HACETTEPE UNIVERSITY
DEPARTMENT OF
GEOMATICS ENGINEERING

GMT431- PHOTOGRAMMETRIC IMAGE ANALYSIS

Dr. Ali Özgün OK

T.A. Dr. Beste TAVUS

ASSIGNMENT -1

Content

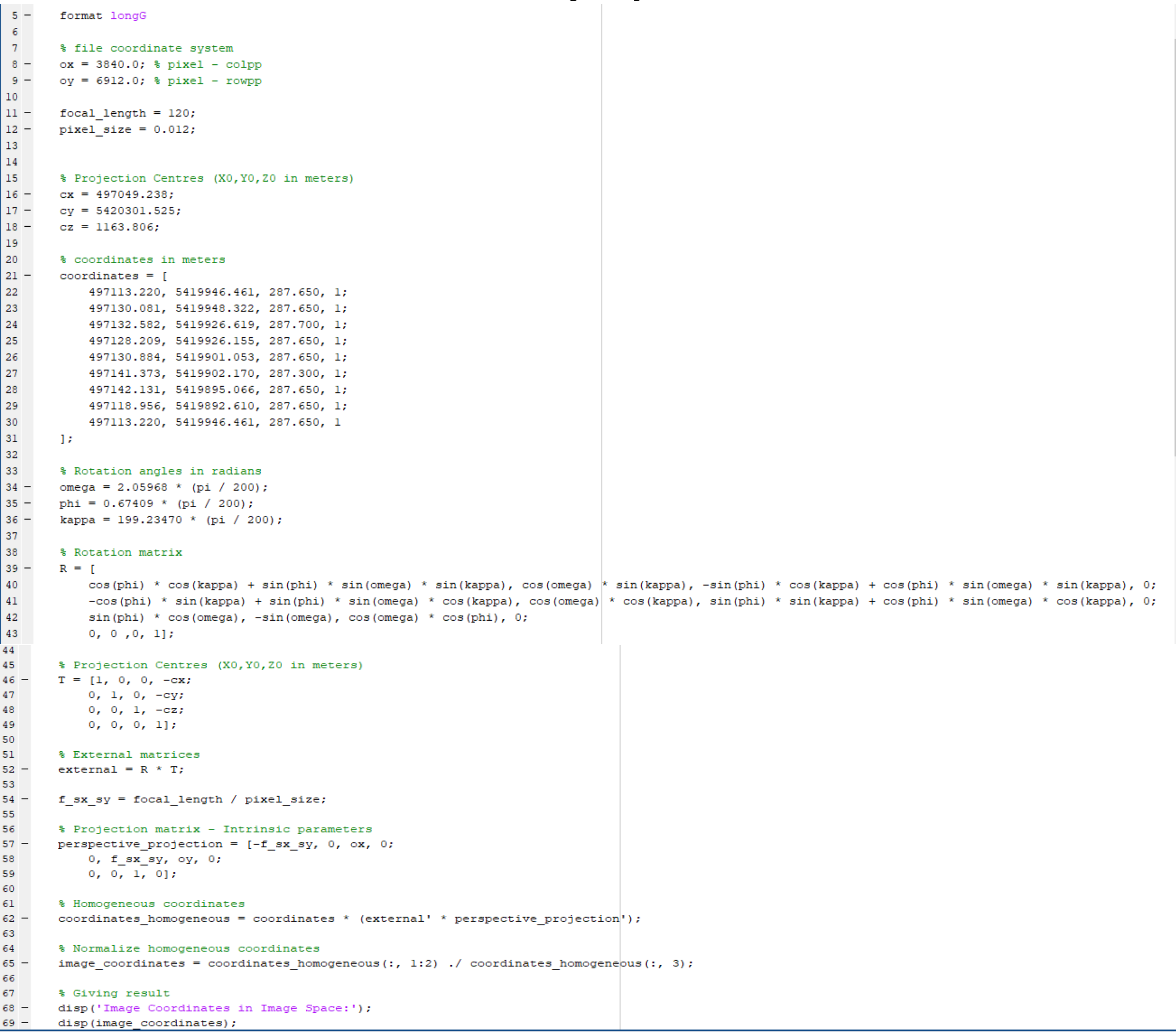
ABDULSAMET TOPTAŞ – 21905024

1- Explanations for Question 1

In this section, utilizing lecture notes, the Object Space to Image Space transformation process was executed through the aid of the Projection Matrix in homogeneous coordinates.

Firstly, the MATLAB programming language has been employed in this project due to its ease in creating mathematical functions, including matrices. The solution for Question 1 has been derived from lecture notes and the information provided in Assignment 1.

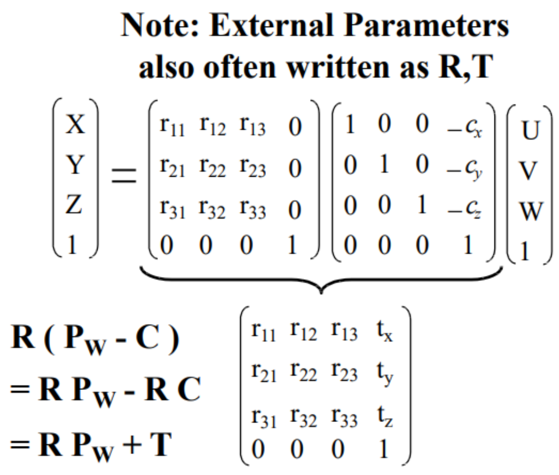
The solution to Question 1 is elucidated in Figure 1 provided below.



```matlab
format longG

% file coordinate system
ox = 3840.0; % pixel - colpp
oy = 6912.0; % pixel - rowpp

focal_length = 120;
pixel_size = 0.012;


% Projection Centres (X0,Y0,Z0 in meters)
cx = 497049.238;
cy = 5420301.525;
cz = 1163.806;

% coordinates in meters
coordinates = [
    497113.220, 5419946.461, 287.650, 1;
    497130.081, 5419948.322, 287.650, 1;
    497132.582, 5419926.619, 287.700, 1;
    497128.209, 5419926.155, 287.650, 1;
    497130.884, 5419901.053, 287.650, 1;
    497141.373, 5419902.170, 287.300, 1;
    497142.131, 5419895.066, 287.650, 1;
    497118.956, 5419892.610, 287.650, 1;
    497113.220, 5419946.461, 287.650, 1
];

% Rotation angles in radians
omega = 2.05968 * (pi / 200);
phi = 0.67409 * (pi / 200);
kappa = 199.23470 * (pi / 200);

% Rotation matrix
R = [
    cos(phi) * cos(kappa) + sin(phi) * sin(omega) * sin(kappa), cos(omega) * sin(kappa), -sin(phi) * cos(kappa) + cos(phi) * sin(omega) * sin(kappa), 0;
    -cos(phi) * sin(kappa) + sin(phi) * sin(omega) * cos(kappa), cos(omega) * cos(kappa), sin(phi) * sin(kappa) + cos(phi) * sin(omega) * cos(kappa), 0;
    sin(phi) * cos(omega), -sin(omega), cos(omega) * cos(phi), 0;
    0, 0 ,0, 1];

% Projection Centres (X0,Y0,Z0 in meters)
T = [1, 0, 0, -cx;
    0, 1, 0, -cy;
    0, 0, 1, -cz;
    0, 0, 0, 1];

% External matrices
external = R * T;

f_sx_sy = focal_length / pixel_size;

% Projection matrix - Intrinsic parameters
perspective_projection = [-f_sx_sy, 0, ox, 0;
    0, f_sx_sy, oy, 0;
    0, 0, 1, 0];

% Homogeneous coordinates
coordinates_homogeneous = coordinates * (external' * perspective_projection');

% Normalize homogeneous coordinates
image_coordinates = coordinates_homogeneous(:, 1:2) ./ coordinates_homogeneous(:, 3);

% Giving result
disp('Image Coordinates in Image Space:');
disp(image_coordinates);
```

Figure 1: MATLAB Code for Question 1 Solution

The necessary parameters for the transition from object space to image space in homogeneous coordinates, as seen in Figure 1, are the rotation matrix and the projection matrix. These matrices were obtained with the help of the lectures and lecture notes covered in Week 4.



Note: External Parameters also often written as R,T

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{pmatrix}} \begin{pmatrix} U \\ V \\ W \\ 1 \end{pmatrix}$$

$$\begin{aligned} &\mathbf{R}\,(\mathbf{P_W} - \mathbf{C}) \\ &= \mathbf{R}\,\mathbf{P_W} - \mathbf{R}\,\mathbf{C} \\ &= \mathbf{R}\,\mathbf{P_W} + \mathbf{T} \end{aligned} \qquad \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

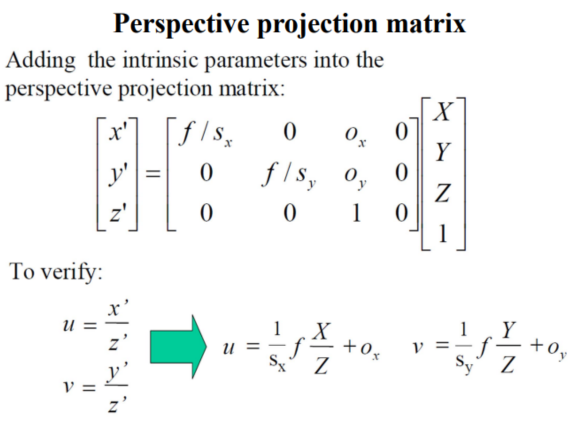
Figure 2: R and T Matrices

The rotation matrix (R) seen in Figure 2 is constructed in lines 39-43. This rotation matrix, along with the angles omega, phi, and kappa to be input into it, is created thanks to Assignment 1. Here, the external orientation parameters omega, phi, and kappa are given in units of grads (gradians). Since it is necessary for these angles to be dimensionless, a transformation is required by multiplying the angles by $\pi/200$, as indicated by the equation $G/400 = R/2\pi$. This transformation is carried out in lines 34-36.

For the creation of another matrix, the projection center matrix (T), the matrix in Figure 2 and the parameters to be entered into it are obtained from the information in Assignment 1.

Subsequently, as seen in line 52 of Figure 1, the two matrices created with the variable named "external" are multiplied, completing the first crucial step for the transition to image space.

Another crucial step is the creation of the perspective projection matrix, as seen in lines 57-59 of Figure 1. For the construction of this matrix, lecture notes, as depicted in Figure 3, have been utilized. Additionally, the internal orientation parameters to be entered into the matrix are obtained from Assignment 1.



Perspective projection matrix

Adding the intrinsic parameters into the perspective projection matrix:

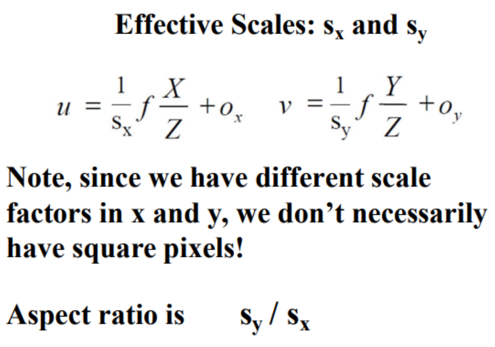
$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} f/s_x & 0 & o_x & 0 \\ 0 & f/s_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

To verify:

$$u = \frac{x'}{z'} \quad v = \frac{y'}{z'} \quad \Rightarrow \quad u = \frac{1}{s_x} f \frac{X}{Z} + o_x \qquad v = \frac{1}{s_y} f \frac{Y}{Z} + o_y$$

Figure 3 : Perspective Projection Matrix

The formula from Figure 3 has been used because an affine transformation was not applied. An important point is that since sx and sy scales are equal to each other (in other words, since they represent a square), in the focal length / pixel size operation seen in lines 57-59 of Figure 1, pixel sizes are divided by the same pixel sizes in both x and y directions. Additionally, since sx and sy scales are equal, the aspect ratio of the scales is 1. This information is referenced from the lecture notes (Figure 4).



Effective Scales: s_x and s_y

$$u = \frac{1}{s_x} f \frac{X}{Z} + o_x \qquad v = \frac{1}{s_y} f \frac{Y}{Z} + o_y$$

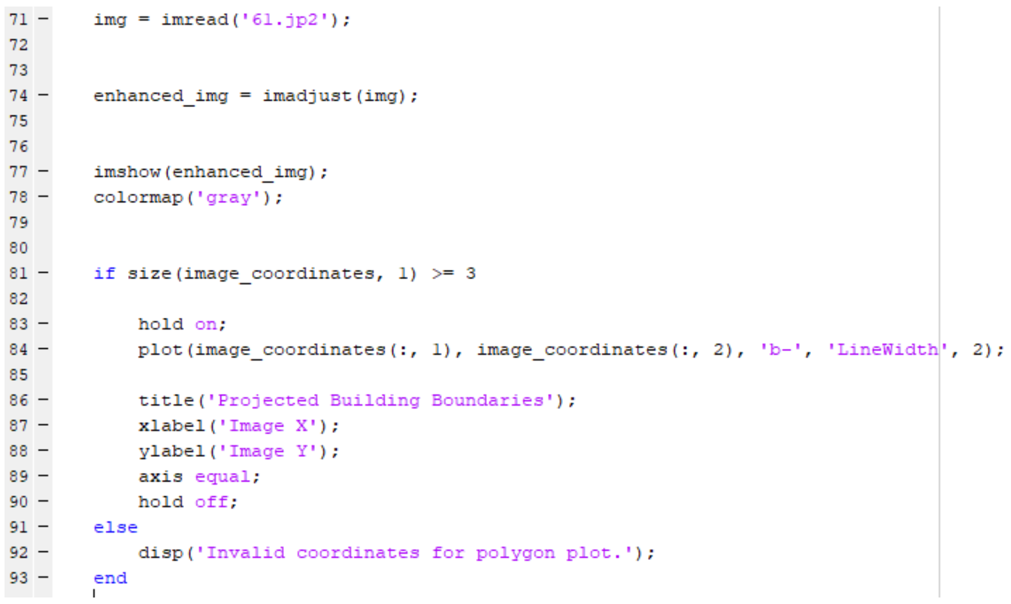
Note, since we have different scale factors in x and y, we don't necessarily have square pixels!

Aspect ratio is s_y / s_x

Figure 4

The reason for performing "- focal length / pixel size" in the perspective projection matrix, as seen in line 57 of Figure 1, is to ensure the correct position and orientation of the building in the x-direction.

It has not been included in the assignment since it was not requested, but for the validation of the accuracy of the solution, the code seen in Figure 5 was added to our own MATLAB code, confirming the accuracy of the position and orientation.



```
img = imread('61.jp2');


enhanced_img = imadjust(img);


imshow(enhanced_img);
colormap('gray');


if size(image_coordinates, 1) >= 3

    hold on;
    plot(image_coordinates(:, 1), image_coordinates(:, 2), 'b-', 'LineWidth', 2);

    title('Projected Building Boundaries');
    xlabel('Image X');
    ylabel('Image Y');
    axis equal;
    hold off;
else
    disp('Invalid coordinates for polygon plot.');
end
```

Figure 5: Accurate detection of building location and orientation

Another crucial aspect in this matrix is the parameters ox and oy. Ox and oy represent the process of shifting the starting point in pixel coordinates from the center of the image (Figure 6). The parameters ox and oy are provided to us as rowpp and colpp in the assignment.

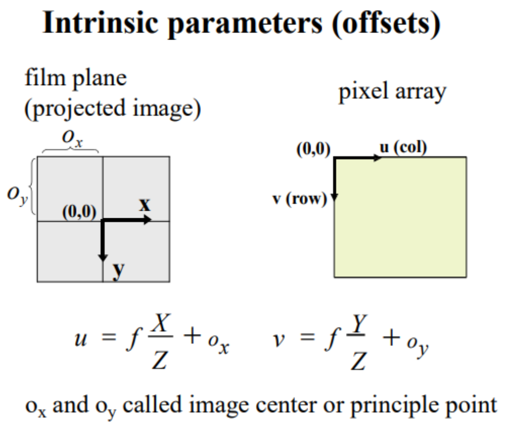

Intrinsic parameters (offsets)
film plane
(projected image)
o_x
o_y
(0,0)
x
y
pixel array
(0,0)
u (col)
v (row)
$u = f\frac{X}{Z} + o_x \quad v = f\frac{Y}{Z} + o_y$
o_x and o_y called image center or principle point

Figure 6

Indeed, as depicted in Figure 6, since the coordinates need to be shifted in the same direction for both ox and oy, there should not be any negative sign applied to these parameters.

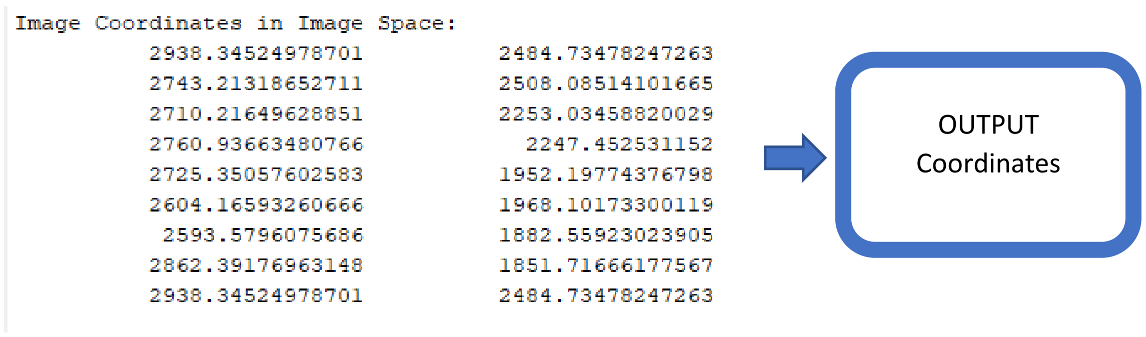
In lines 21-30 of Figure 1, ones are added to the coordinates. The reason for this is to facilitate the multiplication of points with matrices when using homogeneous coordinates. Homogeneous coordinates are particularly useful for the easy application of transformation matrices.

In line 62 of Figure 1, the multiplication operation of the created matrices is performed. First, the transformation matrices (external * perspective_projection) are multiplied. The result of this multiplication is the transformation matrix used to move points from object space to camera space. Then, this transformation matrix is multiplied by the coordinates matrix. This operation is used to move points from object space to camera space.

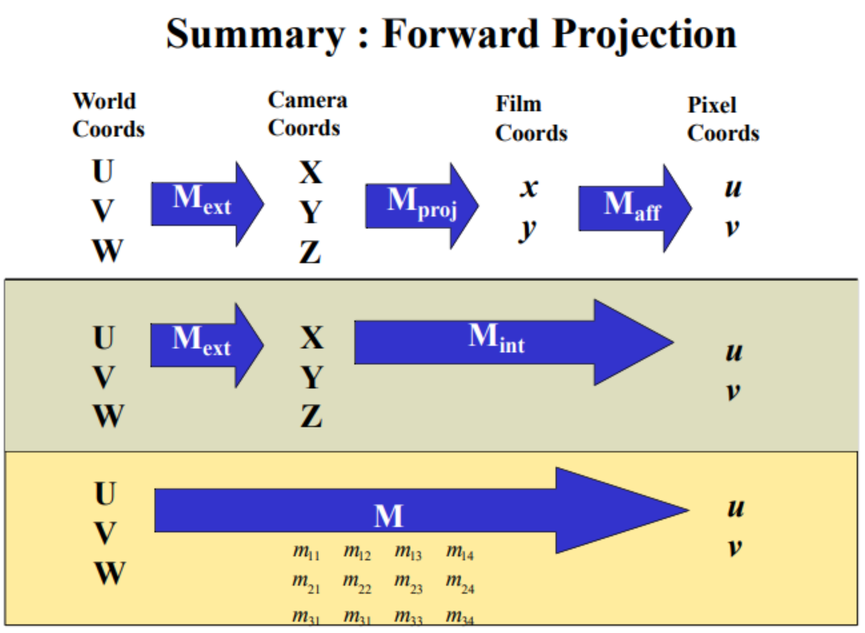
In MATLAB, the apostrophe symbol (') represents taking the transpose. This operation is utilized to ensure proper dimensionality during matrix multiplication and to achieve the correct transformation.

In line 65 of Figure 1, a normalization operation has been performed. Homogeneous coordinates are typically represented by four-element vectors (x, y, z, w), and in their unnormalized form, w is usually 1. However, during transformations, w may have a different value.

Exactly, the normalization process involves bringing these homogeneous coordinates into the 2D plane, ensuring that w becomes equal to 1. As a result, after the normalization process, the 2D coordinates of the building boundary in image space have been obtained.



```
Image Coordinates in Image Space:
          2938.34524978701         2484.73478247263
          2743.21318652711         2508.08514101665
          2710.21649628851         2253.03458820029
          2760.93663480766           2247.452531152
          2725.35057602583         1952.19774376798
          2604.16593260666         1968.10173300119
           2593.5796075686         1882.55923023905
          2862.39176963148         1851.71666177567
          2938.34524978701         2484.73478247263
```


Summary : Forward Projection
World Coords
Camera Coords
Film Coords
Pixel Coords
U V W
M_{ext}
X Y Z
M_{proj}
x y
M_{aff}
u v
U V W
M_{ext}
X Y Z
M_{int}
u v
U V W
M
m_{11} m_{12} m_{13} m_{14}
m_{21} m_{22} m_{23} m_{24}
m_{31} m_{31} m_{33} m_{34}
u v

As a result, there are methods for conversion to pixel coordinates. The method used in this section is Mex → Mint.

Result accuracy test ;

It was mentioned in the report that a plot was added for result accuracy. The code seen in Figure 5 was added and the results were tested. The results provided a logical output.

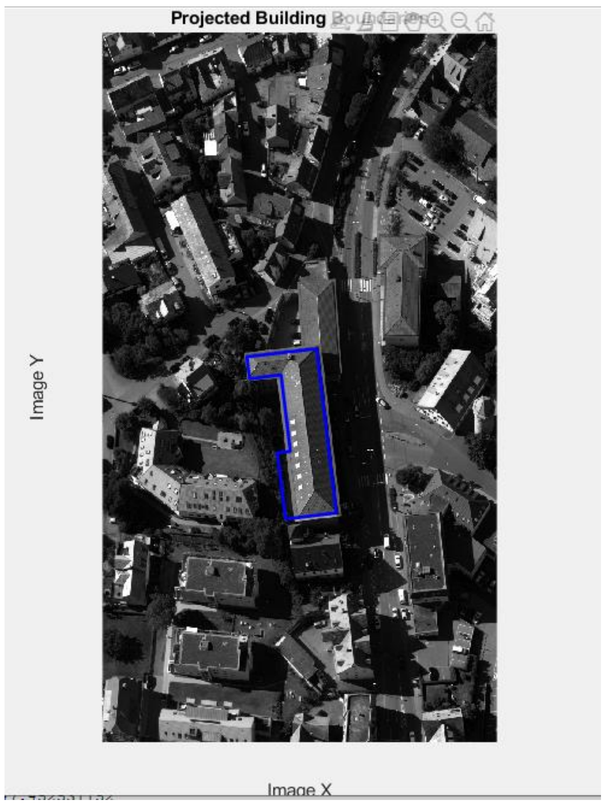
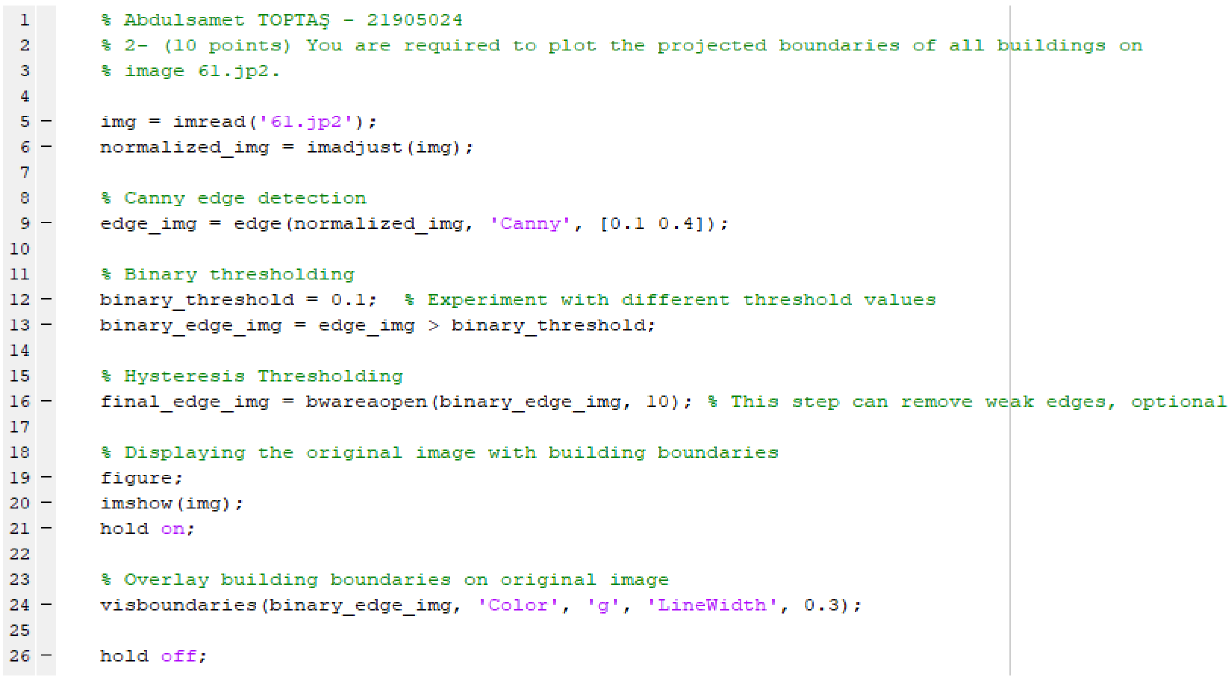


Figure 7 : Projected Building Boundaries

2- Explanations for Question 2

In the second part, the reflected boundaries of the buildings were drawn. You can see the MATLAB code that performs this operation in Figure 8.



```
% Abdulsamet TOPTAŞ - 21905024
% 2- (10 points) You are required to plot the projected boundaries of all buildings on
% image 61.jp2.

img = imread('61.jp2');
normalized_img = imadjust(img);

% Canny edge detection
edge_img = edge(normalized_img, 'Canny', [0.1 0.4]);

% Binary thresholding
binary_threshold = 0.1;  % Experiment with different threshold values
binary_edge_img = edge_img > binary_threshold;

% Hysteresis Thresholding
final_edge_img = bwareaopen(binary_edge_img, 10); % This step can remove weak edges, optional

% Displaying the original image with building boundaries
figure;
imshow(img);
hold on;

% Overlay building boundaries on original image
visboundaries(binary_edge_img, 'Color', 'g', 'LineWidth', 0.3);

hold off;
```

Figure 8: Projected boundaries of all buildings

imread: Reads an image file. It's a part of the MATLAB base functionality and doesn't require a specific toolbox.

imadjust: Adjusts the intensity values of an image. It's part of the Image Processing Toolbox.

edge: Performs edge detection on an image. The 'Canny' option specifies the Canny edge detection algorithm. This function is also part of the Image Processing Toolbox.

bwareaopen: Removes small objects from binary images. This function is part of the Image Processing Toolbox.

imshow: Displays images. It's part of the MATLAB base functionality.

figure: Creates a new figure window for plotting. It's part of the MATLAB base functionality.

hold: Holds the current plot or flushes the current figure. It's part of the MATLAB base functionality.

visboundaries: Visualizes boundaries on a binary image. It's part of the Computer Vision Toolbox.

In summary, the code primarily uses functions from the Image Processing Toolbox and the Computer Vision Toolbox.

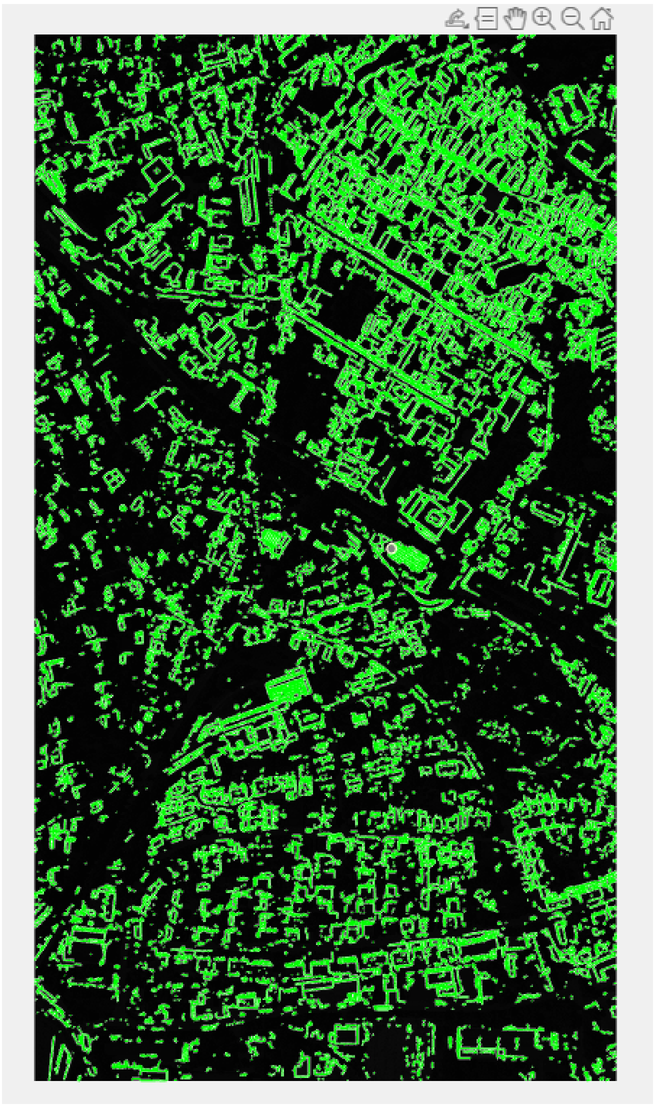


Figure 9: Projected boundaries of all buildings

In this part, I tried Hysteresis Thresholding on my own initiative. The Hysteresis thresholding algorithm was covered in the lecture, and I can say that it is an algorithm used to assign the most optimum value for the threshold.

Additionally, the Canny edge detector, as covered in the lecture, was employed. This operator ensures that edges are thin and connected, allowing for the most optimal extraction of edges.

As seen in Figure 9, only the boundaries of the buildings have been extracted from the image, and the boundaries have been colored.

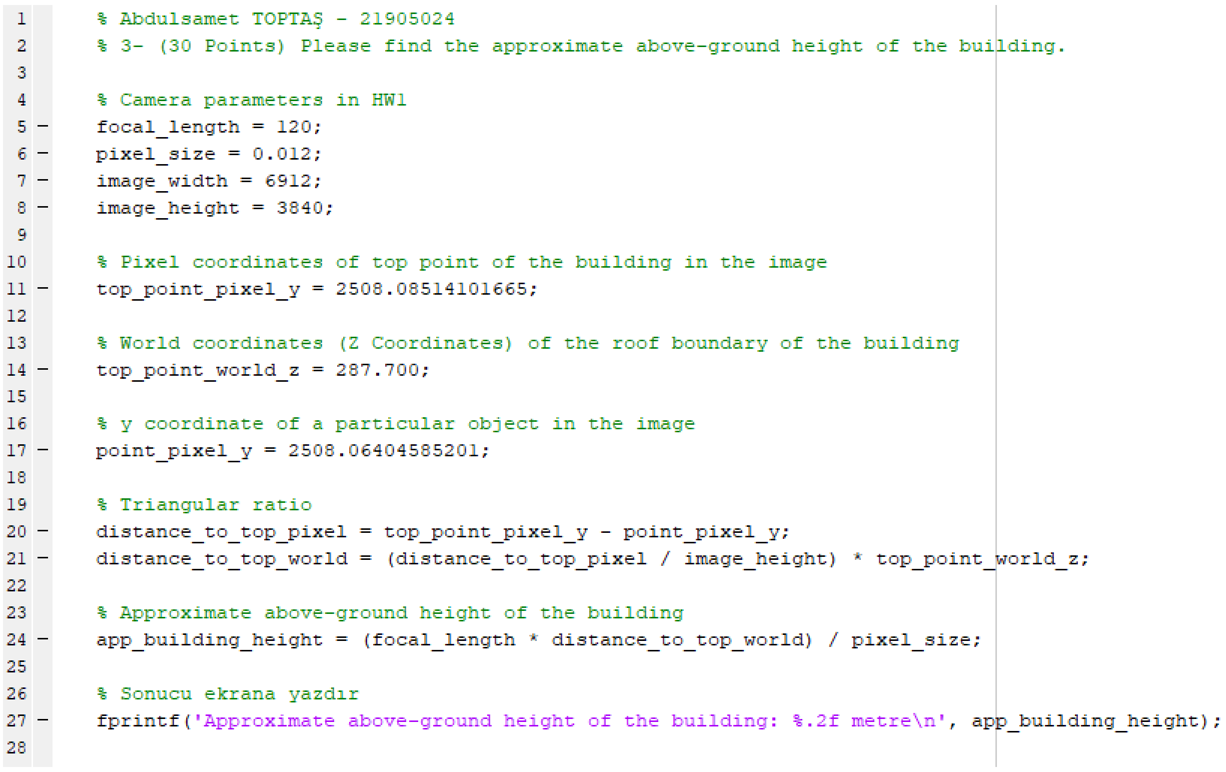
3- Explanations for Question 3

First of all, if I were to talk about different methods verbally;

Shadow is available. For this, the angle of the light coming from the sun must be known. For example, along with the satellite images, there is also information about the date of collection, the off-center angle of the spacecraft at the time of capture (degrees perpendicular to the sensor from the array), the azimuth of the sun, and the angle of the satellite. With this information, sun angle and shadow length can be calculated. We can calculate building height from sun angle and shadow length. For example, tan(Sun Height) = (Height of the Object) / (Length of the Shadow) will give us an estimate of the height of the object.

Another method is geometric height estimation. Additionally, a MATLAB code was written for this method.

A number of formulas can be used to calculate the approximate ground height of a building using the pixel coordinates of a particular point in a camera image and the Z coordinate of that point in world coordinates.



```
1      % Abdulsamet TOPTAŞ - 21905024
2      % 3- (30 Points) Please find the approximate above-ground height of the building.
3
4      % Camera parameters in HW1
5  -   focal_length = 120;
6  -   pixel_size = 0.012;
7  -   image_width = 6912;
8  -   image_height = 3840;
9
10     % Pixel coordinates of top point of the building in the image
11 -   top_point_pixel_y = 2508.08514101665;
12
13     % World coordinates (Z Coordinates) of the roof boundary of the building
14 -   top_point_world_z = 287.700;
15
16     % y coordinate of a particular object in the image
17 -   point_pixel_y = 2508.06404585201;
18
19     % Triangular ratio
20 -   distance_to_top_pixel = top_point_pixel_y - point_pixel_y;
21 -   distance_to_top_world = (distance_to_top_pixel / image_height) * top_point_world_z;
22
23     % Approximate above-ground height of the building
24 -   app_building_height = (focal_length * distance_to_top_world) / pixel_size;
25
26     % Sonucu ekrana yazdır
27 -   fprintf('Approximate above-ground height of the building: %.2f metre\n', app_building_height);
28
```

Figure 10: Approximate above-ground height of the building

First, the farthest value in the y-line from the building roof pixel coordinates is selected, then the value of an object right next to it on the y-line is selected. While selecting the value of this approximate object, a border next to the previously detected building was selected from the edge algorithm seen in Figure 9 (2508.06404585201 value in Figure 10). Then, the distance differences between these two points are taken "20.row in Figure 10" and divided by the image height in pixels "21.row in Figure 10". Normalization is done with this process, that is, it allows us to make the vertical position of the point on an independent scale when working with different image sizes or resolutions.

Then, we multiply this ratio by the elevation of that corner from the world coordinates "21.row in Figure 10". It represents the vertical distance of a corner of the building in world coordinates. That is, we use it to convert the height of an object in the image into world coordinates.

Then, this found value is multiplied by the focal length, the reason for this is to calculate the dimensions or distances of an object within perspective geometry.

Finally, the ratio of this value to the pixel size will approximately give us the height of the building.

4- Please comment/discuss/explain the problems/reasons/causes you face in the output building boundaries projected on image 61.jp2

First of all, there are so many edges in the image that it is very difficult to find a very reasonable "threshold value". In other words, even if a reasonable threshold value is found, the boundaries of some buildings may disappear and this is something we do not want.

In addition, the line thickness value can also be adjusted, but it is very difficult to find the most optimum value. Unless the optimum value is found, the boundaries of adjacent buildings may collide and we may experience building loss.

One of the difficulties was that the image was grayscale. This was because there was no color in the image, so some building boundaries did not come out sharply as they made a smoother transition, offering an opposite approach to the Canny algorithm because the edges were not connected. Additionally, lines appeared in some shadow areas, creating the appearance of a fake building.

Another difficulty is that the structures in the image appear very close to each other. When the building boundaries were removed, some boundaries were seen to be intertwined, causing the building shape and structure to deteriorate.

Since there are many buildings in the image and it is a high resolution image, it has a large data volume. This complicated the transactions and made some details unclear by providing excessive details.