COMPUTACION GRAFICA

RECONSTRUCCION DE OBJETO “3D” A PARTIR DE IMÁGENES CALIBRADAS

ESCUELA:

-INGENIERIA DE SISTEMAS-

INTEGRANTES:

- ALVARADO FIGUEROA, KENNY
- BROUSSET LOPEZ, WENDY
- MACEDO VALENCIA, JONATHAN

AREQUIPA – PERU

2018

RECONSTRUCCIÓN DE OBJETO 3D A PARTIR DE IMÁGENES CALIBRADAS

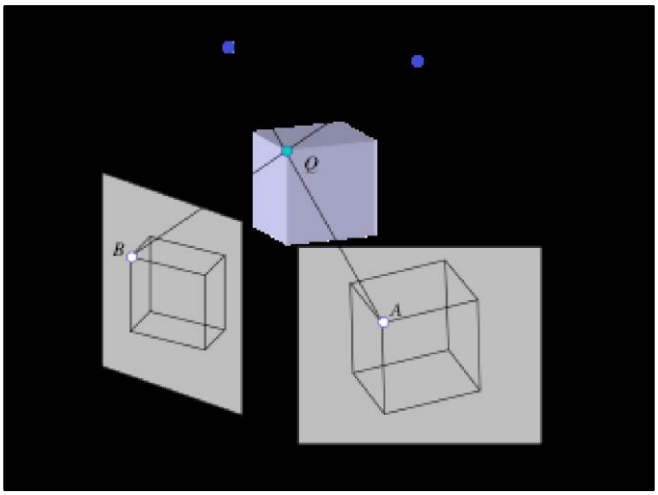
I. Resumen:

Este trabajo presenta el desarrollo de un sistema de reconstrucción de objeto 3D, a partir de una colección de vistas. El sistema se compone de dos módulos principales. El primero realiza el procesamiento de imagen, cuyo objetivo es determinar el mapa de profundidad en un par de vistas, donde cada par de vistas sucesivas sigue una secuencia de fases: detección de puntos de interés, correspondencia de puntos y reconstrucción de puntos; en el proceso de reconstrucción se determinan los parámetros que describen el movimiento (matriz de rotación R y el vector de traslación T) entre las dos vistas. Esta secuencia de pasos se repite para todos los pares de vistas sucesivas del conjunto.

El segundo módulo tiene como objetivo crear el modelo 3D del objeto, para lo cual debe determinar el mapa total de todos los puntos 3D generados; en cada iteración del módulo anterior, una vez obtenido el mapa de profundidad total, genera la malla 3D, aplicando el método de triangulación de Delaunay.

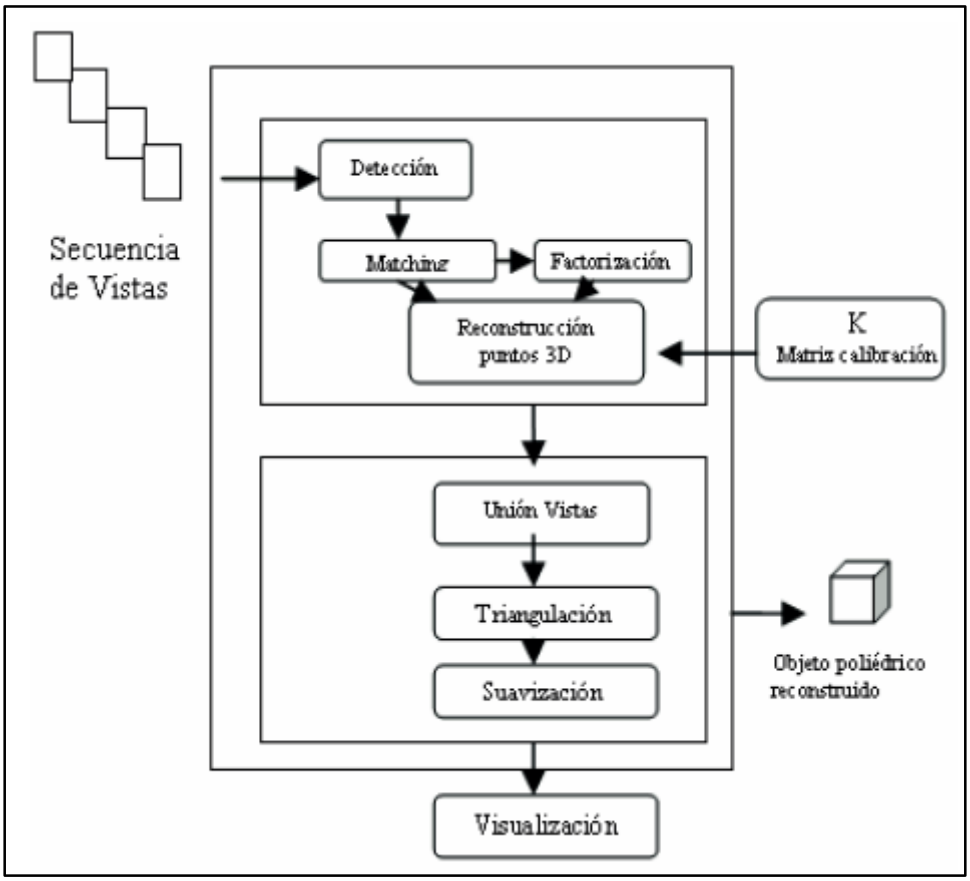
II. Introducción

La reconstrucción 3D es el proceso mediante el cual objetos reales son reproducidos en la memoria de un computador, manteniendo sus características físicas (dimensiones, volumen y forma). Existen diversas técnicas de reconstrucción y métodos de malla 3D, cuyo objetivo principal es obtener un algoritmo que sea capaz de realizar la conexión del conjunto de puntos representativos del objeto en forma de elementos de superficie, ya sean triángulos, cuadrados o cualquier otra forma geométrica.



III. ARQUITECTURA DEL SISTEMA DE RECONSTRUCCIÓN

La entrada del sistema está compuesta de una secuencia de vistas *I(1,2)*, *I(2,3),I(3,4),I(4,5),…. I(i,i+1),……,I(n -1,n),I(n,1)*, las cuales son procesadas de vistas sucesivas. La otra entrada la compone la matriz de calibración (*K*), que está compuesta por los parámetros intrínsecos de la cámara determinada. El sistema está compuesto por dos módulos principales, el primero realiza el procesamiento de dos vistas y el segundo determina el modelo 3D incorporando los puntos de los pares de vistas que no han sido considerados (ver figura 1).



El primer módulo tiene como entrada un par de vistas y la salida son los puntos 3D reconstruidos. El módulo realiza un proceso iterativo, donde primero detecta los puntos característicos de las vistas, utilizando para este efecto el método detección de Harris [18]. La salida de este proceso es el conjunto de puntos detectados, a partir de ellos se estima la matriz de homologación para establecer la correspondencia o matching. La determinación de la correspondencia estará dada por la ventana de búsqueda centrada en los puntos de interés de la vista derecha que maximice el criterio de similitud.

IV. IMPLEMENTACION

```
function [K, R, t] = vgg_KR_from_P(P, noscale)
% This function was used in the C-LAB 6, so we are basicly reusing it since
% it was necessary to uqet the K,R,T values from a
%
%VGG_KR_FROM_P Extract K, R from camera matrix.
%
%    [K,R,t] = VGG_KR_FROM_P(P [,noscale]) finds K, R, t such that P =
K*R*[eye(3) -t].
%    It is det(R)==1.
%    K is scaled so that K(3,3)==1 and K(1,1)>0. Optional parameter noscale
prevents this.
%
%    Works also generally for any P of size N-by-(N+1).
%    Works also for P of size N-by-N, then t is not computed.


% Author: Andrew Fitzgibbon <awf@robots.ox.ac.uk>
% Modified by werner.
% Date: 15 May 98

N = size(P,1);
H = P(:,1:N);

[K,R] = vgg_rq(H);

if nargin < 2
  K = K / K(N,N);
  if K(1,1) < 0
    D = diag([-1 -1 ones(1,N-2)]);
    K = K * D;
    R = D * R;

  %  test = K*R;
  %  vgg_assert0(test/test(1,1) - H/H(1,1), 1e-07)
  end
end

if nargout > 2
  t = -P(:,1:N)\P(:,end);
end

return


% [R,Q] = vgg_rq(S)  Just like qr but the other way around.
%
% If [R,Q] = vgg_rq(X), then R is upper-triangular, Q is orthogonal, and X==R*Q.
% Moreover, if S is a real matrix, then det(Q)>0.

% By awf

function [U,Q] = vgg_rq(S)

S = S';
[Q,U] = qr(S(end:-1:1,end:-1:1));
Q = Q';
Q = Q(end:-1:1,end:-1:1);
U = U';
U = U(end:-1:1,end:-1:1);

if det(Q)<0
  U(:,1) = -U(:,1);
  Q(1,:) = -Q(1,:);
end

return
```

```
function image = read_image(imagenumber)

imageFileNumber = sprintf('%.2d', imagenumber);
imageFile = strcat('data/images/', imageFileNumber, '.jpg');
image = imread(imageFile);

return
```

```
function contour = read_contour(imagenumber)

imageFileNumber = sprintf('%.4d', imagenumber);
imageFile = strcat('data/contour/', imageFileNumber, '.txt');
fp = fileread(imageFile);

%% Parse lines
% match "(float) (float)"
expr = '\d*\.\d* \d*\.\d*';
filetext = regexp(fp, expr, 'match');

%% Convert to numbers
contour = str2num(char(filetext(4:end)));

return
```

```
function [voxels,voxelsKept] = projection_carve( voxels, camera )
% projection_carve( voxels, camera )
%  This function is used to remove all the voxels that are outside of the
%  sillhouete, creating a carving effect.
%
% ARGUMENTS:
% VOXELS = this arguments contains the voxels to be cropped. The amount of
% voxels will reduce as the number of carves is increased.
%
%
% CAMERA = this is a struct that contains all the information of the that
% image. Thus, the sillhouete of the camera can be accessed as
% camera.Sillhouete
%
%
% RETURNS:
% VOXELS = the returned voxels will contain the next voxel matrix with the
% camera sillhouete carve applied onto it
%
% VOXELSKEPT = contain the indeces of the voxel that were kept during the process
% of carving


% Project into image
[x_proj,y_proj] = project_3d_to_2d( camera, voxels.XData, voxels.YData,
voxels.ZData );

%From all the voxels continaed initially, remove all those are not in the
%image
[h,w,d] = size(camera.Image);
voxelsKept = find( (x_proj>=1) & (x_proj<=w) & (y_proj>=1) & (y_proj<=h) );

%update the x and y values removing the values not necessary.
x_proj = x_proj(voxelsKept);
y_proj = y_proj(voxelsKept);

% this is the proper carving, this create a matrix of indices that will
% refer to voxels that will be removed and kept.
ind = sub2ind( [h,w], round(y_proj), round(x_proj) );
voxelsKept = voxelsKept(camera.Silhouette(ind) == 1);

%update the voxels with the new carved voxels
```

```
voxels.XData = voxels.XData(voxelsKept);
voxels.YData = voxels.YData(voxelsKept);
voxels.ZData = voxels.ZData(voxelsKept);
voxels.Value = voxels.Value(voxelsKept);
```

```
function [X_2D, Y_2D] = project_3d_to_2d( camera, X_3D, Y_3D, Z_3D )
%% project_3d_to_2d: project a 3d-point into a 2d-point.
% ARGUMENTS:
%
%     CAMERA = Camera information. Needs only the matrix P, that contains
%     the rotation and the translation of the camera in relation to the
%     center point coordinates.
%     X_3D = x coordinate on 3-dimensional world
%     Y_3D = y coordinate on 3-dimensional world
%     Z_3D = z coordinate on 3-dimensional world
%
% RETURNS:
%     X_2D = x coordinate on 2-dimensional image
%     Y_2D = y coordinate on 2-dimensional image

P = camera.P;

k = P(3,1) * X_3D + P(3,2) * Y_3D + P(3,3) * Z_3D + P(3,4);
X_2D = round( (P(1,1) * X_3D + P(1,2) * Y_3D + P(1,3) * Z_3D + P(1,4)) ./ k);
Y_2D = round( (P(2,1) * X_3D + P(2,2) * Y_3D + P(2,3) * Z_3D + P(2,4)) ./ k);
```

```
% Eduardo Neiva - u5257353
% Luiz Moreira - u5250345
% Matheus Portela - u5250333
% Thiago Oliveira - u5249937
% ENGN4528 - COMPUTER VISION PROJECT
% PROJECT 1

resolution = 500000;
addpath('../data/contour')
addpath('../data/images')
%% Setup
%It is loaded all the functions and the imagepath
datadir = fullfile( fileparts( mfilename( 'fullpath' ) ), 'data' );
close all;


%% Load the Camera and Image Data
% This reads the "/data/images" directory, loading the camera definitions
% (internal and external calibration) and image file for each camera.
cameras = load_cameras( datadir );




%% Generate the Silhouttes
% Generate the silhouettes based on the points provided as contour
% Some image dilation and erosion are used to conect the points.
SI = generate_silhouettes();
for c=1:numel(cameras)
    cameras(c).Silhouette = SI(c).si;
end



%% Work out the space occupied by the scene
%In this part, we define a standard volume that the voxels will be placed
%at, in other words, this is the space xyz that the intial voxel structure
%will be placed so it can be carved. Increasing this world we have a
%greater volume coverage but each voxel will be more spread which would
%result in a 3D model with less resolution.
test = 1000;

x_boundaries = [120, 600];
y_boundaries = [120, 550];
```

```
z_boundaries = [-100, 550];

%% Create a Voxel Array
% in this part, we create a voxel 3d mapping within those limits defined before.
% the forth argument indicates how many voxels there will be in each
% component xyz, thereby increasing this number reduces the spread between
% each voxel and increases the accuracy of the model. This comes at a cost
% of a increased run time and memory usage.
voxels = generate_voxels( x_boundaries, y_boundaries, z_boundaries, resolution );



%% Carving
% Now, using every camera position, we carve the voxels using their
% sillhuete.
for ii=1:numel(cameras)
    voxels = projection_carve( voxels, cameras(ii) );
end
figure();
build_model(voxels);
grid on;
title( '3D Morpheus Reconstruction Model' )
```

```
function cameras = load_cameras(data_directory)
% load_cameras(data_directory): Generate a camera struct
%   which contains all important data: the image itself, calibration
%   matrices and the silhouette image
%
%
% ARGUMENTS:
% data_directory = Directory which contains the necessary data
%
% RETURNS:
% CAMERAS = Struct with the camera data
%


num_cameras = 23;

cameras = struct();

% Loading calibration matrices P
P_matrices = load(fullfile(data_directory, 'morp.mat'));

% Load all structs
for ii = 1:num_cameras
    % Get the image file name
    filename = fullfile(data_directory, sprintf('%d.jpg', ii-1));

    % Decompose matrices
    [K, R, t] = vgg_KR_from_P(P_matrices.P{ii});

    % Load data
    cameras(ii).Image = imread(filename);
    cameras(ii).P = P_matrices.P{ii};
    cameras(ii).K = K./K(3,3); % remove the scaling by K(3, 3) as 1
    cameras(ii).R = R;
    cameras(ii).T = -R'*t;
    cameras(ii).Silhouette = []; % empty silhouette
end
```

```
function voxels = generate_voxels(qtd_x, qtd_y, qtd_z, N)
%% generate_voxels - creates a grid for the future carving
% ARGUMENTS:
%     qtd_x = inferior and superior limits of the X coordinate in the 3D
%     world - ex: [inf_x sup_x]
%     qtd_y = inferior and superior limits of the Y coordinate in the 3D
%     world - ex: [inf_x sup_x]
%     qtd z = inferior and superior limits of the Z coordinate in the 3D
```

```
%     world - ex: [inf_x sup_x]
%     N = amount of voxels to be created
%
% RETURNS:
%     VOXELS = grid



volume = (qtd_x(2) - qtd_x(1)) * (qtd_y(2) - qtd_y(1)) * (qtd_z(2) - qtd_z(1));
voxels.Resolution = power( volume/N, 1/3 );
x = qtd_x(1) : voxels.Resolution : qtd_x(2);
y = qtd_y(1) : voxels.Resolution : qtd_y(2);
z = qtd_z(1) : voxels.Resolution : qtd_z(2);


[X,Y,Z] = meshgrid( x, y, z );
voxels.XData = X(:);
voxels.YData = Y(:);
voxels.ZData = Z(:);
voxels.Value = ones(numel(X),1);
```

```
function SI = generate_silhouettes()
%% generate silhouettes
    import spacecarving.*;

    for j=0:23
        image = read_image(j);
        contour = read_contour(j);

        A = zeros(size(image,1),size(image,2));
        for i = 1: size(contour, 1)
            A(uint32(contour(i,2)), uint32(contour(i,1))) = 1;
        end

        se1= [0 0 0 1 0 0 0; 0 1 1 1 1 1 0; 0 1 1 1 1 1 0; 1 1 1 1 1 1 1; 0 1 1 1
1 1 0; 0 1 1 1 1 1 0;  0 0 0 1 0 0 0];
        se2= [0 1 0; 1 1 1; 0 1 0];

        A = imdilate(A, se1);
        A = imdilate(A, se1);
        A = imerode(A, se2);
        A = imdilate(A, se1);
        A = imerode(A, se2);
        A = imdilate(A, se1);
        A = imerode(A, se2);
        A = imdilate(A, se1);
        A = imerode(A, se2);

        B = imfill(A);

        SI(j+1).si = B;

    end

%     %% TEST - vizualize silhouettes
%     for i = 1:24
%         image = read_image(i-1);
%         aux = [rgb2gray(image) SI(i).si*255];
%         imshow(aux);
%         pause(0.05);
% %         figure
%         imshow(aux)
%     end
end
```

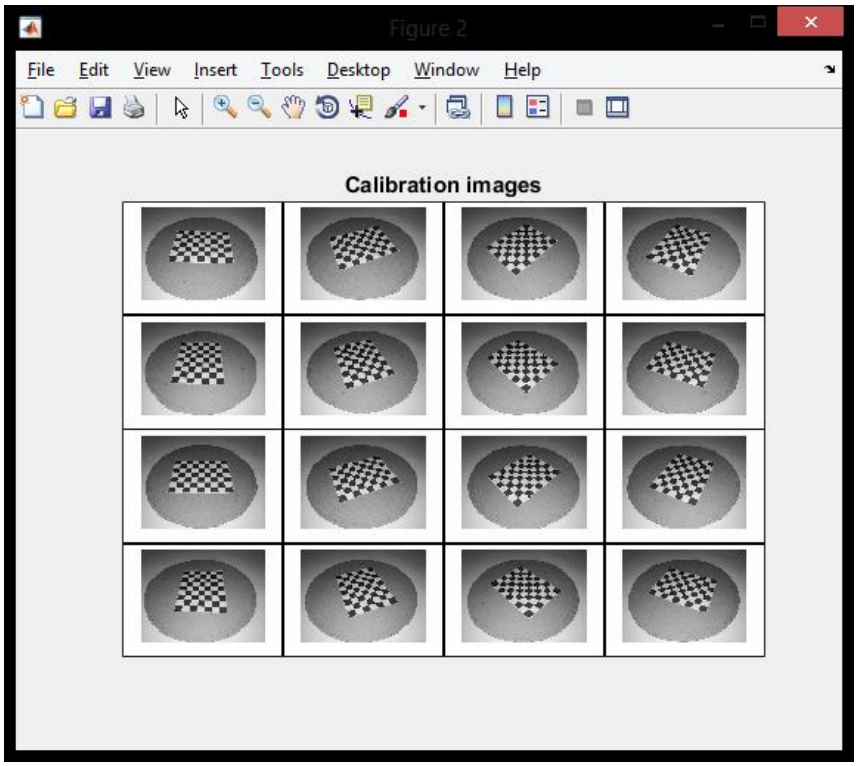
```
function ptch = build_model( voxels )
% build_model( voxels )
%  This function is used to draw a surface based on the provided voxels
%  structure. It uses Matlab's ISOSURFACE command to do that.
%
```

```
% ARGUMENTS:
% VOXELS = this argument contains the voxels to be drawn.
%
% RETURNS:
% PTCH = SHOWSURFACE(VOXELS) also returns handles to the patches created.
%

% First, puts the data in a grid
data_x = unique(voxels.XData);
data_y = unique(voxels.YData);
data_z = unique(voxels.ZData);

% Then, expands the model in each direction by one step
data_x = [data_x(1)-voxels.Resolution; data_x; data_x(end)+voxels.Resolution];
data_y = [data_y(1)-voxels.Resolution; data_y; data_y(end)+voxels.Resolution];
data_z = [data_z(1)-voxels.Resolution; data_z; data_z(end)+voxels.Resolution];

% Convert to a grid
[X,Y,Z] = meshgrid( data_x, data_y, data_z );

% Create a voxel grid with empty spaces, and fill only those elements
% where voxels are present.
V = zeros( size( X ) );
N = numel( voxels.XData );
for ii=1:N
    ix = (data_x == voxels.XData(ii));
    iy = (data_y == voxels.YData(ii));
    iz = (data_z == voxels.ZData(ii));
    V(iy,ix,iz) = voxels.Value(ii);
end

% Now draw it
ptch = patch( isosurface( X, Y, Z, V, 0.5 ) );
isonormals( X, Y, Z, V, ptch )
set( ptch,'FaceColor', 'black', 'EdgeColor', 'red' );

set(gca,'DataAspectRatio',[1 1 1]);
xlabel('X');
ylabel('Y');
zlabel('Z');
view(130,-25)
lighting( 'gouraud' )
axis( 'tight' )
```

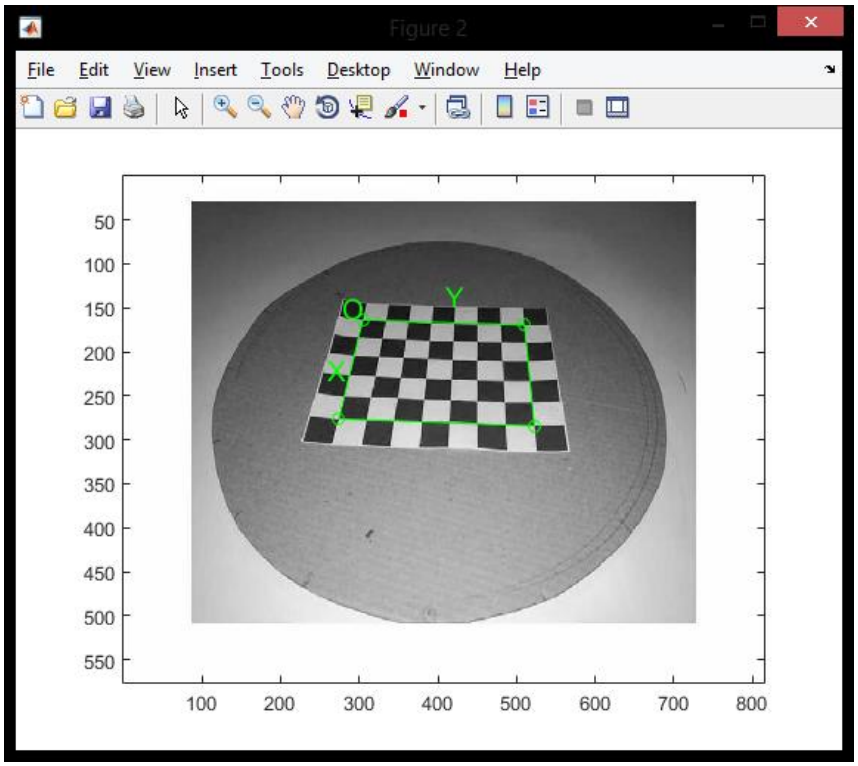
V. RESULTADOS

Para validar el sistema fueron consideradas figuras captadas por un sensor cámara en un ambiente donde el objeto se encuentra sobre una plataforma giratoria con un fondo celeste.

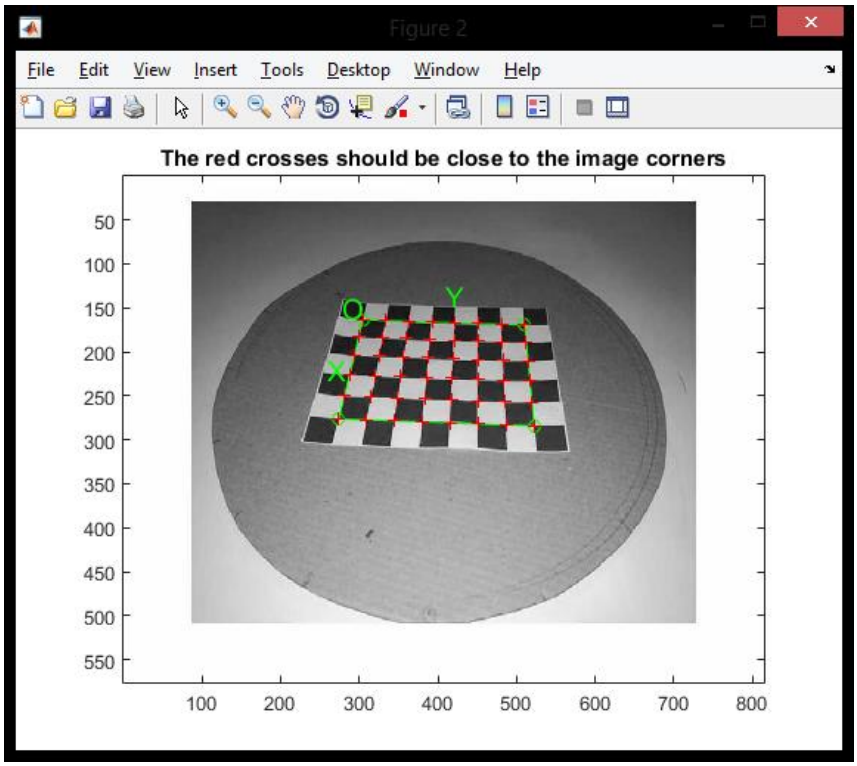
Lectura de las imágenes de calibración de la cámara



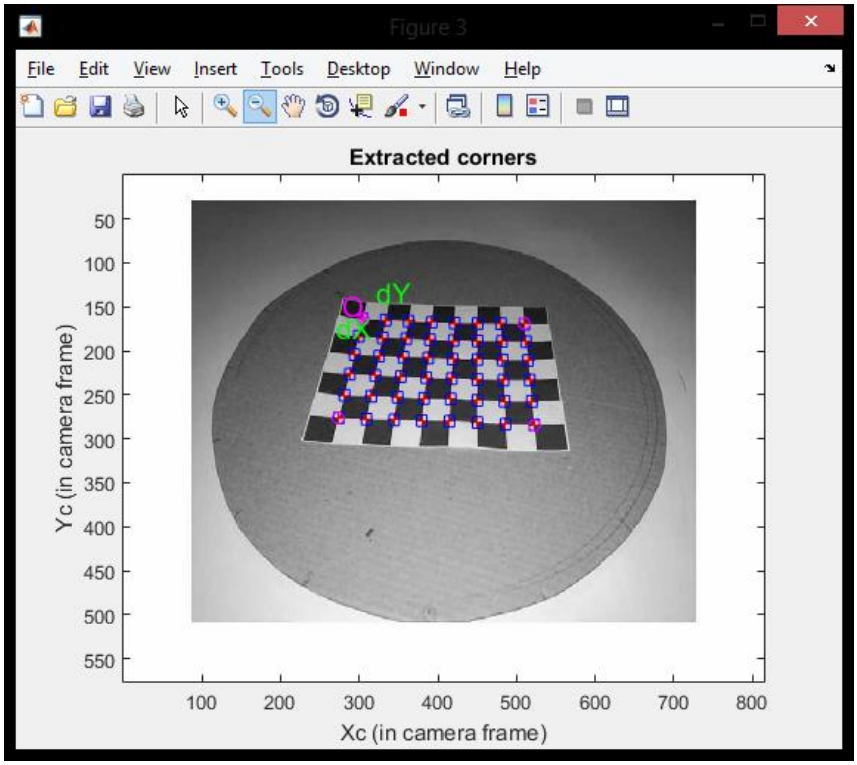
Calibración de las imágenes según el patrón



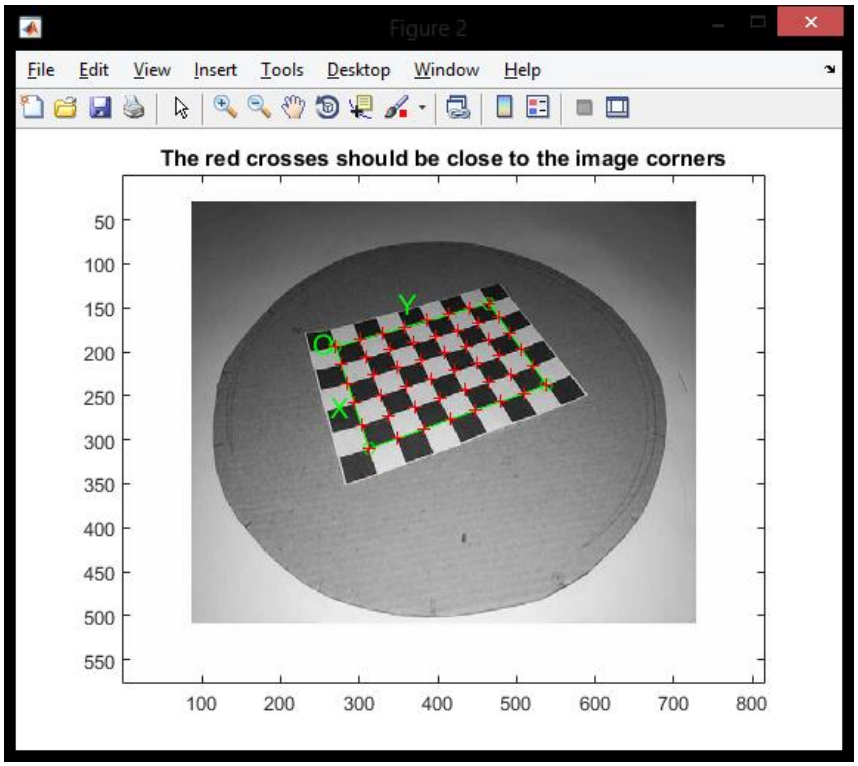
Deteccuion de los Cuadrados de la base de patrón



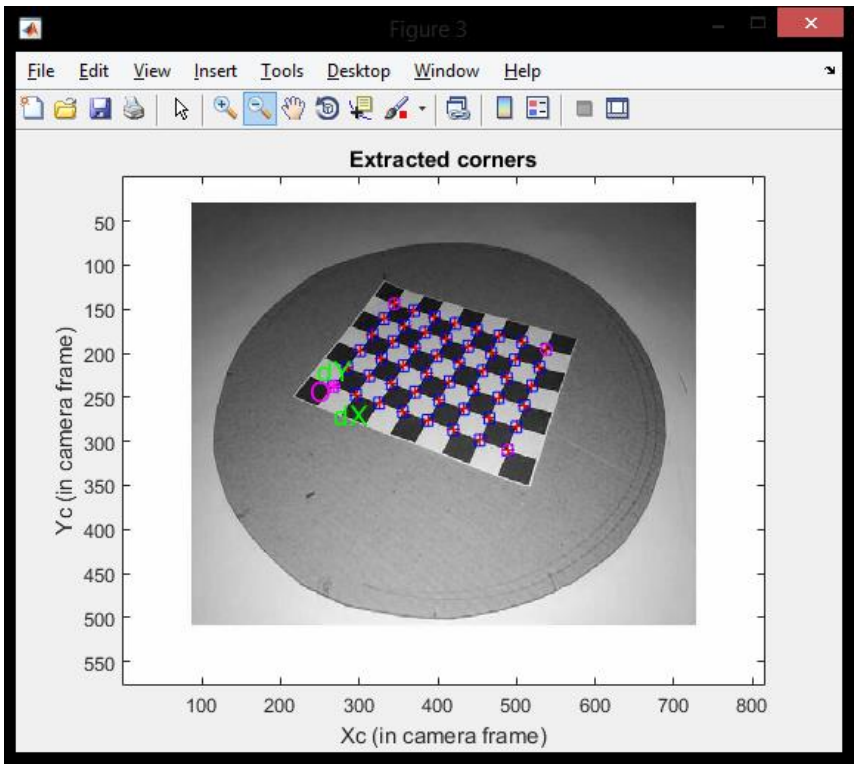
Reconocimietno de las esquinas de los cuadrados dentro del área de escaneo



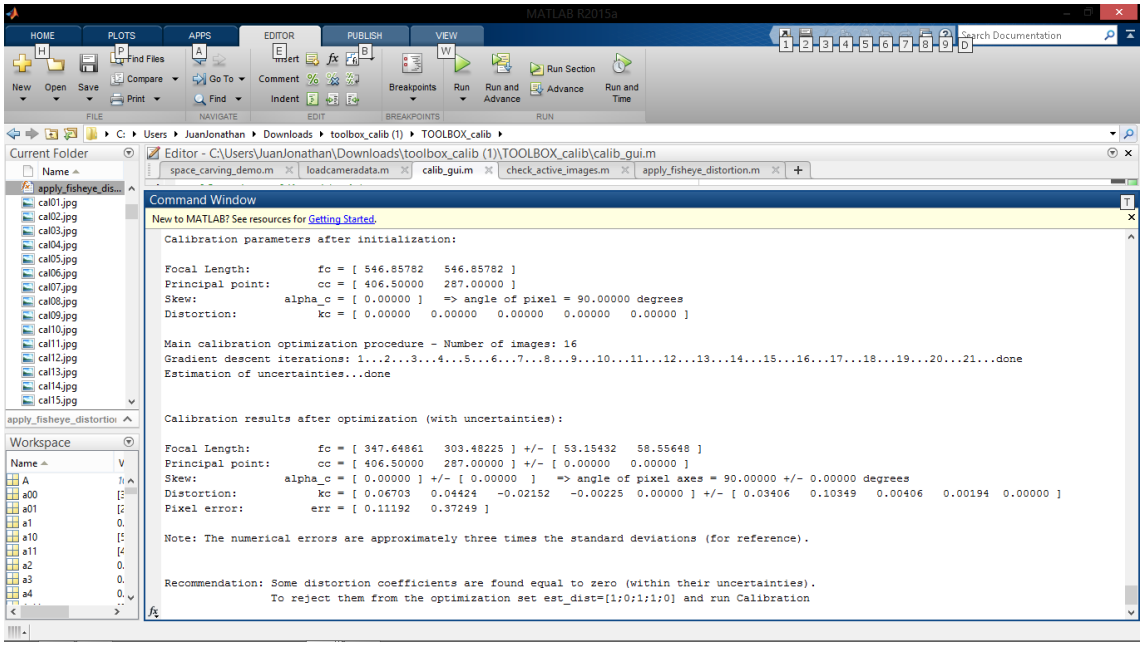
Este procedicmiento se hara en todas las imágenes que se recolectaron para calibrar la cámara



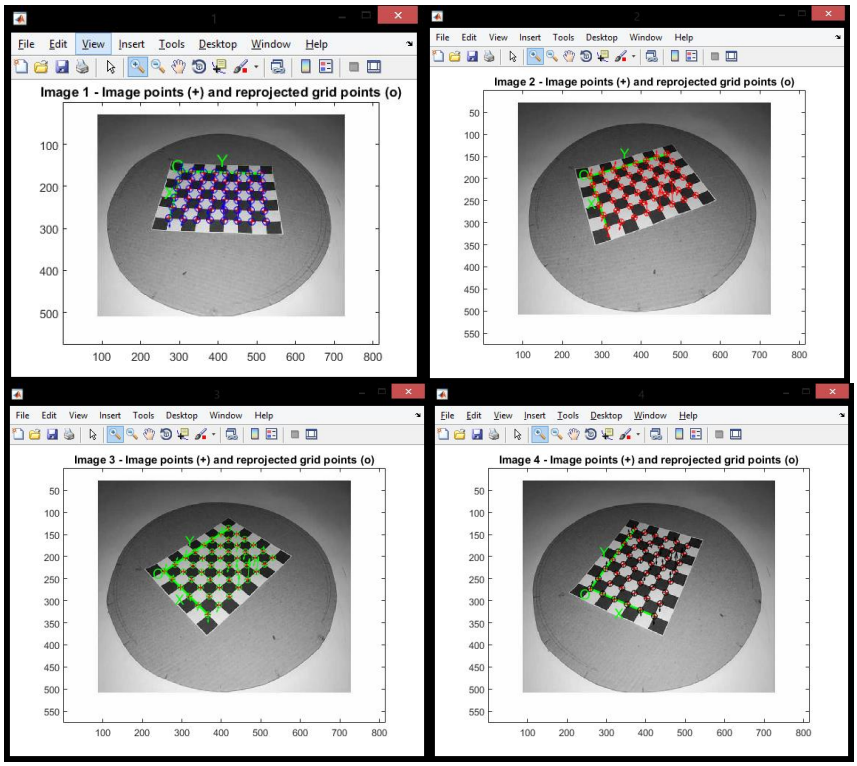
Procesamiento de la ultima imagen



Matlab genera varias variables que serán usadas para nuestra creación de matrices de calibración del sistema



Reproyectamos nuestras claibraciones



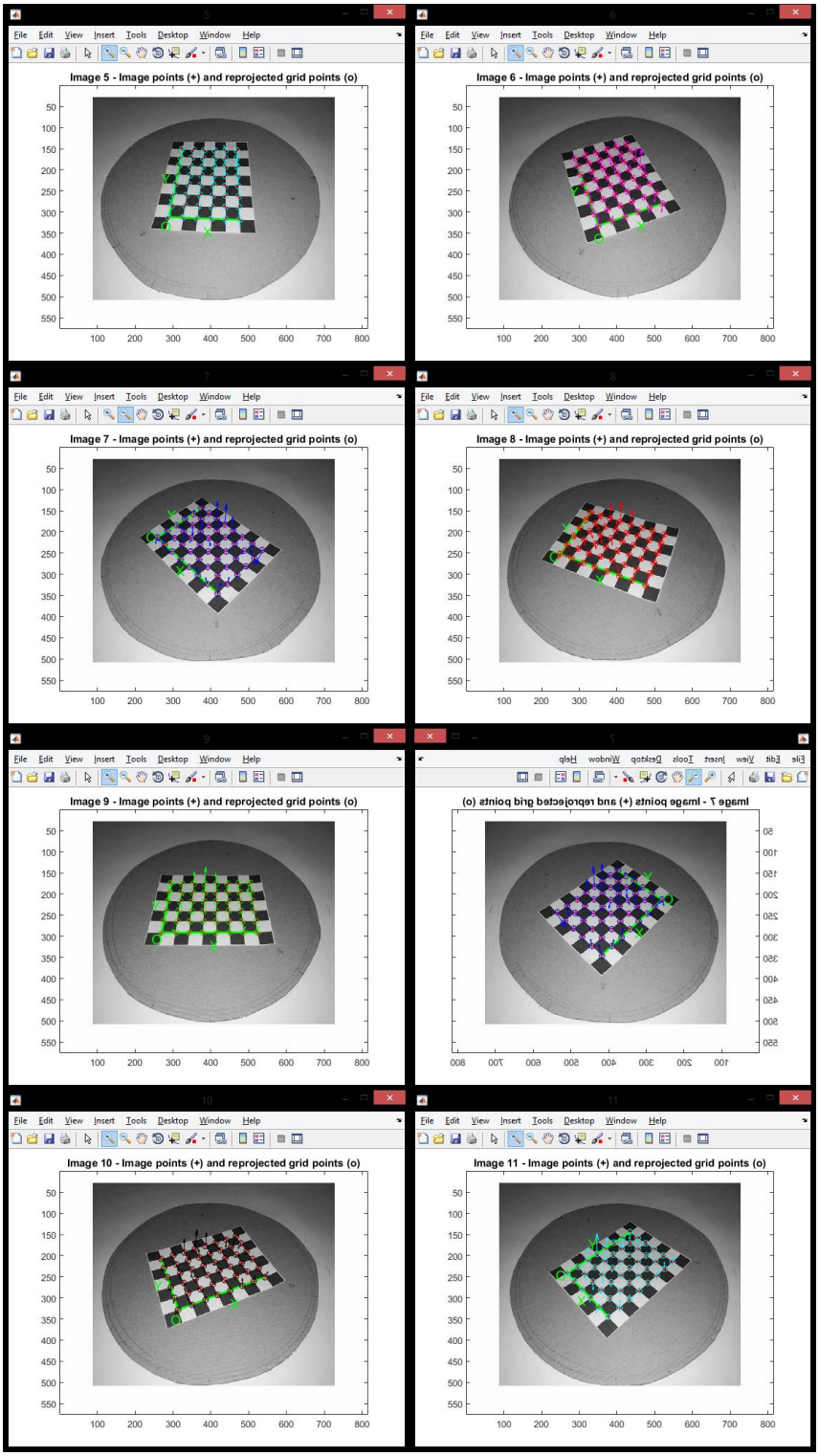
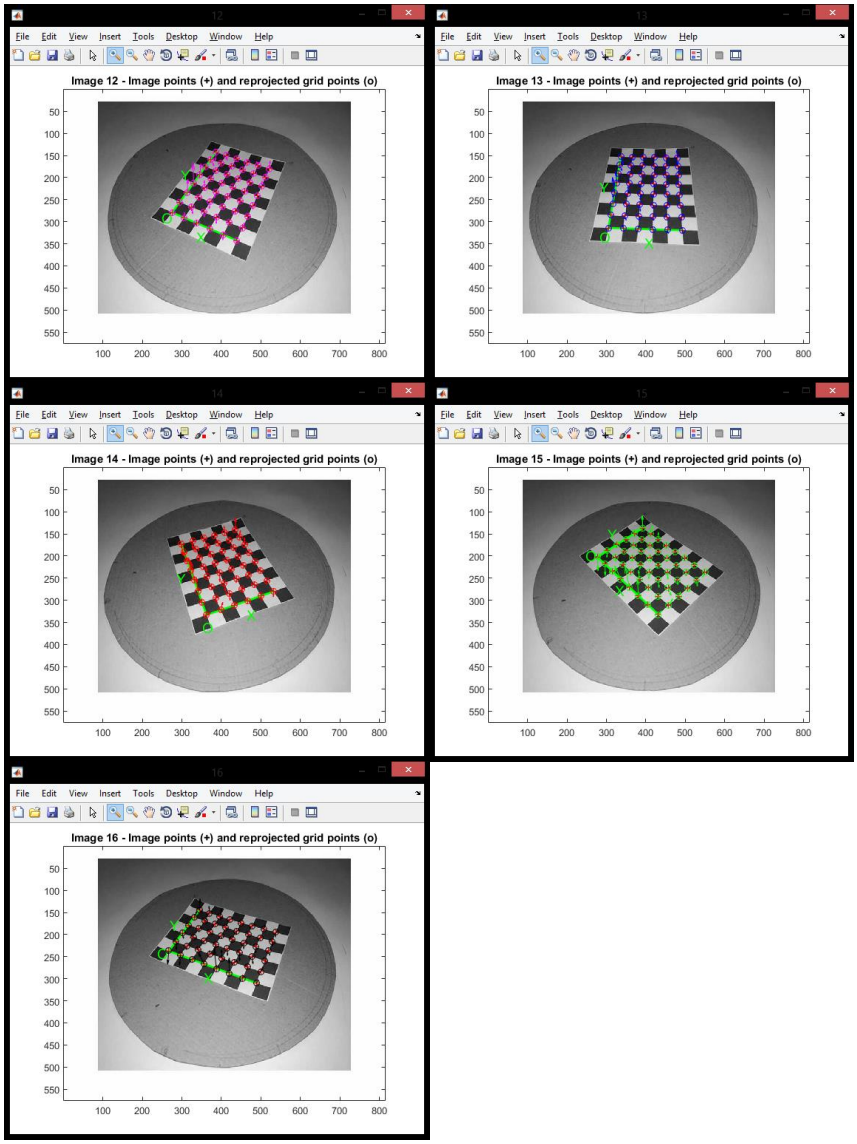
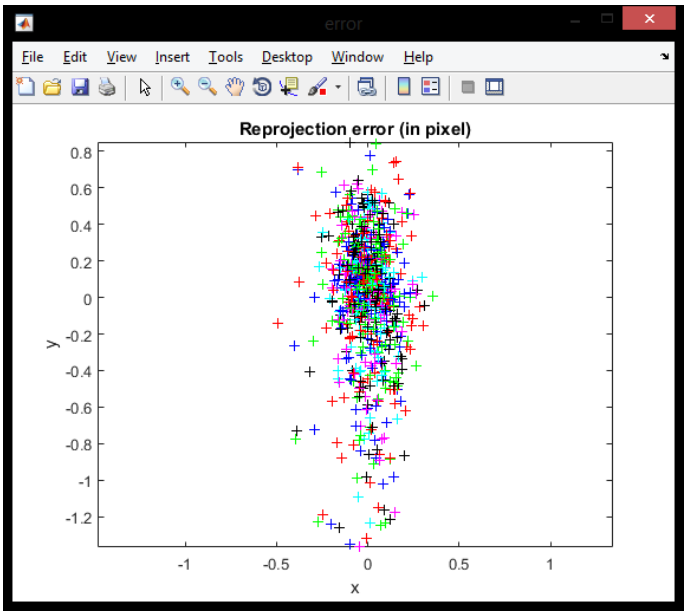


Image 5 - Image points (+) and reprojected grid points (o)
Image 6 - Image points (+) and reprojected grid points (o)
Image 7 - Image points (+) and reprojected grid points (o)
Image 8 - Image points (+) and reprojected grid points (o)
Image 9 - Image points (+) and reprojected grid points (o)
Image 10 - Image points (+) and reprojected grid points (o)
Image 11 - Image points (+) and reprojected grid points (o)



Error de la calibracuin

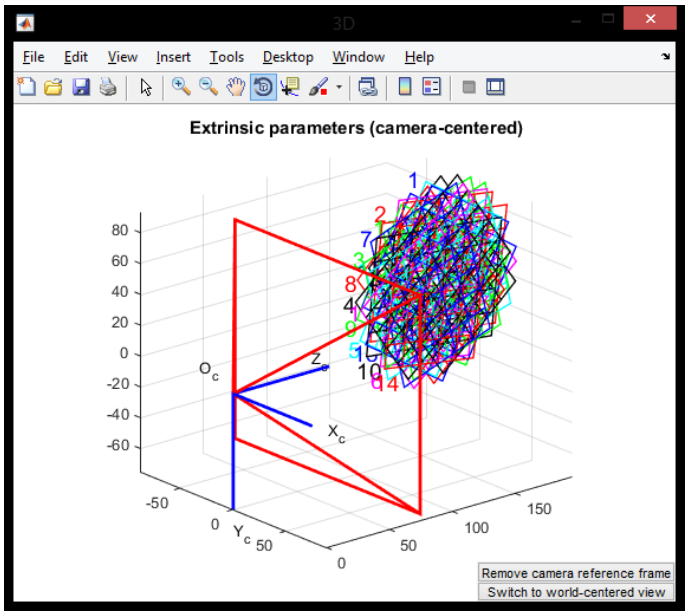


Siendo nuestro error de proyección

Number(s) of image(s) to show ([] = all images) =

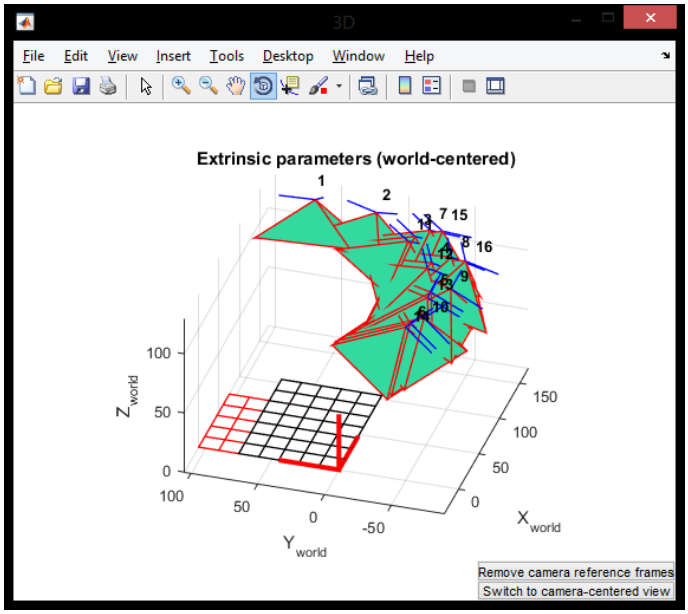
Pixel error: err = [0.11192 0.37249] (all active images)

Los parámetros extrínsecos (posiciones relativas de las cuadrículas con respecto a la cámara) se muestran a continuación en forma de una trama 3D

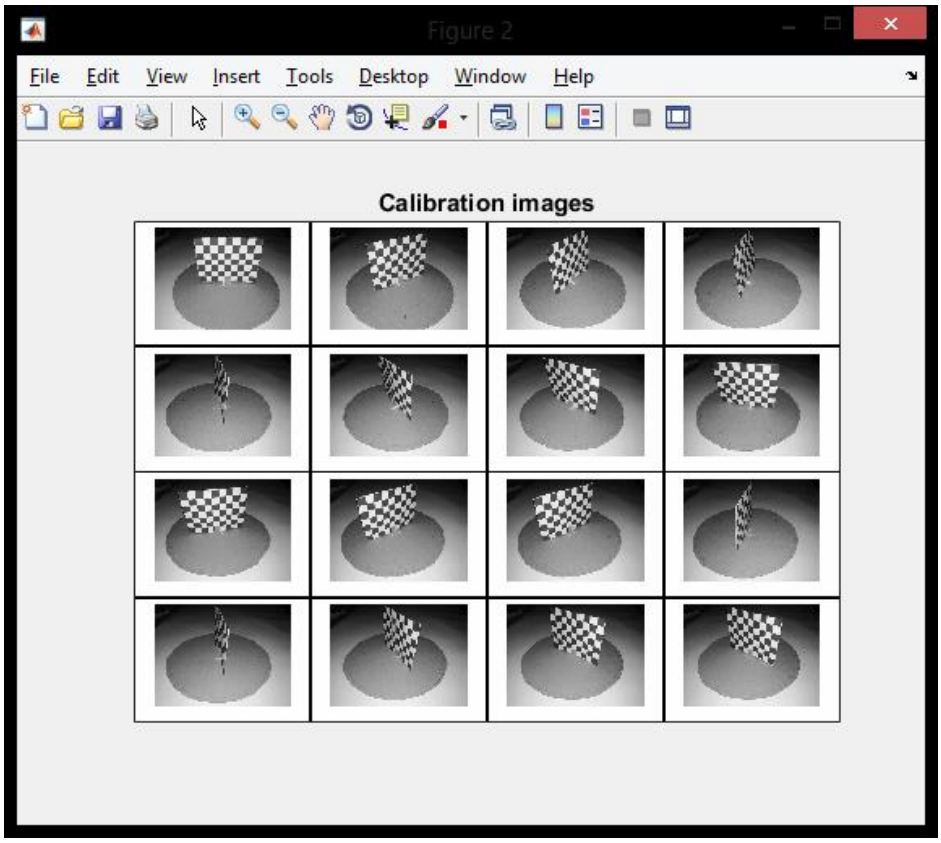


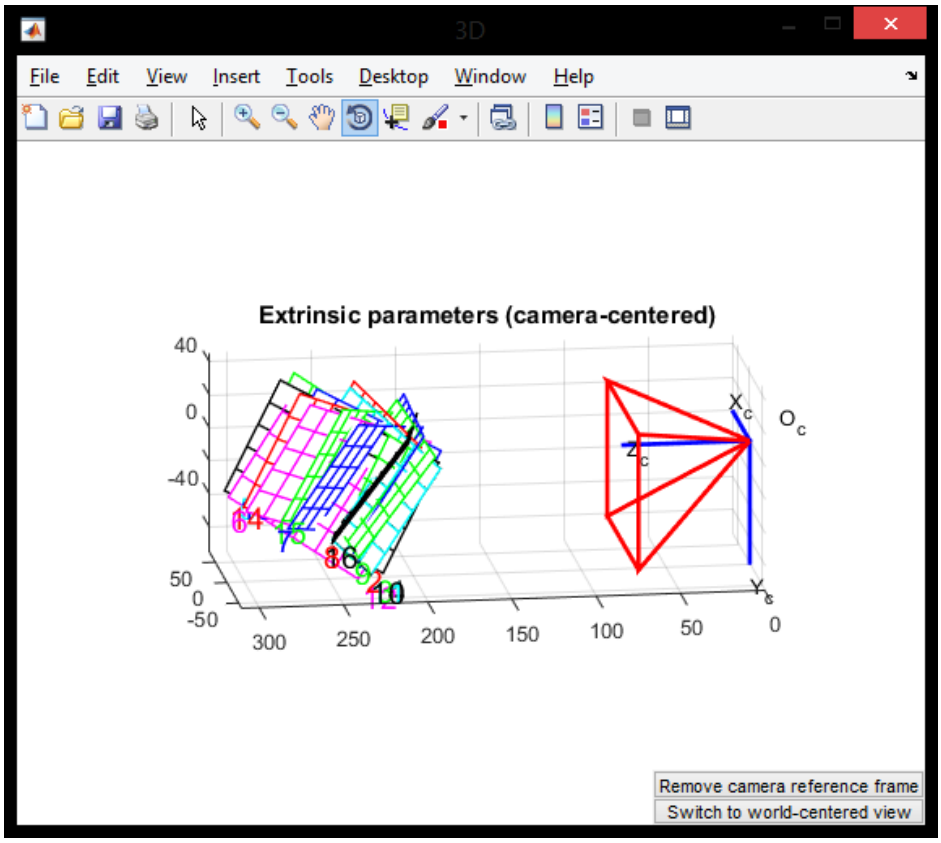
En esta figura, el marco (O c , X c , Y c , Z c) es el marco de referencia de la cámara. La pirámide roja corresponde al campo de visión efectivo de la cámara definida por el plano de la imagen. Para pasar de una vista "centrada en la cámara" a una vista "centrada en el mundo", simplemente haga clic en el botón Cambiar a vista centrada en el mundo que se encuentra en la esquina inferior izquierda de la figura.

Mostramos en si lo que la cámara se centra en el mundo de la proyección



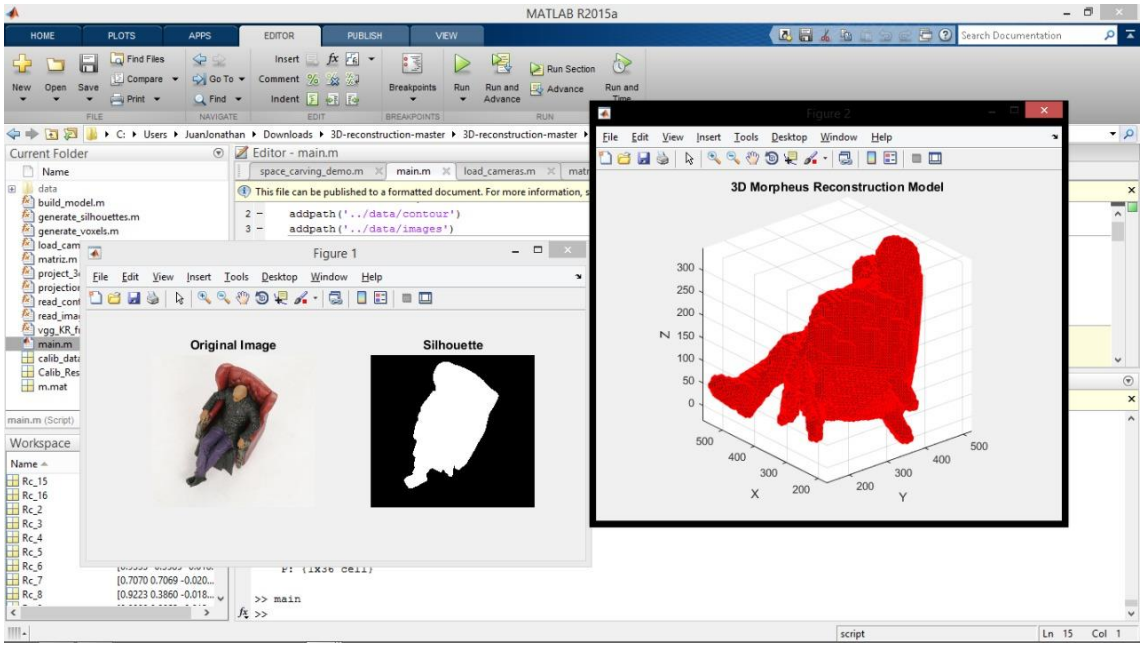
Desde otra perspectiva





Modelado de la imagen:

El modelado de la imagen en MatLab utilizando el space carving utiliza la captura de dos cámaras, utilizando la técnica de visión estereoscópica



VI. CONCLUSION

- Lo que se investigó nos ayudó a entender que la cantidad de puntos característicos detectados influye directamente en la cantidad de puntos 3D reconstruidos y la calidad de la reconstrucción, es decir, mayores detalles del objeto, por lo tanto, una mayor cantidad de puntos en el mapa nos define las áreas con más detalles.