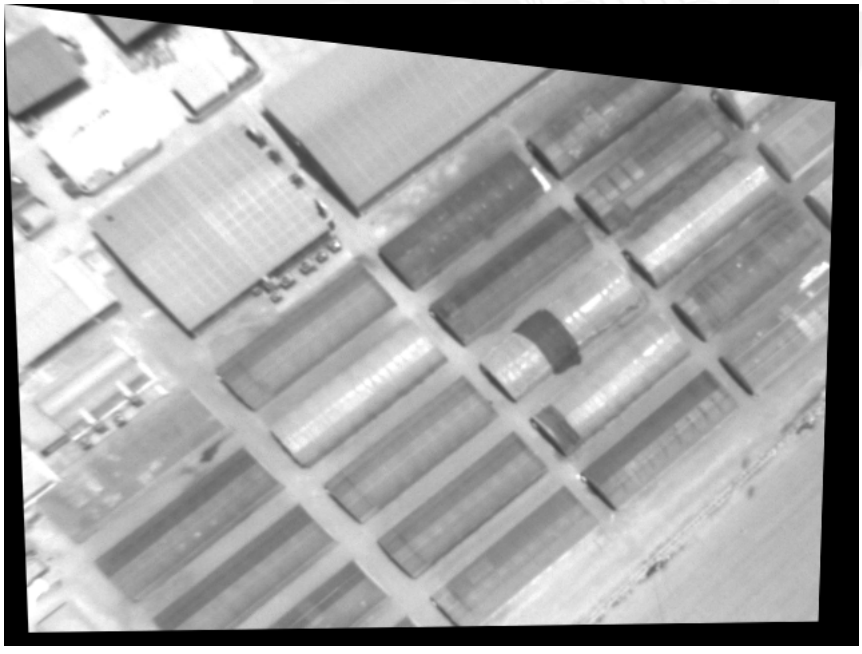


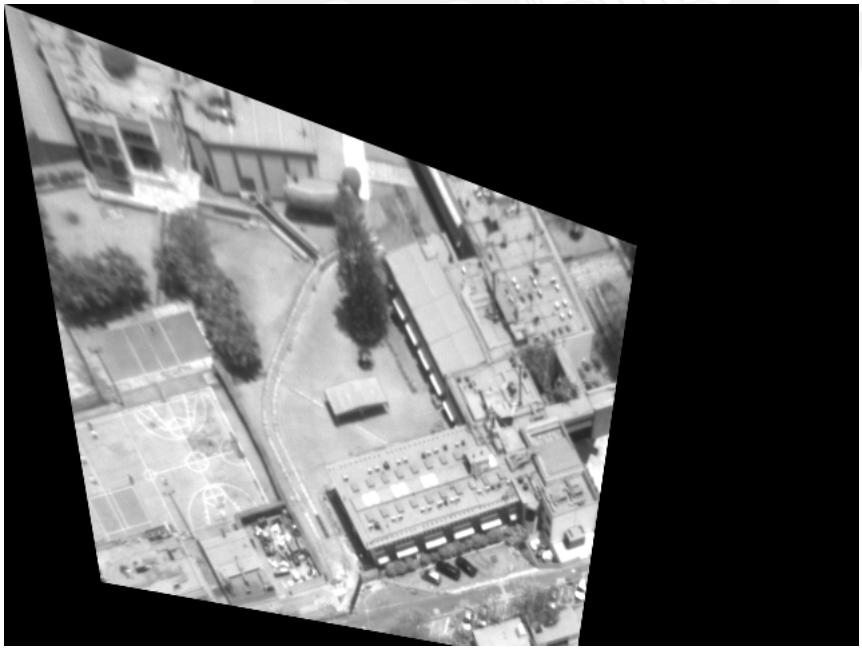
Imagen rectificada (correcta)



r: -8 p: -21 y: 0

Imagen 29

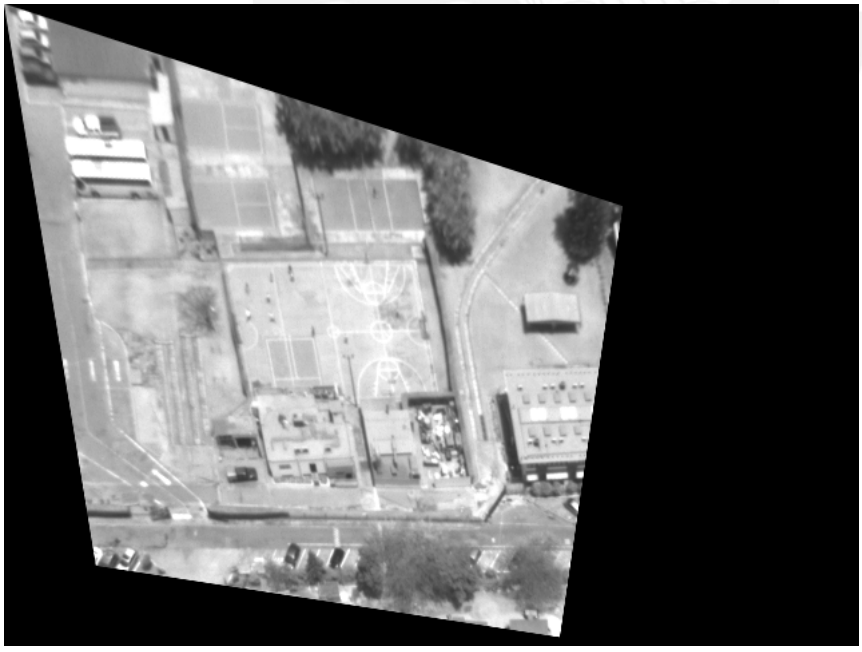
Imagen rectificada



r: -38 p: -20 y: 0

Imagen 30

Imagen rectificada



r: -35 p: -17 y: 0

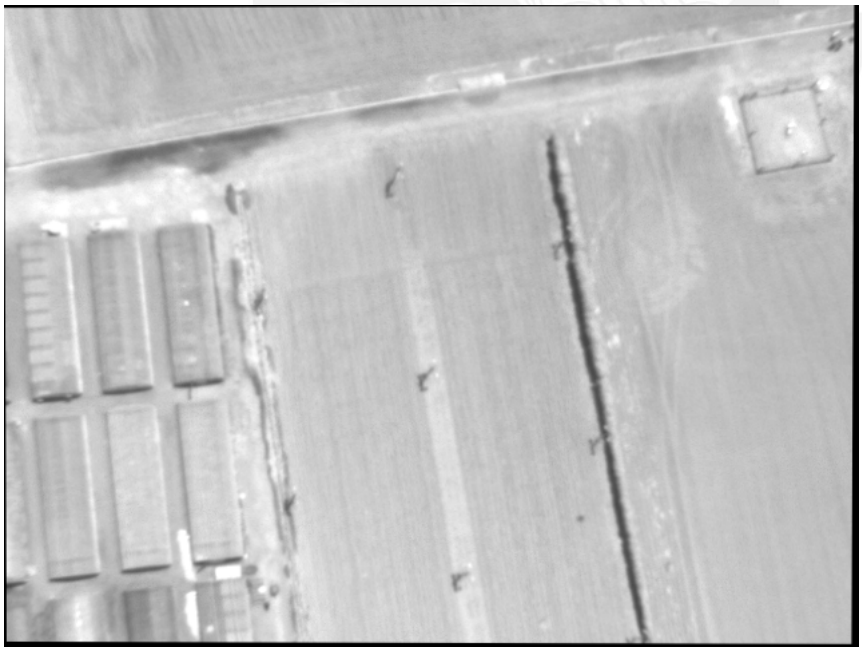
Imagen 31

Imagen rectificada (correcta)

r: -24 p: -15 y:0

Imagen 32

Imagen rectificada (correcta)



r: -1 p: 1 y: 0

Imagen 33

Imagen rectificada (correcta)

r: 26 p: -1 y: 0

Imagen 34

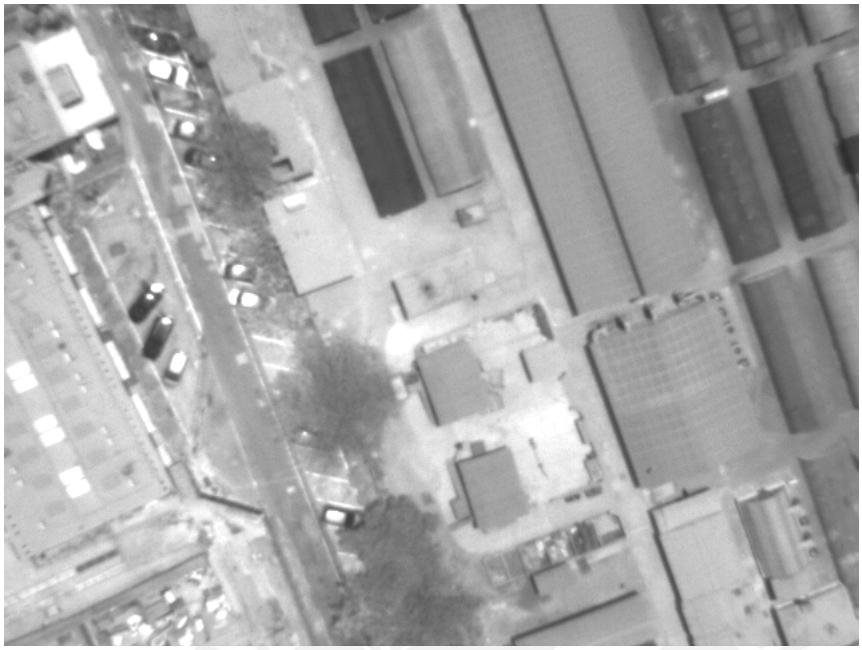
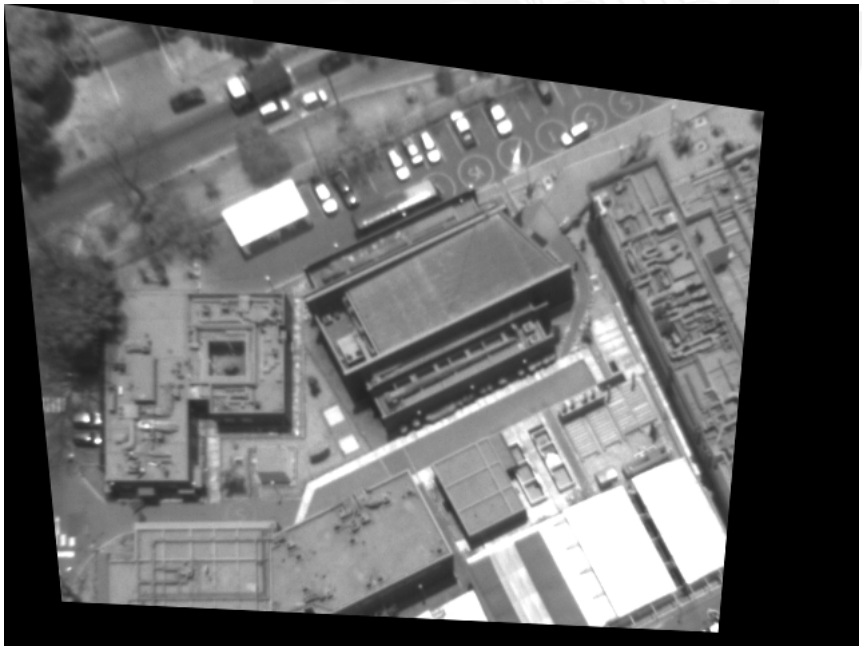


Imagen rectificada (correcta)

r: 9 p: 0 y: 0

Imagen 35

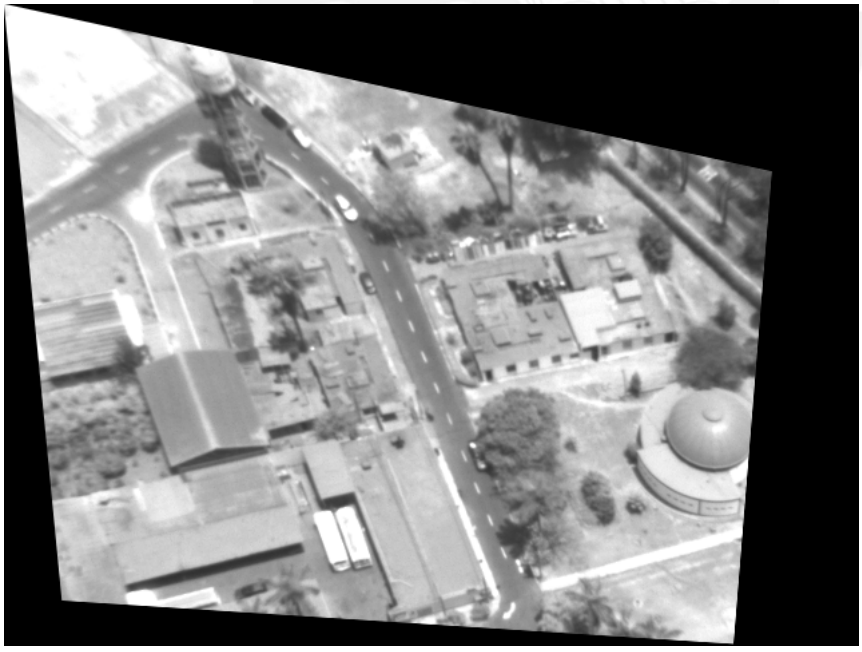
Imagen rectificada (correcta)



r: -22 p: -13 y: 0

Imagen 36

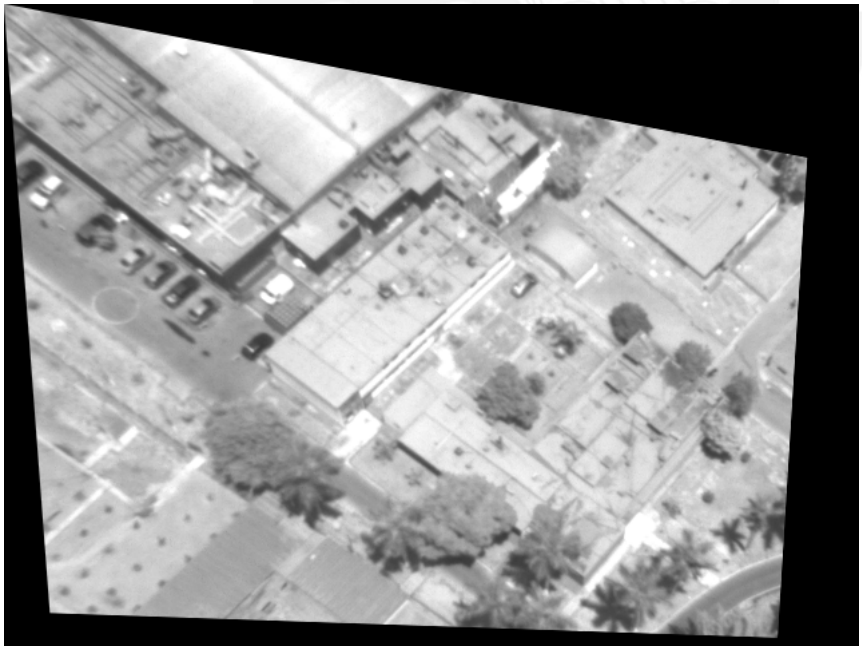
Imagen rectificada



r: -20 p: -22 y: 0

Imagen 37

Imagen rectificada



r: -15 p: -24 y:0

Imagen 38

Imagen rectificada (correcta)

r: -12 p: - 7 y:0

Imagen 39

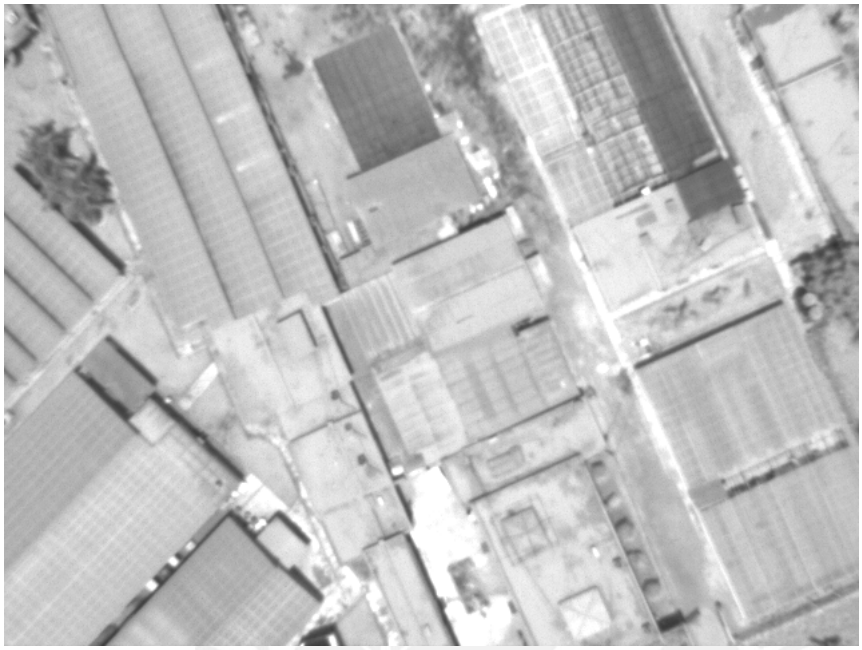
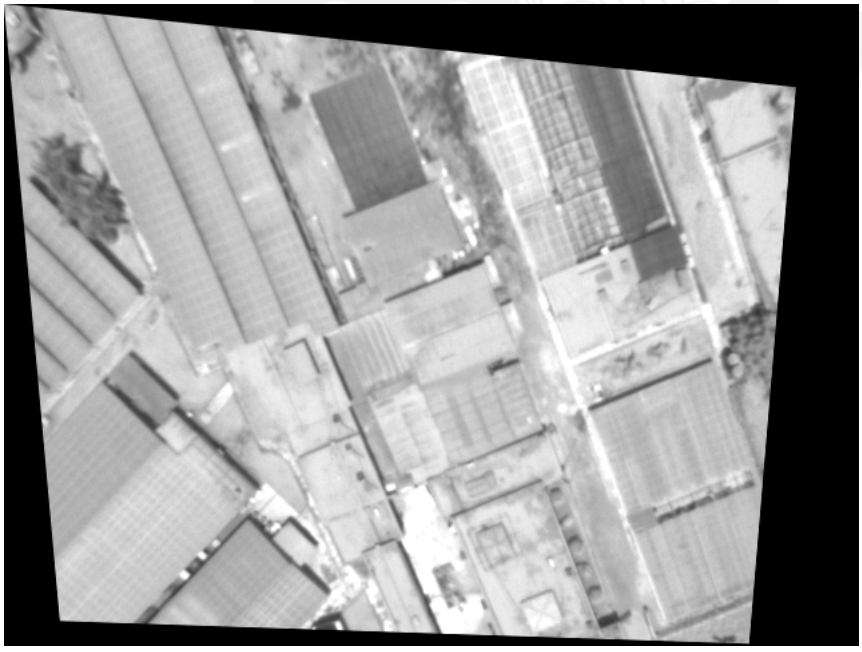


Imagen rectificada (correcta)



r: -21 p: -10 y:0

Imagen 40

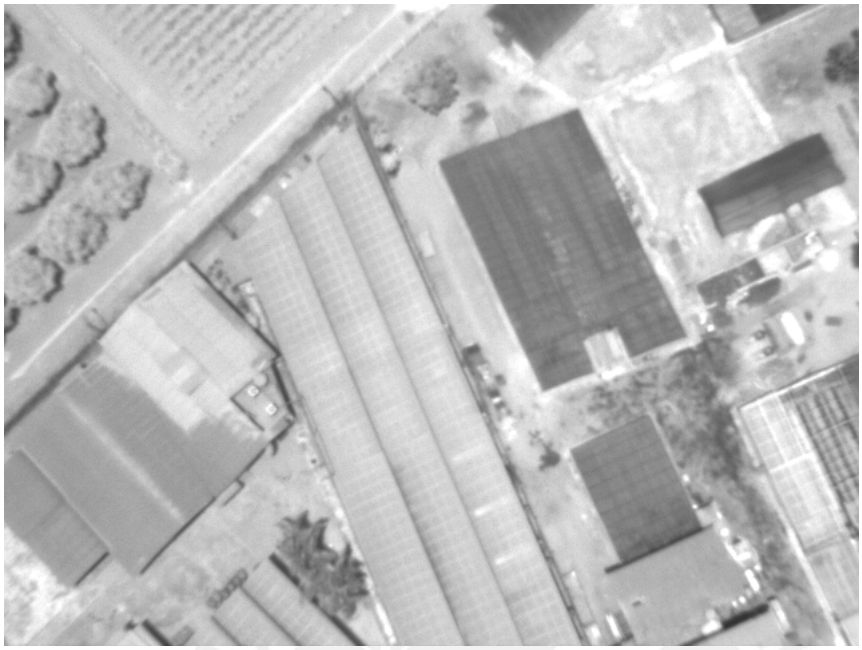
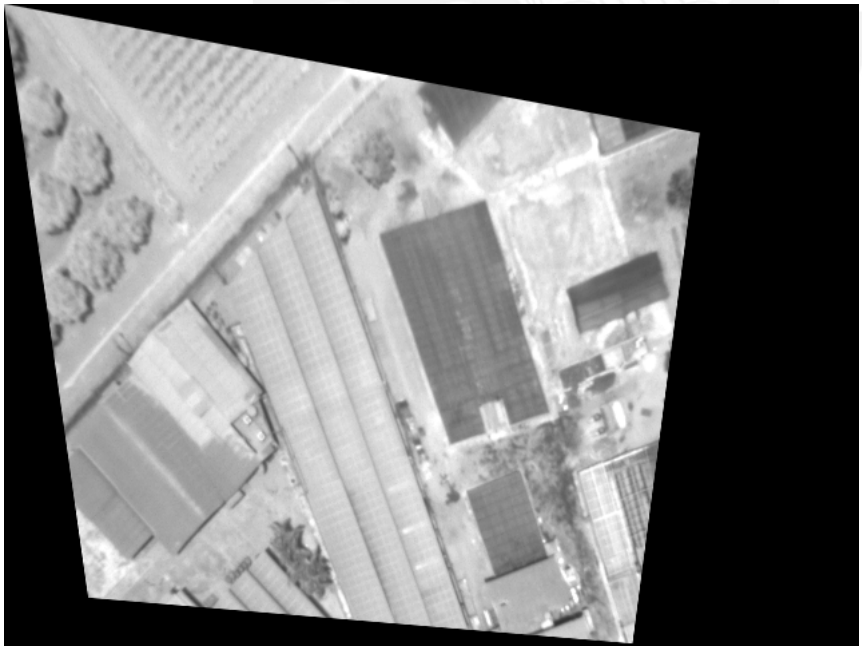


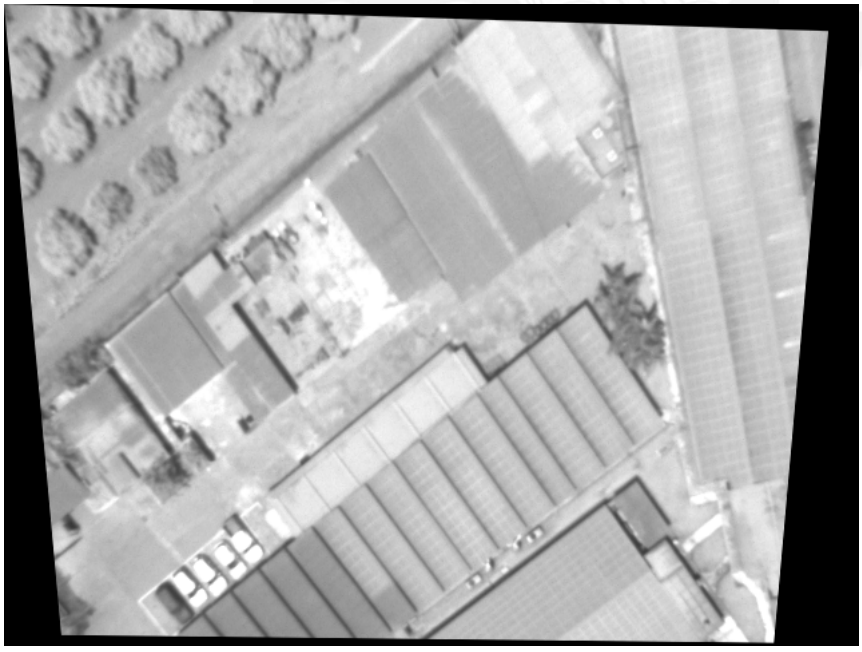
Imagen rectificada (correcta)



r: -34 p: -11 y:0

Imagen 41

Imagen rectificada (correcta)



r: -22 p: -3 y:0

Imagen 42

Imagen rectificada

r: -3 p: -15 y:0

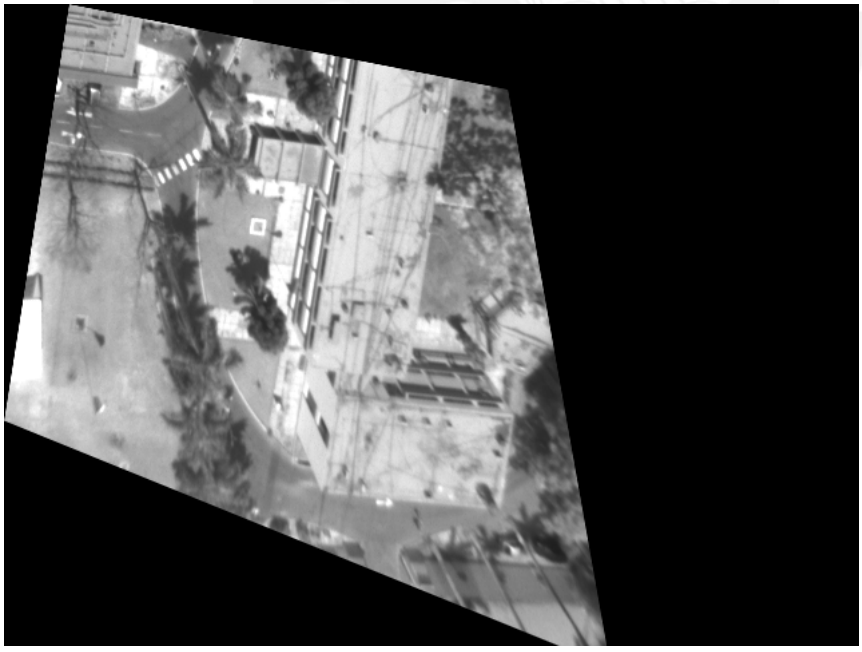
Imagen 43

Imagen rectificada

r: 9 p: -7 y:0

Imagen 44

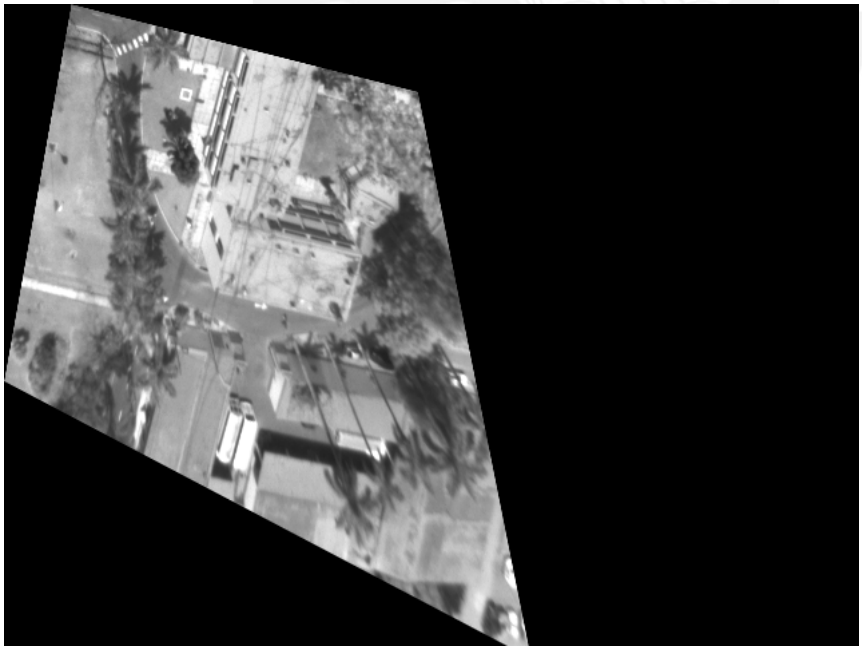
Imagen rectificada (correcta)



r: 41 p: 19 y:0

Imagen 45

Imagen rectificada (correcta)



r: 48 p: 19 y:0

Imagen 46

Imagen rectificada

r: 46 p: 5 y:0

Imagen 47

Imagen rectificada

r: 40 p: -4 y:0

Imagen 48

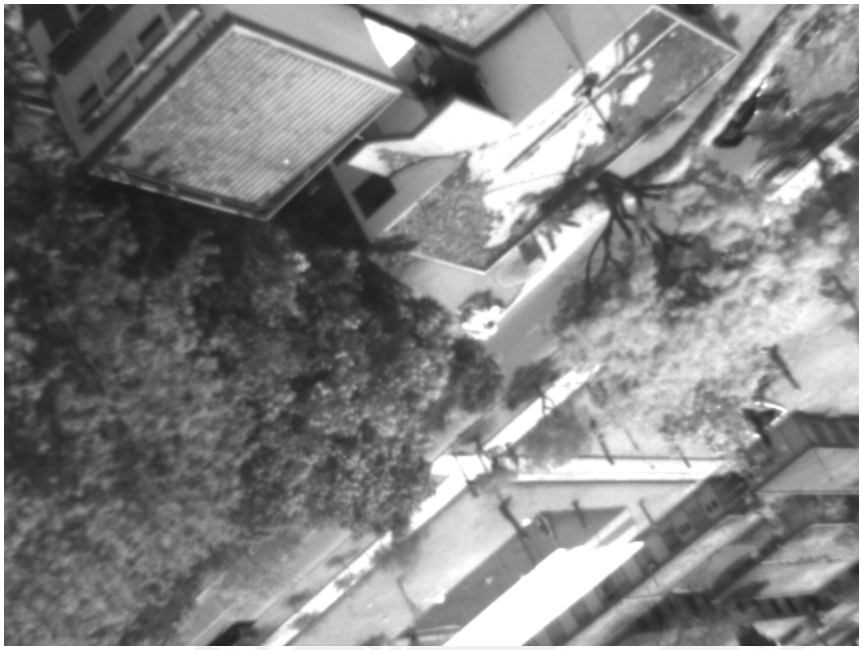
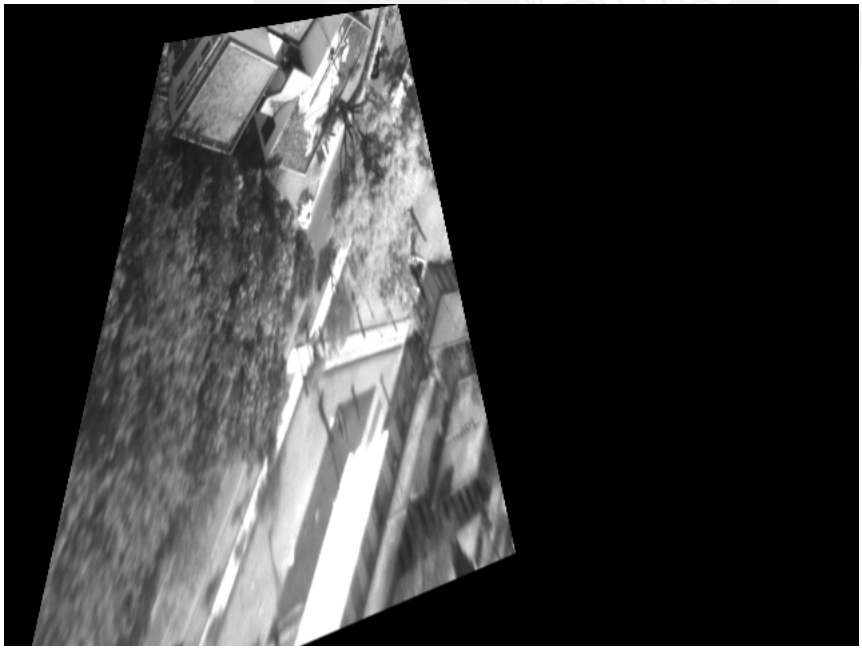


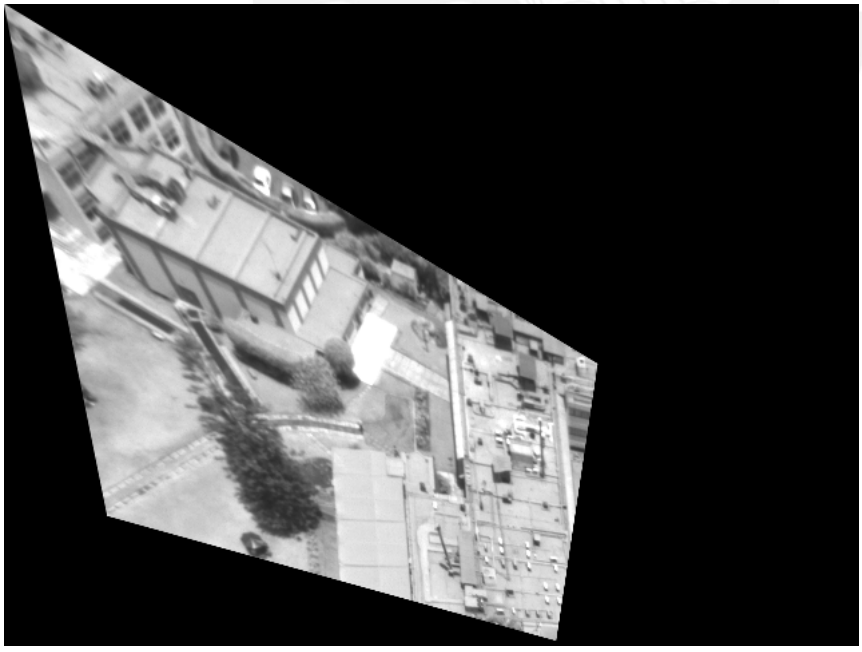
Imagen rectificada



r: 64 p: -7 y:0

Imagen 49

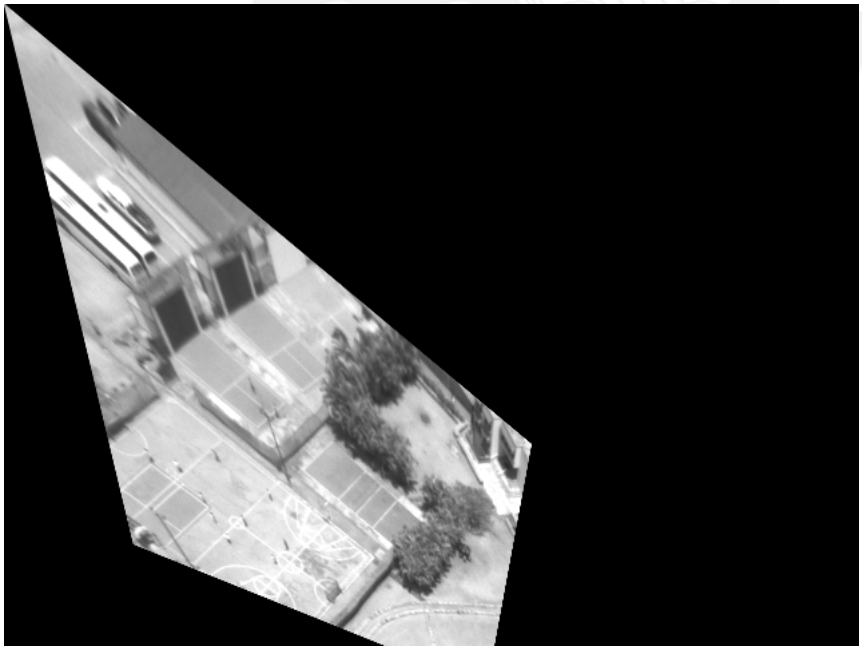
Imagen rectificada



r: -35 p: -37 y:0

Imagen 50

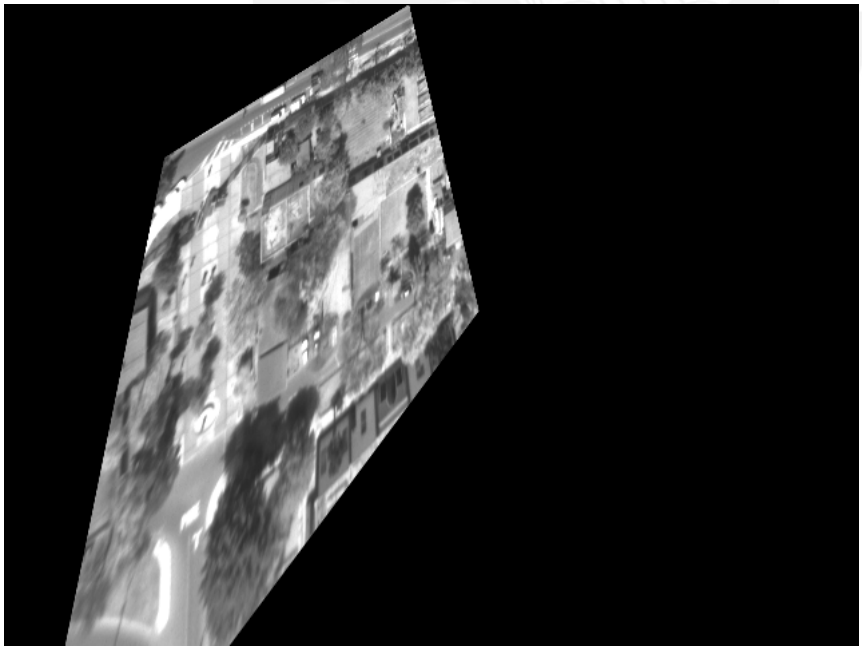
Imagen rectificada



r: -44 p: -37 y:0

Imagen 51

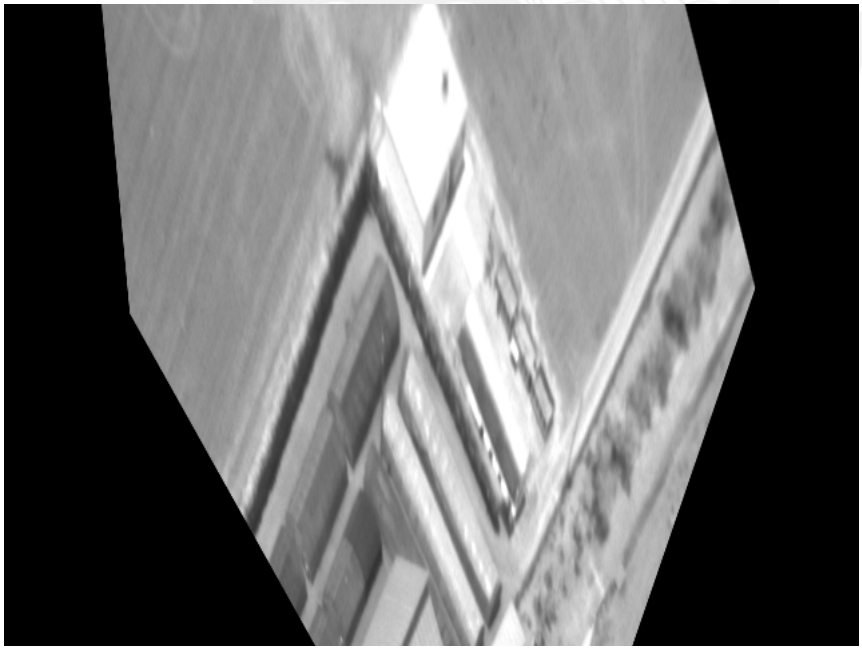
Imagen rectificada



r: 61 p: - 29 y:0

Imagen 52

Imagen rectificada



r: -73 p: -38 y:0

Imágenes sin necesidad de Rectificar r: 0 p: 0 y:0

Imagen 53

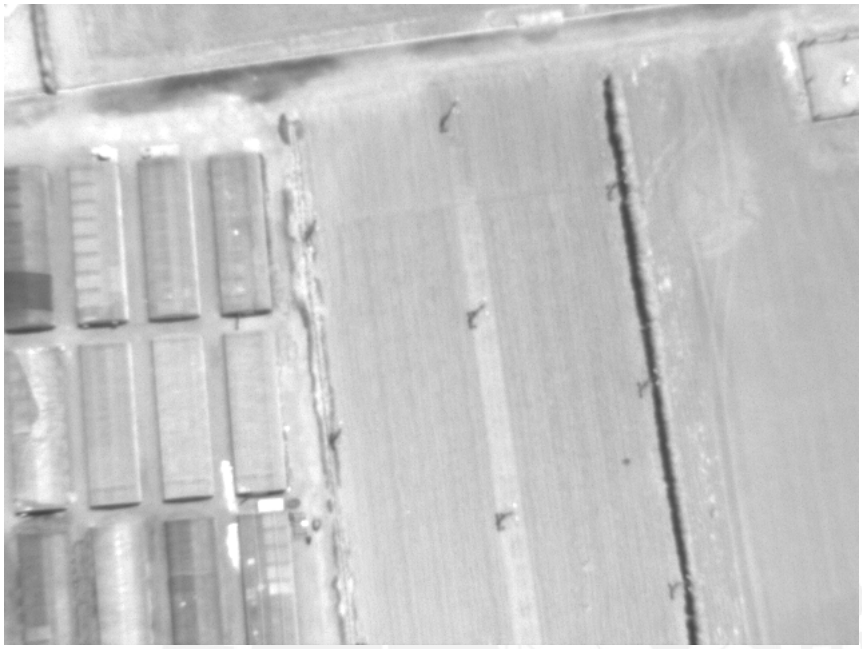


Imagen 54

Imagen 55

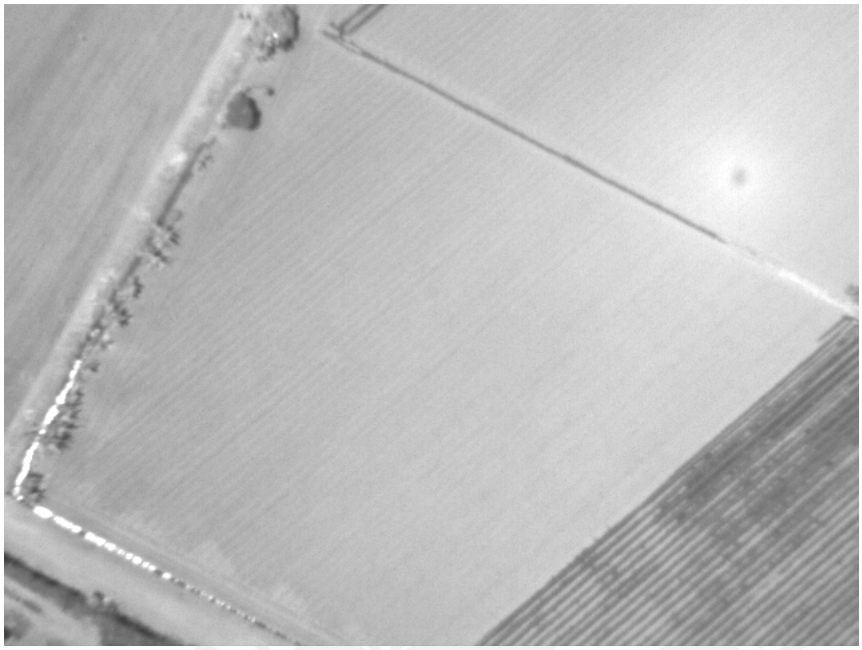


Imagen 56

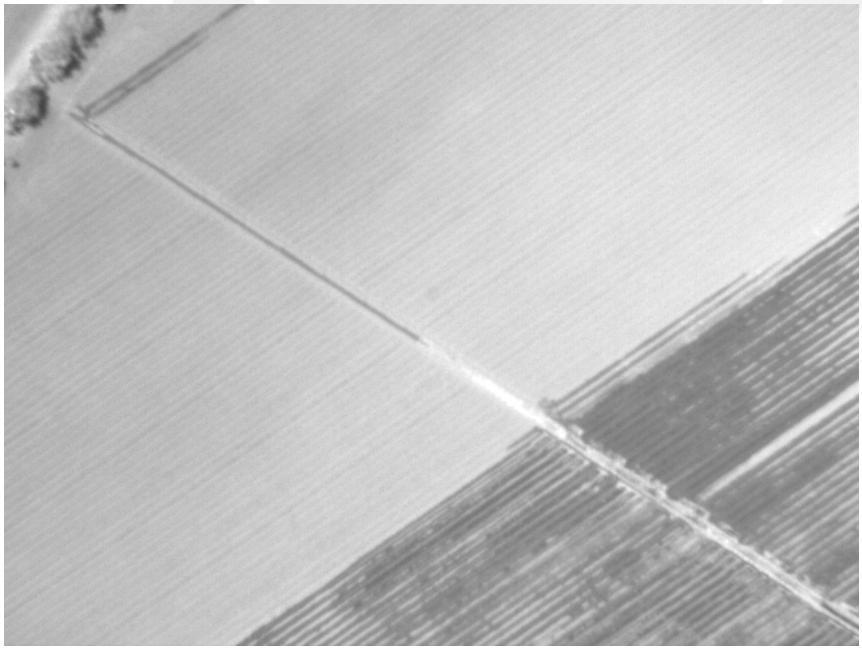


Imagen 57

Imagen 58