Attitude Determination of a Moon Orbiting Satellite with a Unity-Aided Visualization

Murathan Bakır[1]
Department of Astronautical Engineering, Istanbul Technical University

I.Abstract

This research focuses on enhancing the attitude determination capabilities of a satellite orbiting the Moon through Unity-aided visualization. The primary objectives involve 3D modeling of the Moon's surface in Unity for comprehensive crater catalogue comparison, investigation of attitude determination methods, and modeling/analysis of the satellite's rotational motion.

The methodology begins with the creation of a detailed 3D model of the Moon using Unity. A camera located on satellite captures the lunar surface, and the acquired images undergo processing using various image processing algorithms such as template matching and edge detection. These processed images are then compared with an existing crater catalogue. Upon successful matching, three craters are chosen as reference points, and the triangulation method is applied to determine the spatial position of the spacecraft equipped with the camera. Subsequently, post-attitude determination, the research delves into modeling and analyzing the rotational motion of the satellite.

This project not only contributes to the advancement of attitude determination techniques for Moon orbiting satellites but also employs Unity as a powerful visualization tool for space missions. The combination of 3D modeling, crater catalogue analysis, and rotational motion modeling aims to provide a comprehensive understanding of the satellite's dynamics and positioning in lunar orbit.

II.Introduction and Objectives

In the dynamic landscape of space exploration, the precise determination of a satellite's attitude—its orientation with respect to a reference frame—is integral to the success of missions, particularly those involving lunar orbits. Against the background of an interest expanding to Lunar discoveries, this project is the result of personal interest in space-themed engineering practices.

The motivation driving this project is multifaceted. In recent space missions, there has been a discernible shift toward exploring the Moon for scientific research, resource assessment, and potential future human habitation. As these missions grow in complexity, so does the need for sophisticated systems that can precisely determine the attitude of satellites orbiting the Moon. This project, therefore, stands as a response to this pressing need.

Besides the methods and approaches that used in this project, there are also AI (artificial intelligence) applications on image processing of the craters in literature. YOLO (You Only Look Once) and unsupervised algorithms were employed in earlier studies to detect craters. From one point of view, the alternative methods for crater detection are also successful. Unsupervised domain adaptation (UDA) is a machine learning paradigm that involves adapting a model that was trained on a source domain to perform well in a target domain. When there are variations in the domains' distributions, this adaptation is required. In order to help the model generalize well to new, unseen data without labelled information in the target domain, UDA seeks to minimize these disparities. UDA's primary advantage is its capacity to improve model performance when switching between various environments, which makes it a useful tactic for practical applications [1]. The YOLO algorithm, an AI framework intended for object detection, is the other effectively implemented system. One of its main uses is for object recognition in pictures or videos. YOLO sets itself apart by being able to quickly and accurately detect objects in

[1] E-mail: bakirm18@itu.edu.tr
Advisor: Demet Çilden Güler, Astronautical Engineering, Istanbul Technical University, cilden@itu.edu.tr

real time by analyzing an entire image or video in a single pass. It differs from other object detection techniques with this feature, which also makes it more appropriate for real-time processing applications. In the field of object detection, YOLO has gained widespread adoption due to its consistent ability to produce effective and efficient results [2]. Behind this project's vision, it is important to know that there are different approaches to the crater definition made with AI support.

The choice of Unity as a central tool in this project is deliberate and strategic. Unity, renowned for its prowess in 3D modeling and visualization, offers a robust platform to construct an intricate model of the Moon's surface. This model, when integrated with advanced imaging techniques, will allow for a detailed comparison of captured images with an existing crater catalogue. The incorporation of template matching and edge detection algorithms in image processing further refines this comparison, providing a nuanced understanding of the satellite's spatial position concerning lunar surface features.

The goals of the project are twofold:

3D Modeling of the Moon in Unity for Crater Catalogue Comparison:

The initial phase involves the creation of a high-fidelity 3D model of the Moon using Unity. This model serves as the canvas for subsequent imaging and analysis. Through the lens of a camera mounted on the satellite, the lunar surface is observed, and these images are processed for a comparative analysis with an established crater catalogue.

Investigating Pose Determination Methods:

Position and attitude determination, a critical aspect of satellite navigation, will be thoroughly explored. The project delves into various methods, with a focus on their efficacy in lunar scenarios. Particularly through the triangulation method, will play a central role in pinpointing the satellite's position concerning identified lunar features.

III.Methods and Resources

A. Methods Used in This Project

Crater Catalogue:

Catalogues of scientific craters are most useful in the development and testing stages. They offer a quick and simple way to get representative crater shapes and distributions, which are useful when creating algorithms or planning trajectories. Robbins Crater Catalogue will be used, despite the fact that there are many crater catalogues available. The Lunar Reconnaissance Orbiter (LRO) and SELENE data were primarily used to create the over two million craters in this catalogue that are located on the lunar surface [3].

Detection of Crater:

Several different techniques will be employed to locate the crater. There is never just one best crater detection algorithm [4]. The orbital regime (or descent trajectory), camera specifications, on-board memory and processing power, operational cadence, and other mission-specific factors all influence the right answer. The crater catalogue (raw data source), index scale (local, regional, global, or a combination), index organisation, and invariant descriptor type (projective invariants or p2 invariants) are important design decisions for crater detection [4].

Furthermore, we observe that the kind of invariance required for lost-in-space terrain relative algorithm (TRN) is not characterized by popular image feature descriptors such as SURF, BRIEF, ORB, SIFT, and others. They are not the appropriate tool, by design, for the problem this manuscript attempts to solve. There are two primary causes for this. First off, most of these descriptors are only formally invariant for a similarity transformation (translation, rotation, or scaling), even though many of them have been shown to be robust for a moderate amount of affine shear [5].

None of the classical feature descriptors for lost-in-space TRN are illumination geometry invariant, which is the second (and more significant) problem with them. In summary, all these feature descriptors do is identify and describe regions of unique two-dimensional patterns in an image. When light bounces off an airless body (like the Moon or an asteroid) and returns towards the camera, it creates a 2D pattern that is visible in the image [5].

Despite the mentioned drawbacks, widely known and commonly used algorithms such as SURF, ORB, and SIFT will be initially experimented with. Under the idealized conditions of the Unity environment, where real-world factors like light reflections and shadow illusions are absent.

Instead of algorithms mentioned above, for initial trials, edge detection method used for detection of craters. It is selected due to its fast and easy algorithm with well-known arhitecture. The examinations done on JupyterLab with Python 3. OpenCV library is used with numpy and matplotlib libraries. Canny Edge Detector algorithm is used. The Canny algorithm is unique in that it detects edges by lowering noise, determining edge gradients, joining edge pixels, and strengthening weak edges through a sequence of steps [6]. We will discuss canny parameters that are calculated using a guiding formula. The color map of the image was set to gray before the Canny algorithm was used to reinforce it. Craters are defined using the Hough Circle Detection method after the edges have been clarified and sharpened. Often used in computer vision, the Hough Circle Detection Method entails converting an image using edge detection, projecting possible circles onto a parameter space, adding up edge pixel contributions to determine probable circle parameters, and then projecting these parameters back to the original image to detect circles with a fixed radius. Hough method parameters derived from empirical observations [7].

Identification of Crater:

Robbin's Crater Database may be used for the comparison between taken photos from Unity environment and database. Crater detection algorithm (CDA) and crater identification algorithm (CIA) will be implemented. Comparison process of the craters made by template matching method. Again with OpenCV library, different types of template matching methods applied and compared with each other according to success of the results. The methods tried are: "TM_CCOEFF", "TM_CCOEFF_NORMED", "TM_CCORR", "TM_CCORR_NORMED", "TM_SQDIFF" and "TM_SQDIFF_NORMED". With the help of template matching, a heat map created which shows the possibility distribution of the wanted match on crater catalogue. It puts a red bold rectangle when the possibility of match is higher than the threshold value. Subsequently, the red bold rectangle taken apart and examined as a whole. Thence, comparison can be made more accurately due to the fact that that method eliminates the risk of the relevant part of the catalogue being searched by being cut off and scattered into different parts during the comparison process when the catalogue map is divided into pieces with certain fixed areas and examined one by one.

Triangulation:

After detection and identification, craters will be represented by placing dots at their centers. Then, they will be utilized for triangulation method. Through this way, the position of the spacecraft will be determined. To detect the spacecraft's position, Moon surface will be georeferenced in Unity environment. Georeferencing allows us to know the locations of craters as latitude and longitude after identification process. Thus, triangulation process determines the satellite's location by using latitudes and longitudes of detected and identified craters.

Because of their frequent occurrence on the Moon and the wealth of data regarding their positions, lunar craters are special features to use for terrain relative navigation (TRN). These benefits encouraged us to investigate a crater-based TRN solution for situations where optical images of the lunar surface and a priori state information are available. To identify craters, the suggested algorithm triangulates the spacecraft state and compares projected crater geometry. The crater identification algorithm (CIA) links multiple pixel coordinates of crater centers in the image to 3D points in the world frame by closely comparing triads in the crater catalogue against triads in the image. The findings showed that false positives could be rejected at the cost of a marginally higher chance of finding no match if algorithm parameters were carefully chosen [3].

B. Analysis of The Resources for This Project

All the codes and simulation applications done on the computer. The computer has following technical specifications:

- Intel® Tiger Lake Core™ i5-11400H 6C/12T; 12MB L3; 8GT/s; 2.7GHz > 4.5GHz; 45W; 10nm SuperFin
- Nvidia RTX3050 Max-Performance 4GB GDDR6 128-Bit DX12
- 15,6" FHD 1920x1080 144Hz AHVA Mat LED Screen
- 8GB (1x8GB) DDR4 1.2V 3200MHz SODIMM
- 500GB CRUCIAL P2 PCIe M.2 2280 PCIe 3.0 x4

IV.Details on the Simulation, Algorithms and Technical Analysis

1. 3D Modeling of Moon on Unity

Visualization of Moon:

In the project, a high-fidelity simulation of the Moon was crafted through the Unity program to achieve realistic visualization. A spherical model representing the Moon was generated, and a high-resolution 2D image of the Moon from NASA's website was mapped onto its surface [10]. However, distortions occurred in the polar regions due to the projection of a 2D image onto a 3D sphere.

Satellite with a camera orbitted in a circular orbit around the moon with a fixed velocity. The camera always looks at the Moon surface and takes pictures. Then, the pictures automatically being sent to a file which is source file for the image processing codes mentioned above. Finally, images taken from the file with python OpenCV library and the whole process begin. All the procedure mentioned above executed with C# language on Unity environment.

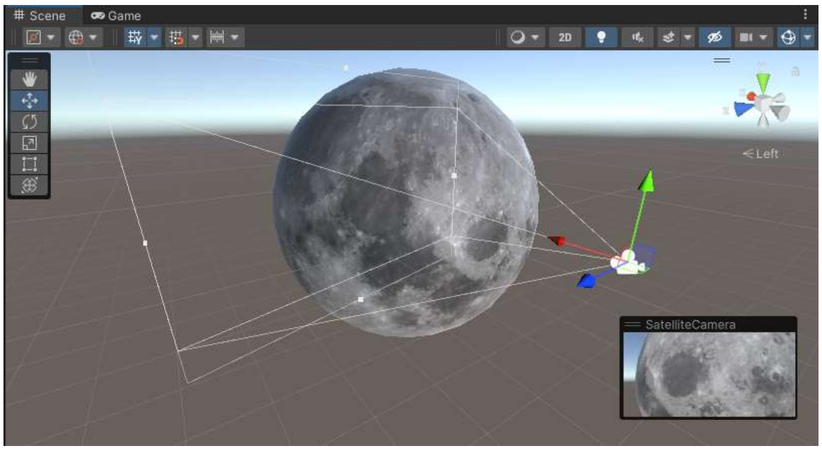


Figure 1: 3D Moon model on Unity Environment with satellite camera point of view in Scene mode.

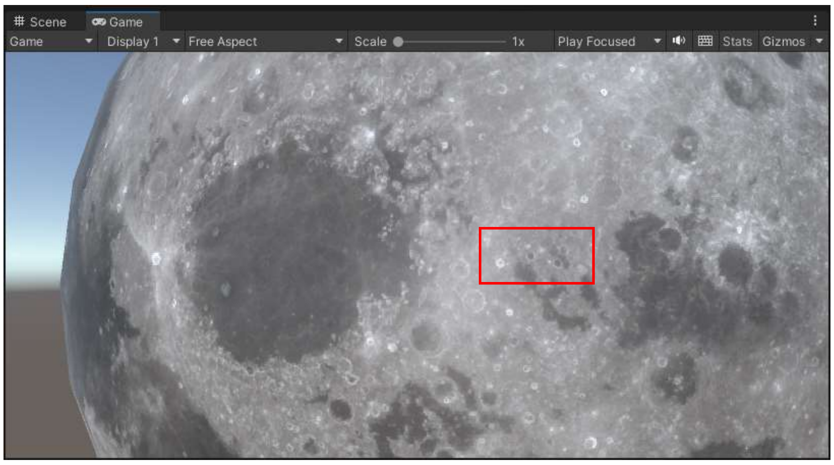


Figure 2: A look at the "Area 1" (the zone in the red rectangle) on the Moon from the satellite camera in Game mode.

To address the distortions, research was conducted on potential enhancements using alternative programs such as Blender. The simulation environment was populated with celestial bodies, including the Sun, Earth, Moon, and a satellite in lunar orbit. These elements were positioned in their respective orbits, with the requirement of maintaining a fixed position for the Sun. Rotation commands were defined in the C# programming language for all celestial bodies.

Ultimately, additional visual components such as light source and a camera on the satellite were integrated into the simulation environment. As a result, the modeling in Unity is considered complete, with the exception of addressing distortions in the polar regions, which are under active improvement.

Georeferencing:

Georeferencing is necessary to define the coordinates of the craters on the surface of the Moon. Through georeferencing, it is possible to obtain the locations of the craters in terms of latitude and longitude. This will be particularly helpful for satellite navigation purposes.

To perform georeferencing, it is necessary to transfer the cartesian coordinates of the Moon (x, y, z) to the global coordinates (latitude, longitude, altitude) in Unity. The formulation required for this process is given below in dark red lines.

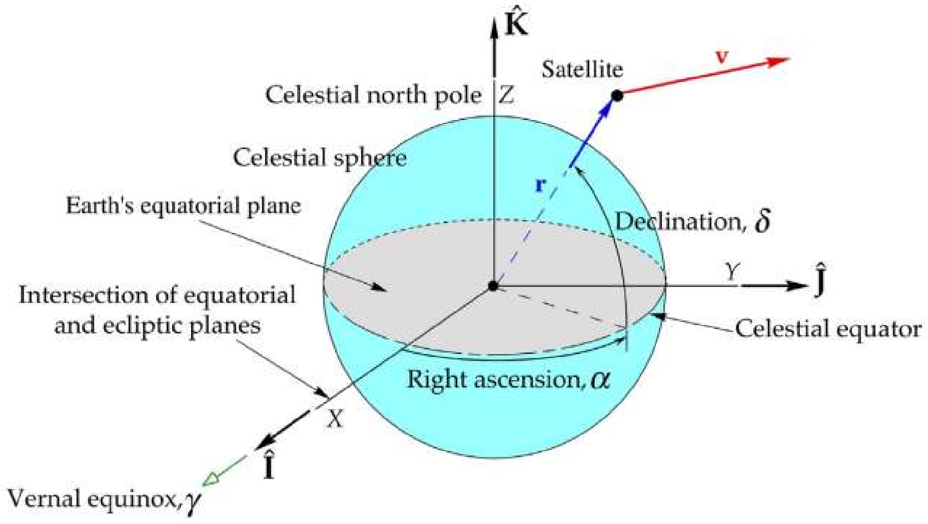
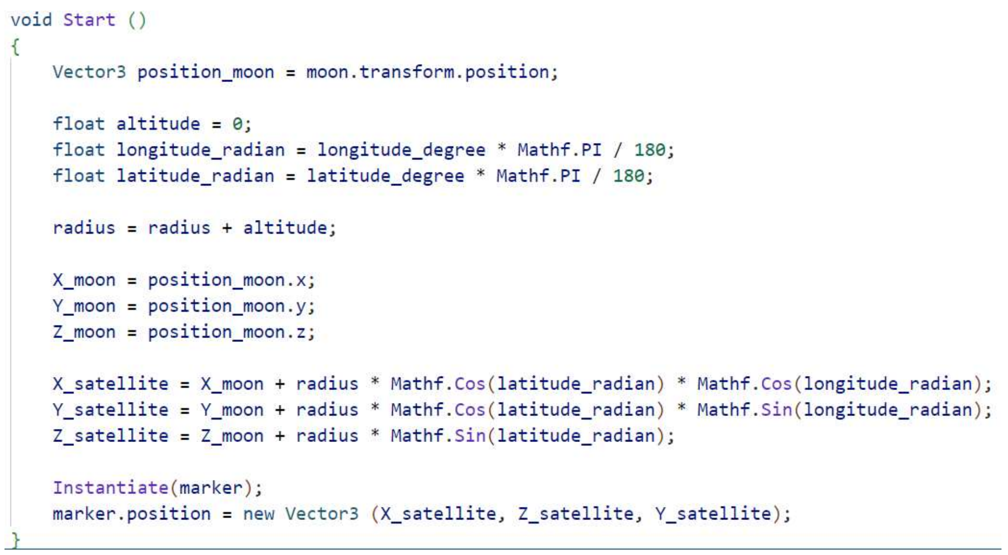


Figure 3: Transfer formulation from cartesian coordinates to global coordinates [11].

The formula for coordinate transfer mentioned above is:

$$ur = li + mj + nk = \cos\delta.\cos\alpha\ i + \cos\delta.\sin\alpha\ j + \sin\delta\ k$$

Here is the C# code used in Unity to perform the coordinate transfer mentioned above in Fig. 3:

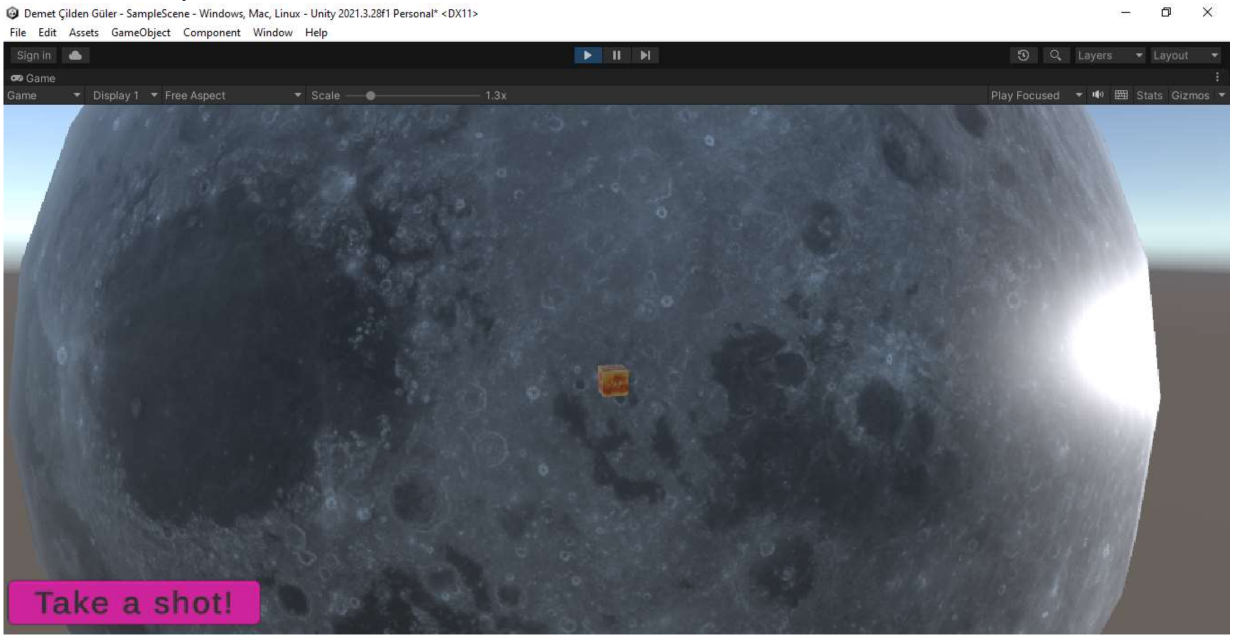


```
void Start ()
{
    Vector3 position_moon = moon.transform.position;

    float altitude = 0;
    float longitude_radian = longitude_degree * Mathf.PI / 180;
    float latitude_radian = latitude_degree * Mathf.PI / 180;

    radius = radius + altitude;

    X_moon = position_moon.x;
    Y_moon = position_moon.y;
    Z_moon = position_moon.z;

    X_satellite = X_moon + radius * Mathf.Cos(latitude_radian) * Mathf.Cos(longitude_radian);
    Y_satellite = Y_moon + radius * Mathf.Cos(latitude_radian) * Mathf.Sin(longitude_radian);
    Z_satellite = Z_moon + radius * Mathf.Sin(latitude_radian);

    Instantiate(marker);
    marker.position = new Vector3 (X_satellite, Z_satellite, Y_satellite);
}
```

Figure 4: Transfer formulation from cartesian coordinates to global coordinates in C#.

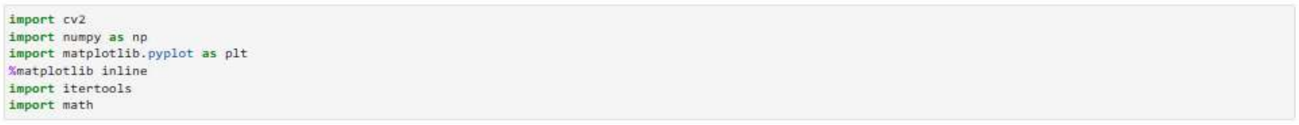
The snippet in Fig. 4:

1) Gets latitude, longitude and altitude data form python codes (at the end of the triangulation process, these data become obtainable via python codes that will be mentioned in the next chapters). However temporarily, manual assignments were made for the sake of simplicity of display in this section.
2) Then, it finds the distance between Moon center and satellite by adding radius and altitude value.
3) Afterall, it locates the position of the satellite with respect to Moon's location by using the formula given in the Fig. 3. In this way, no matter where the Moon is in Unity environment, the satellite will be around the Moon with the correct latitude, longitude and altitude values.
4) At the end of the code, a marker is placed on the satellite's position to verify the accuracy of the georeferencing process. Y and Z coordinates assigned in reverse in order to comply with Unity's cartesian coordinate system.


Demet Çilden Güler - SampleScene - Windows, Mac, Linux - Unity 2021.3.28f1 Personal* <DX11>
File Edit Assets GameObject Component Window Help
Sign in
Layers
Layout
Game
Game
Display 1
Free Aspect
Scale
1.3x
Play Focused
Stats
Gizmos
Take a shot!
Figure 5: A cubic red marker placed on the "Area 1" by using the coordinate values of the satellite.

2. Edge Detection

As an initial survey for detection of crater, edge detection method is selected. Canny Edge Detector algorithm is used. For all processes such as edge detection, circle detection and triangulation, the libraries given below in Fig. 6 are used.



```
import cv2
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import itertools
import math
```

Figure 6: The libraries used in detection, identification and triangulation processes.

From the catalogue, an example area found and called as "Area 1" for initial examinations shown in Fig. 7. The area is chosen since the number, radii and shapes of the craters are evaluated as an optimum scenario.

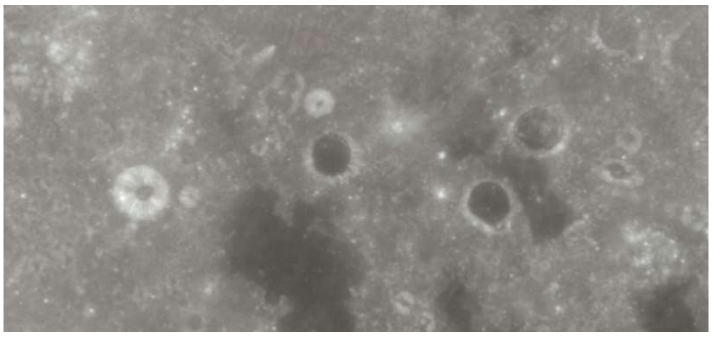
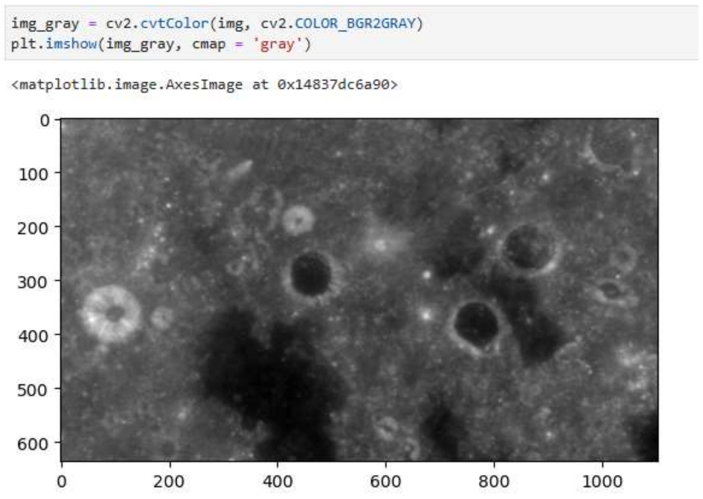


Figure 7: “Area 1” is selected as ideal zone from catalogue.

With OpenCV library, the color map of picture converted into gray in order to help to edge detection algorithm. This application provides more distinctive crater-surface transitions as seen in Fig. 8.



```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(img_gray, cmap = 'gray')
```

```
<matplotlib.image.AxesImage at 0x14837dc6a90>
```

Figure 8: “Area 1” with gray color map.

Canny Edge Detector algorithm is a multi-stage algorithm which is used in the project. It applies Gaussian filter to smooth the image in order to remove the noise. Subsequently, it finds the intensity gradients of the image. Then, it applies non-maximum suppression to get rid of spurious response to edge detection. Afterall, it applies double threshold to determine potential edges. Finally, it ends the detection of edges by suppressing other edges that are weak and not connected to strong edges which called “edge tracking by hysteresis”. The Canny Edge Detector requires 2 threshold level for “double thresholding” step. They are defined and explained in detail below:

Edge Thresholding:

The "hysteresis" thresholding technique is what the Canny Edge Detector employs. It uses both a low threshold and a high threshold. A pixel is designated as an edge pixel if its value is higher than the high threshold. A pixel is also designated as an edge pixel if it is adjacent to an edge pixel and has a value higher than the low threshold. A pixel is not designated as an edge pixel if its value is above the low threshold and it is not next to another edge pixel. A pixel is never designated as an edge pixel if its value is below the low threshold [6].

There is a statement to determine the upper and lower thresholds of Canny Edge Detector. The statement tends to be more accurate especially after blurring the image before Canny method. However, the formula given below is not always correct but tends to be accurate. It is required to know the median pixel value for the formula. Lower and upper thresholds are determined as following statement:

- Lower threshold equals to either 0 or 70% of the median pixel value whichever is greater.
- Upper threshold equals to either 130% of the median pixel value or the maximum 255 whichever is smaller.

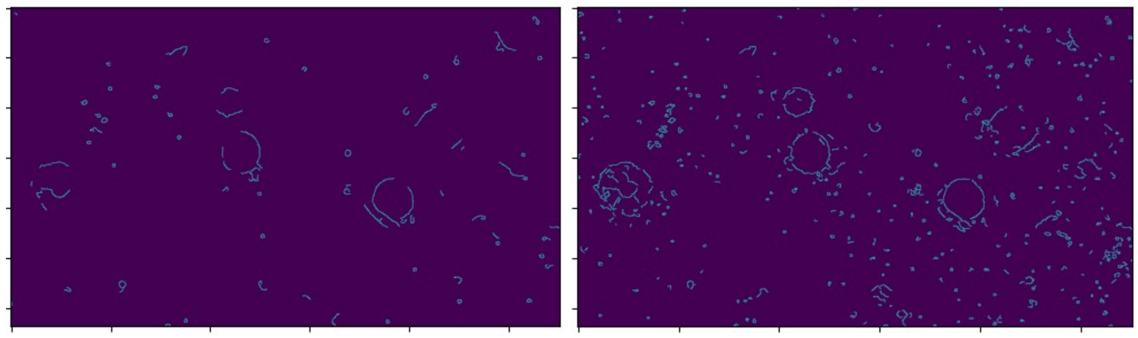
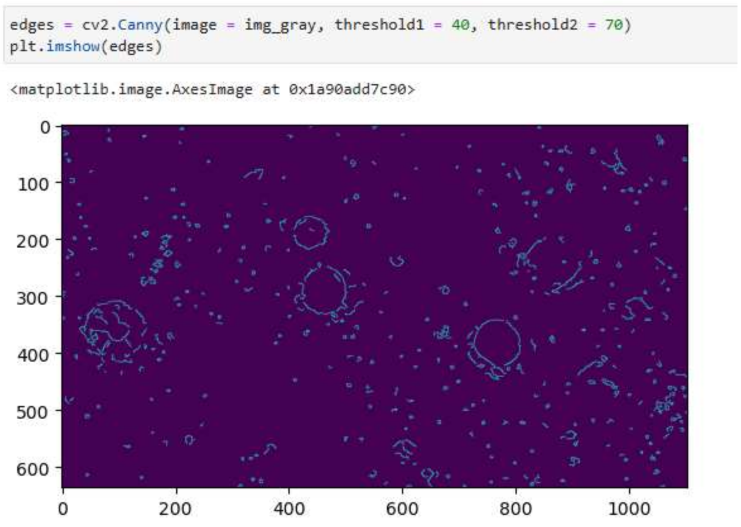


Figure 9: Comparison of applied formula thresholds and empirical thresholds. Left: Thresholds selected according to formula. Right: Thresholds selected according to empirical observations.

According to empirical observations lower threshold is selected using trial and error method as 40 and upper threshold selected as 70. These two thresholds give the best and most accurate result. The thresholds taken by the statement above suppress the edges too much and fail in plenty of area.



```
edges = cv2.Canny(image = img_gray, threshold1 = 40, threshold2 = 70)
plt.imshow(edges)
```

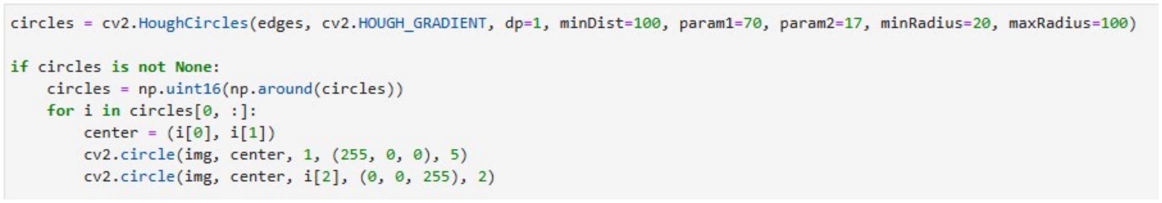
```
<matplotlib.image.AxesImage at 0x1a90add7c90>
```

Figure 10: The snippet of Canny Edge Detector and its result.

3. Hough Circle Detection

The Hough Circle Detection algorithm is used for circle detection after edge detection. The Hough circle transform exhibits significant resistance to problematic object boundaries, including those that are broken, partially obscured by occlusion, overlapping effects, and interference noise, as well as those that are warped by ambient light, object motion, and interference noise [7]. In order to locate circles precisely, the Hough Circle Detection method entails smoothing an image to remove noise, identifying potential circles in parameter space using a Circular Hough Transform, thresholding to highlight important edges, translating identified parameters back to the original image, and finally performing edge recognition [8].

It has 8 parameters in OpenCV library (see Fig. 11).



```
circles = cv2.HoughCircles(edges, cv2.HOUGH_GRADIENT, dp=1, minDist=100, param1=70, param2=17, minRadius=20, maxRadius=100)

if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
        center = (i[0], i[1])
        cv2.circle(img, center, 1, (255, 0, 0), 5)
        cv2.circle(img, center, i[2], (0, 0, 255), 2)
```

Figure 11: Hough Circle Detection parameters.

In the snippet above, the first parameter “edges” represents the result map of Canny algorithm. The second parameter “HOUGH_GRADIENT” is the most common method. The third parameter “dp” is Inverse ratio of the accumulator resolution to the image resolution. The fourth parameter defines the minimum distance for detected circles. The fifth parameter "param1" is the upper threshold value of the Canny algorithm. The sixth and the most important one is "param2" which is accumulator threshold for circle detection. A smaller value will result in more detected circles. 17 selected from empirical observations. The last 2 parameters are for radius limits of detected circles. They are also crucial since their limitations keep the image clear and focused on certain dimensions that we study on. All parameters after "param1" are determined and optimized with empirical trials by observing the results.

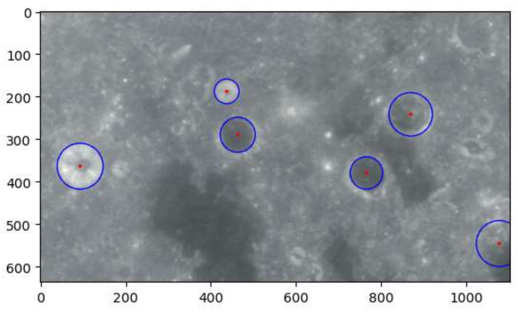


Figure 12: The result of selected best parameters.

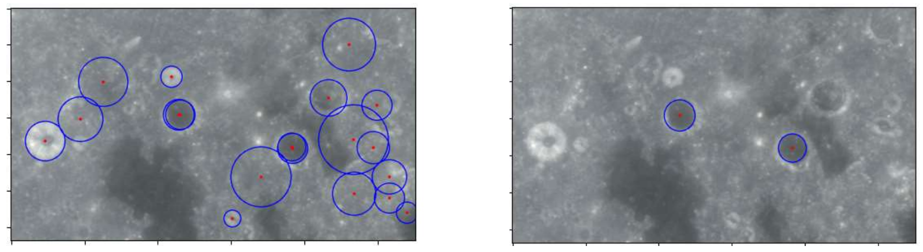
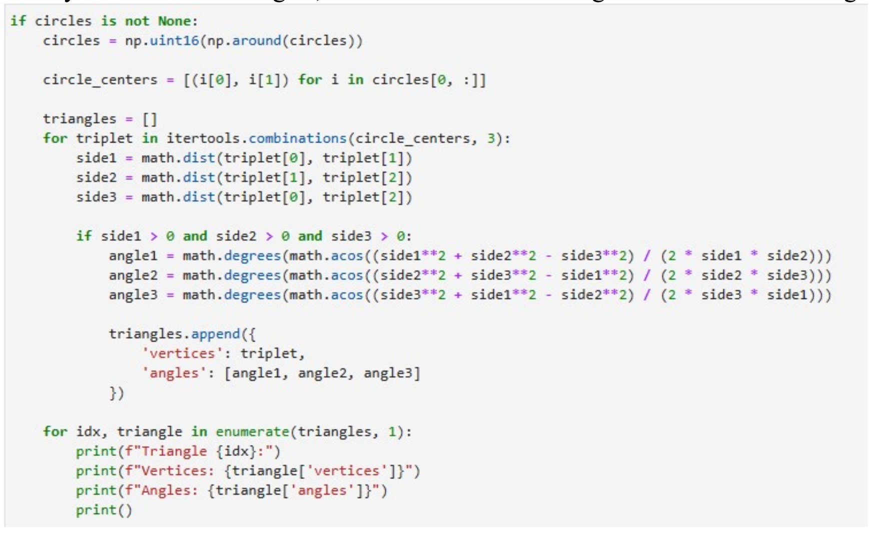


Figure 13: Left: Suppression is less than optimal value (param2=13), Right: Suppression is more than optimal value (param2=25).

In the "for loop" in Fig. 11, detected circles marked as blue and their centers signed as red point for visualization of the result. As seen in Fig. 12, the incomplete circles are also can be detected. That results in better catalogue scanning performance.

4. Triad Formation and Angle Arrangement for Catalog Scan

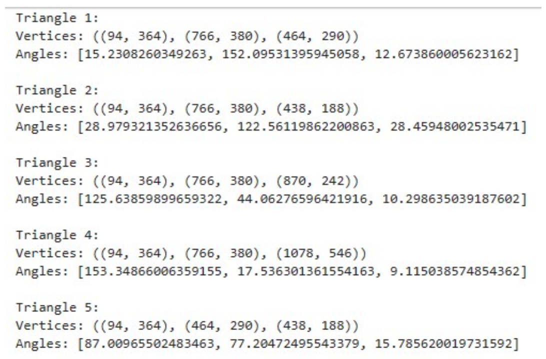
After detection of the craters with their center points, it is time to perform the triangulation method. However, to begin with, it is necessary to define the triangles, determine the interior angles and scan the catalogue first.



```python
if circles is not None:
    circles = np.uint16(np.around(circles))

    circle_centers = [(i[0], i[1]) for i in circles[0, :]]

    triangles = []
    for triplet in itertools.combinations(circle_centers, 3):
        side1 = math.dist(triplet[0], triplet[1])
        side2 = math.dist(triplet[1], triplet[2])
        side3 = math.dist(triplet[0], triplet[2])

        if side1 > 0 and side2 > 0 and side3 > 0:
            angle1 = math.degrees(math.acos((side1**2 + side2**2 - side3**2) / (2 * side1 * side2)))
            angle2 = math.degrees(math.acos((side2**2 + side3**2 - side1**2) / (2 * side2 * side3)))
            angle3 = math.degrees(math.acos((side3**2 + side1**2 - side2**2) / (2 * side3 * side1)))

            triangles.append({
                'vertices': triplet,
                'angles': [angle1, angle2, angle3]
            })

    for idx, triangle in enumerate(triangles, 1):
        print(f"Triangle {idx}:")
        print(f"Vertices: {triangle['vertices']}")
        print(f"Angles: {triangle['angles']}")
        print()
```

Figure 14: Snippet for creation of triangles that their vertices are the centers of detected circles.

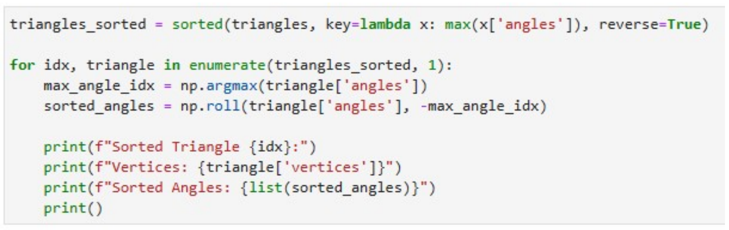
The snippet in Fig. 14 give an output of all possible triangles whose vertices are the centers of a circles. The Hough Circle Detector found 6 circles as seen in Fig. 12. Therefore, the snippet in Fig. 14 defined 20 triangles. Initial part of output of the snippet above shown in Fig. 15 below:



```
Triangle 1:
Vertices: ((94, 364), (766, 380), (464, 290))
Angles: [15.2308260349263, 152.09531395945058, 12.673860005623162]

Triangle 2:
Vertices: ((94, 364), (766, 380), (438, 188))
Angles: [28.979321352636656, 122.56119862200863, 28.45948002535471]

Triangle 3:
Vertices: ((94, 364), (766, 380), (870, 242))
Angles: [125.63859899659322, 44.06276596421916, 10.298635039187602]

Triangle 4:
Vertices: ((94, 364), (766, 380), (1078, 546))
Angles: [153.34866006359155, 17.536301361554163, 9.115038574854362]

Triangle 5:
Vertices: ((94, 364), (464, 290), (438, 188))
Angles: [87.00965502483463, 77.20472495543379, 15.785620019731592]
```

Figure 15: The initial part of the output of the triangle creator snippet.

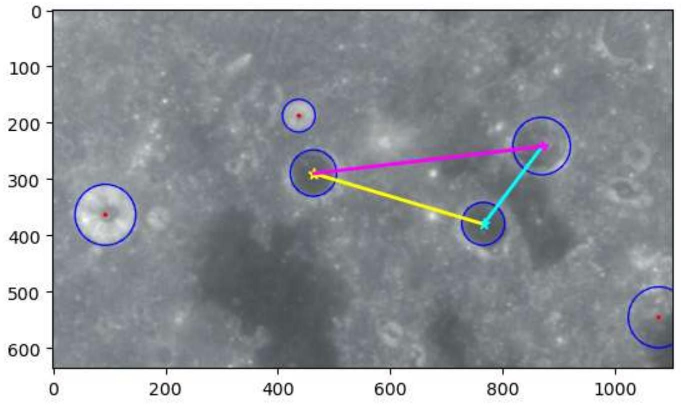
Afterwards, the interior angles of the triangles were arranged with the largest ones first and the others proceeding counterclockwise after largest one. That process eliminates the possible triangle conflicts when making comparison with whole catalogue since the arrangement according to counterclockwise alignment with respect to greatest one makes the superposition probability of triangles lower relative to other arrangement methods like sorting from largest to smallest. The mentioned algorithm done by "roll" method in NumPy library, shown in Fig. 16.

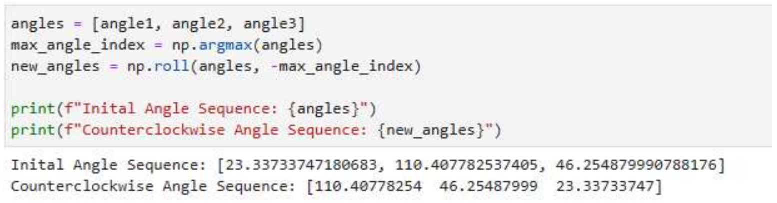


```python
triangles_sorted = sorted(triangles, key=lambda x: max(x['angles']), reverse=True)

for idx, triangle in enumerate(triangles_sorted, 1):
    max_angle_idx = np.argmax(triangle['angles'])
    sorted_angles = np.roll(triangle['angles'], -max_angle_idx)

    print(f"Sorted Triangle {idx}:")
    print(f"Vertices: {triangle['vertices']}")
    print(f"Sorted Angles: {list(sorted_angles)}")
    print()
```

Figure 16: Arranger snippet for interior angles of triangles.

Finally, stars were placed at the corners of a triangle chosen among 20 triangles for a clear demonstration. Then, the interior angles of the selected triangle sorted according to counterclockwise rule mentioned above. The result shown in Fig. 17.





```python
angles = [angle1, angle2, angle3]
max_angle_index = np.argmax(angles)
new_angles = np.roll(angles, -max_angle_index)

print(f"Inital Angle Sequence: {angles}")
print(f"Counterclockwise Angle Sequence: {new_angles}")
```

```
Inital Angle Sequence: [23.33733747180683, 110.407782537405, 46.254879990788176]
Counterclockwise Angle Sequence: [110.40778254  46.25487999  23.33733747]
```

Figure 17: The clear demonstration of triangle formation and angle arrangement.

The triangle (triad) shown in Fig. 17 selected for comparison. The magnitude of interior angles and their sequence will be compared along the catalogue. An error threshold "ε" is going to be determined and will be utilized as accuracy factor throughout the catalogue matches.

5. Catalog Scanning and Template Matching

When it comes to catalog scanning, it is important to scan the catalog as a whole and meticulously. If the catalog map is divided into fixed areas for comparison process, the desired area (in this case the Area 1 region) can be divided into different parts. That results in failure since it is not possible to detect the desired triangle if vertices of the triangle are located on the different divided parts. In order to prevent this, template matching method considered as a solution thanks to its feature of finding parts from whole. While matching the template, the method also can create a heatmap and help the scanning process. One common operation in image analysis is the correlation of a small "template" array with a large image array. Applications of the technique include comer finding, edge detection, and line detection [9].

For template matching, Area 1 and a wider area which includes the Area 1 are selected. Subsequently, 6 different template matching methods tested. These methods, called from the OpenCV library are as follows; "TM_CCOEFF", "TM_CCOEFF_NORMED", "TM_CCORR", "TM_CCORR_NORMED", "TM_SQDIFF" and "TM_SQDIFF_NORMED".

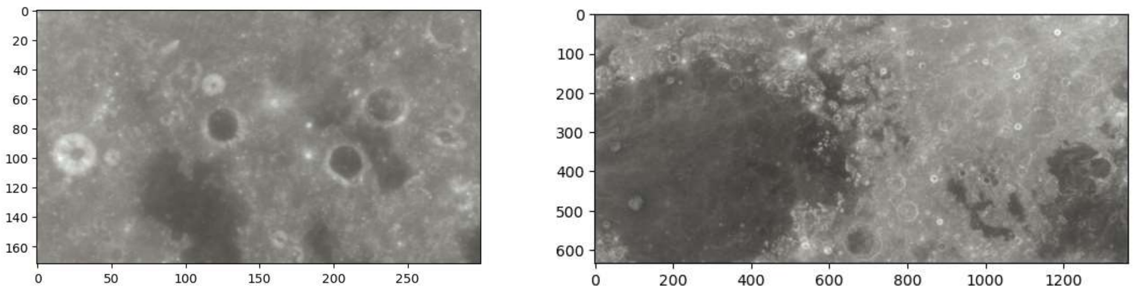
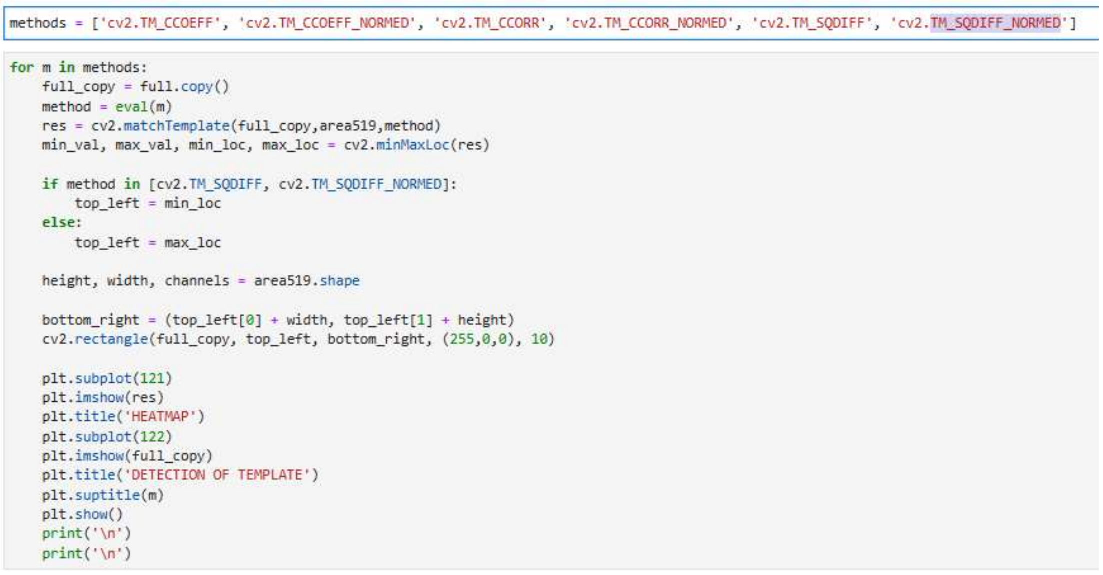


Figure 18: "Area 1" on the left and wider area on the right.



```
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR', 'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']

for m in methods:
    full_copy = full.copy()
    method = eval(m)
    res = cv2.matchTemplate(full_copy,area519,method)
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

    if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc

    height, width, channels = area519.shape

    bottom_right = (top_left[0] + width, top_left[1] + height)
    cv2.rectangle(full_copy, top_left, bottom_right, (255,0,0), 10)

    plt.subplot(121)
    plt.imshow(res)
    plt.title('HEATMAP')
    plt.subplot(122)
    plt.imshow(full_copy)
    plt.title('DETECTION OF TEMPLATE')
    plt.suptitle(m)
    plt.show()
    print('\n')
    print('\n')
```

Figure 19: The snippet for template matching operation.

The snippet in Fig. 19 creates heatmap and puts a red rectangle on the area with the highest probability of match. The results from first 4 methods are wrong. However, last 2 methods called "TM_SQDIFF" and "TM_SQDIFF_NORMED" respectively give correct outputs as seen in the figures below:

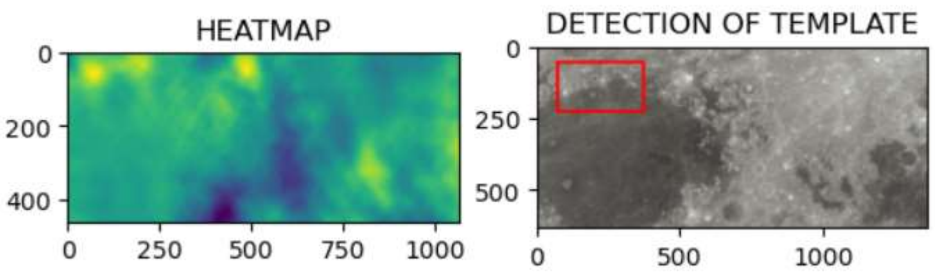


Figure 20: Output of TM_CCOEFF method. The output is wrong with a good heatmap. Yellow regions are stands for the highest match possibilities.

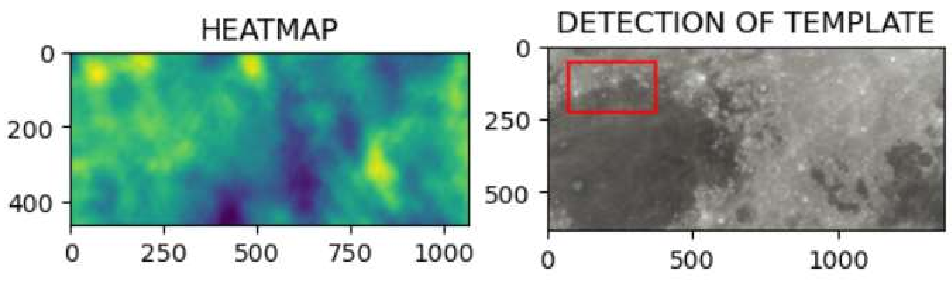


Figure 21: Output of TM_CCOEFF_NORMED method. The output is wrong with a good heatmap. Yellow regions are stands for the highest match possibilities.

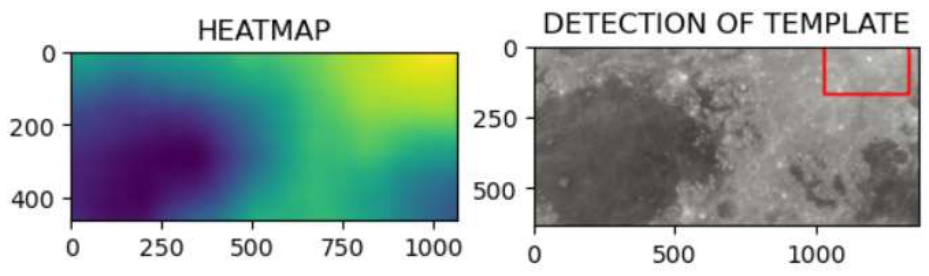


Figure 22: Output of TM_CCORR method. The output is wrong with a bad heatmap. Yellow regions are stands for the highest match possibilities.

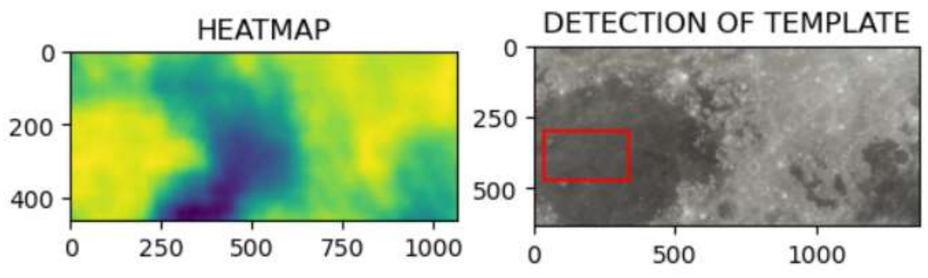


Figure 23: Output of TM_CCORR_NORMED method. The output is wrong with an intermediate heatmap. Yellow regions are stands for the highest match possibilities.

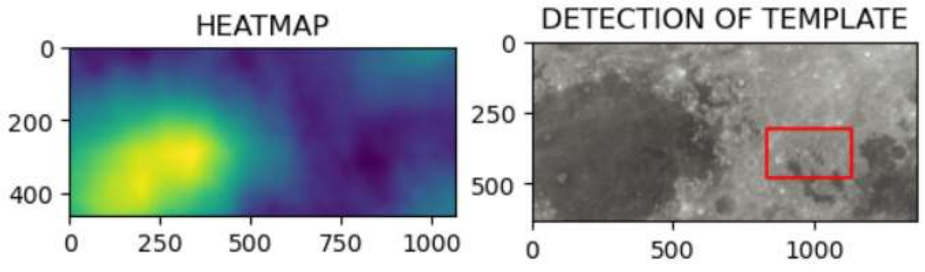


Figure 24: Output of TM_SQDIFF method. The output is correct with an excellent heatmap. The purple regions show the lowest value of method parameter which corresponds the highest match possibility unlike previous methods.

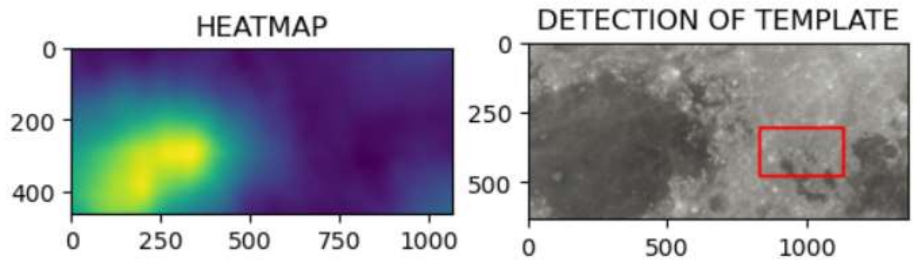
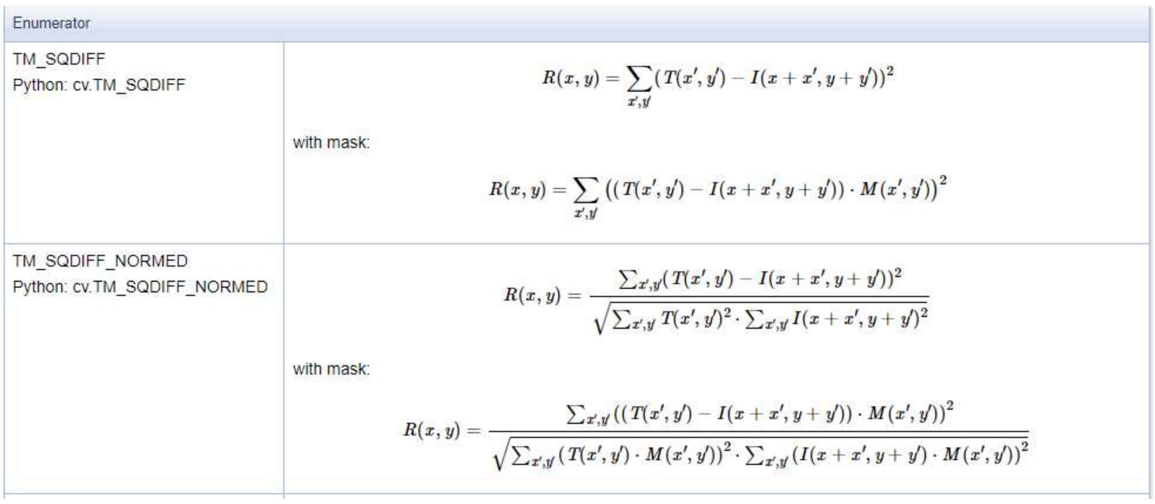


Figure 25: Output of TM_SQDIFF_NORMED method. The output is correct with an excellent heatmap. The purple regions show the lowest value of method parameter which corresponds the highest match possibility unlike previous methods.



Enumerator	
TM_SQDIFF Python: cv.TM_SQDIFF	$R(x,y)=\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2$ with mask: $R(x,y)=\sum_{x',y'}\left((T(x',y')-I(x+x',y+y'))\cdot M(x',y')\right)^2$
TM_SQDIFF_NORMED Python: cv.TM_SQDIFF_NORMED	$R(x,y)=\frac{\sum_{x',y'}(T(x',y')-I(x+x',y+y'))^2}{\sqrt{\sum_{x',y'}T(x',y')^2\cdot\sum_{x',y'}I(x+x',y+y')^2}}$ with mask: $R(x,y)=\frac{\sum_{x',y'}\left((T(x',y')-I(x+x',y+y'))\cdot M(x',y')\right)^2}{\sqrt{\sum_{x',y'}(T(x',y')\cdot M(x',y'))^2\cdot\sum_{x',y'}(I(x+x',y+y')\cdot M(x',y'))^2}}$

Figure 26: Formulations of TM_SQDIFF and TM_SQDIFF_NORMED methods.

6. Determination of Satellite's Position with Triangulation Method

Triangulation:

Triangulation is a common technique for localizing a point as the intersection of directions for a line of sight (LOS). The two possible forms of triangulation are intersection and resection, as demonstrated below. While intersection determines the position of an unknown point by observing it from at least two other known locations, resection is a triangulation technique that can determine an unknown point position by observing at least two other points at known locations [3]. Intersection technique will be used for determination of the satellite's location since satellite observes the Moon surface from an unknown location and aims to find its own position in space by using the 2 craters' known locations.

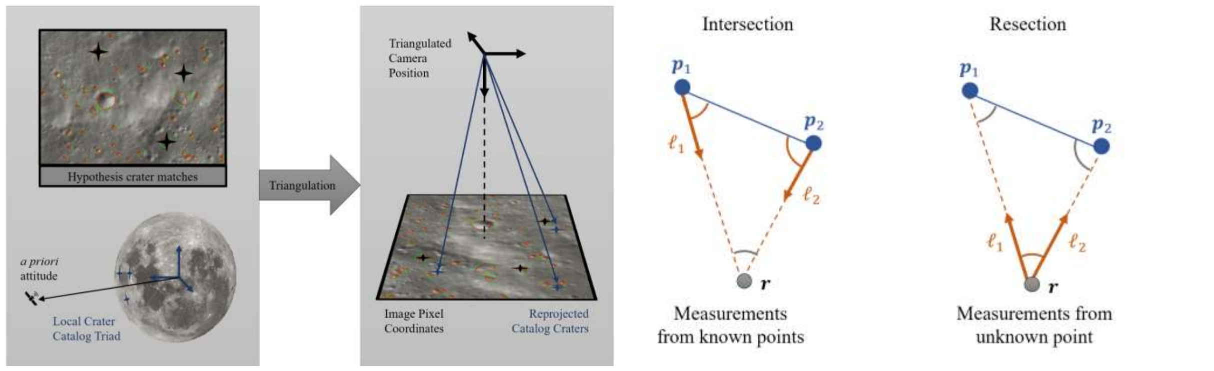
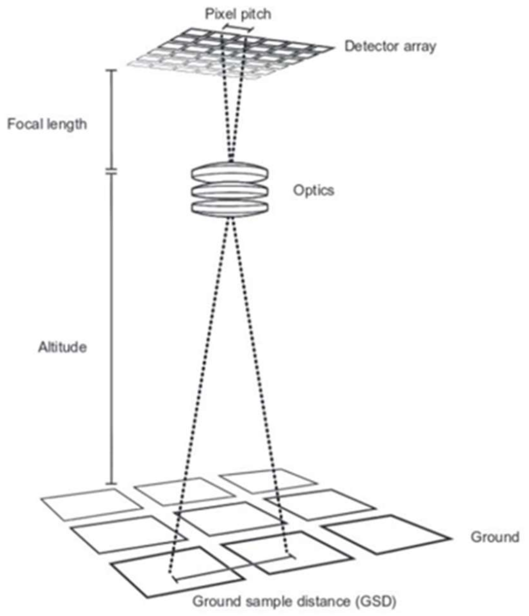

Hypothesis crater matches
a priori attitude
Local Crater Catalog Triad
Triangulation
Triangulated Camera Position
Image Pixel Coordinates
Reprojected Catalog Craters
Intersection
p_1
p_2
ℓ_1
ℓ_2
r
Measurements from known points
Resection
p_1
p_2
ℓ_1
ℓ_2
r
Measurements from unknown point

Figure 27 a): Visualization of triangulation [3]. b) Triangulation techniques Intersection and Resection. Intersection technique uses 2 known locations to find the observer's unknown location. Resection technique uses observer's location and another known location to find an unknown location in space [3].

Altitude Calculation:

It is required to know the altitude of the satellite in order to apply the triangulation problem. Satellite sends the photograph that taken by its own camera to the python code. The python code will calculate the altitude of the satellite by measuring the distances between the two spesific points in pixels. The steps of the altitude calculation process is counted below:

1) Satellite photograph is taken by python code.
2) Coordinates of the craters assigned with respect to the crater map.
3) Distance between 2 random detected craters is measured in pixels (pixel pitch in Fig. 28).
4) Real distance between that 2 points is measured by using coordinates (GSD in Fig. 28).
5) Altitude is calculated via using pixel pitch, ground sample distance (GSD) and focal length of the camera according to the formula given in the Fig. 28.



$$GSD = \frac{Orbital\ Height\ x\ Pixel\ Size}{Focal\ Length}$$

Orbital Height = (GSD x Focal Length) / Pixel Size

Figure 28: Altitude calculation of the satellite.

Note: Focal length of the camera can be arranged in Unity environment.

7. Determination of Satellite's Attitude

Attitude determination of the satellite on the yaw axis will be done by following method. 2 craters in the camera frame are picked and their global coordinates are assigned according to the crater catalogue. A line will be drawn through the centers of the craters. 2 points in the line are known from the craters' coordinates. Thus, slope of the line can be found by using coordinates of craters.

In Fig. 29, there is an illustration of the calculation mentioned above is shown.

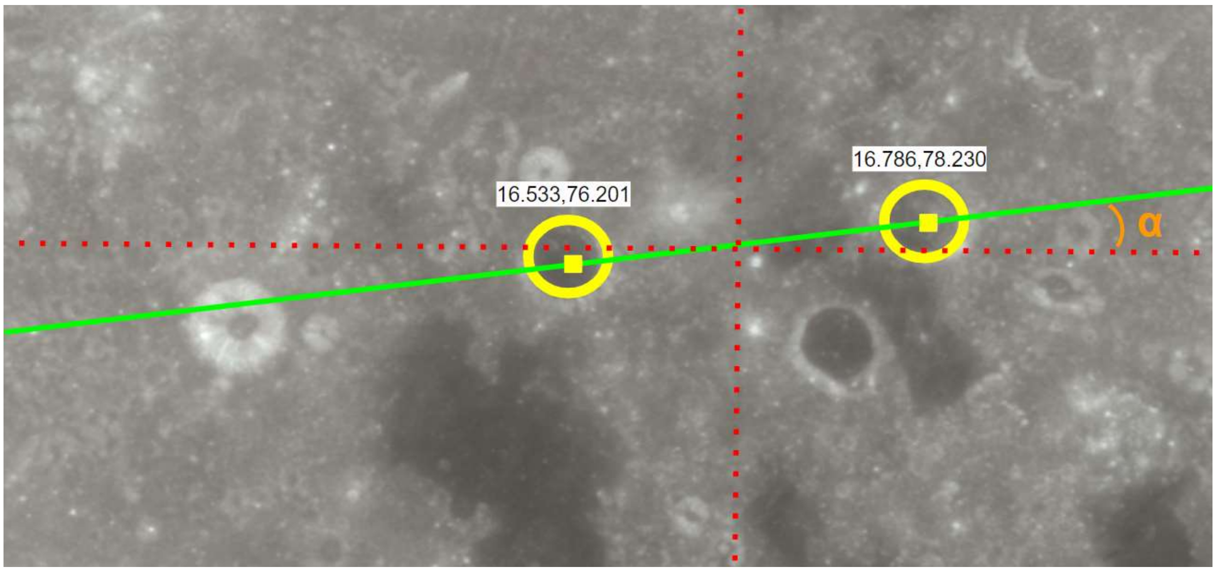


Figure 29: Attitude determination in yaw axis by using straight line approximation.

In the example in Fig. 29, slope of the can be found according to following formulas:

Distance between latitudes:

$$D = \Delta\phi \times (\pi / 180) \times r$$

"Δϕ" is angle difference between latitudes in degrees which equals to 0.253° in the example in Fig. 29.
"r" is radius of the Moon which is 1,737.4 km.
Distance between latitudes "D" is equal to 7.67 km.

Distance between longitudes:

$$L = \Delta\lambda \times (\pi / 180) \times r \times \cos(\phi)$$

"Δλ" is angle difference between longitudes in degrees which equals to 2.03° in the example in Fig. 29.
"ϕ" is latitude which is 16.533° in the example in Fig. 29.
"r" is radius of the Moon which is 1,737.4 km.
Distance between longitudes (Δϕ=1°) "L" is equal to 59 km.

Slople of the straight line:

$$m = D / L$$

"D" is distance between latitudes.
"L" is distance between longitudes.
Slope of the straight line "m" equals to 0.13.

Inclination of the straight line:

$$\alpha = arctan(m)$$

“m” is slope od the line.
“α” equals to 7.45°.

If the camera is looking at the lunar surface with a yaw axis angle of 0°, the slope of the line drawn in the camera image will be equal to the calculated value, that is, 7.45° in the example above. However, if the camera and therefore the satellite have an angle in the yaw axis, the angle calculated between the craters and the angle taken from the camera image will deviate from each other by the angle in the yaw axis. Yaw angle of the camera therefore satellite can be determined with this approximation. Last but not least, there is also another possibility that might occur. If yaw angle is greater than 180°, the detected angle should be added on 180°. To determine whether the angle is greater than 180°, latitudes of the craters can be used again. If pixel values and latitude values of the craters are conflicting, the yaw axis is greater than 180°. For example, let the latitude of crater A is greater than latitude crater B. However, pixel value in the y axis of crater A is smaller than pixel value in the y axis of crater B. In that case, camera looking at the surface upside down which means the yaw angle is greater than 180°.

As a result attitude determination in yaw axis of the satellite can be calculated according to following algorithm:

1) Detect the craters and assign their global coordinates.
2) Draw a straight line passing through craters’ centers.
3) Find the slope of the line and angle between the equator of Moon and drawn line.
4) Determine whether the yaw angle is greater than 180°.
5) If yaw angle is less than 180°, find the yaw angle by taking the difference between calculated angle in step 3 and drawn angle in the camera frame.
6) If yaw angle is greater than 180°, add 180° to the result that found in the step 5.

Important Note: The approach mentioned above is valid when craters are relatively close to each other. Otherwise, the margin of error will become very large since the surface is spherical. However, since the mission requires looking at the Moon's surface in a very narrow area. Otherwise, as mentioned before, the triad formation and crater catalogue comparison processes would be very cumbersome and error-prone. Thus, there is no harm in applying this approach.

Yaw axis determination can be done as mentioned. Roll and pitch angles of the satellite is assumed to be 0° due to the requirement of the perpendicular point of view for crater identifications and triad comparison algorithms. Otherwise they will become error-prone. Subsequently, circles become ellipses when the point of view is not perpendicular which makes the Hough Circle method invalid. The alternative ways to determination of craters from different point of views are possible but not easy as Hough Circle method. This study focuses on the general structure of the mission. Thus, craters that are circles but look like ellipses due to the point of view are not interested as a special case for simplicity. To sum up, roll and pitch angles are assumed to be 0° and yaw angle can be determined according to straight line approximation mentioned above.

8. Overall Mission Overview

The overall process is schemed in Fig. 30.

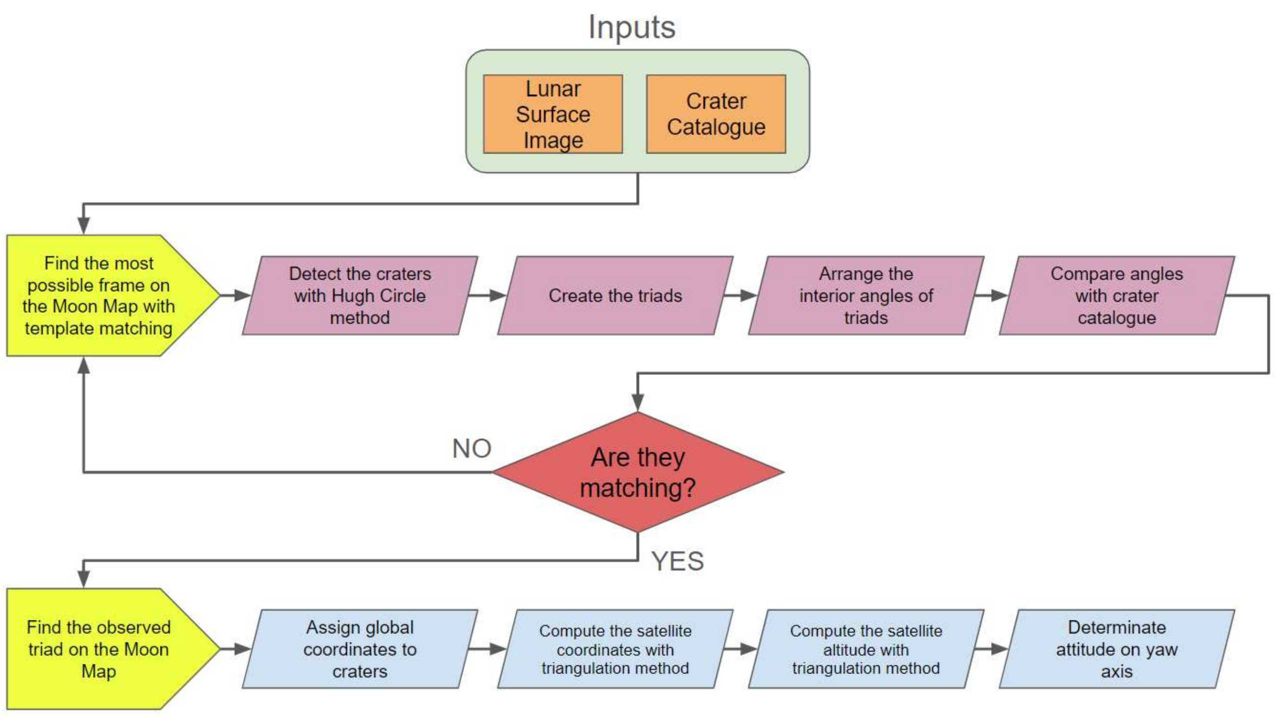

Inputs
Lunar Surface Image
Crater Catalogue
Find the most possible frame on the Moon Map with template matching
Detect the craters with Hugh Circle method
Create the triads
Arrange the interior angles of triads
Compare angles with crater catalogue
Are they matching?
NO
YES
Find the observed triad on the Moon Map
Assign global coordinates to craters
Compute the satellite coordinates with triangulation method
Compute the satellite altitude with triangulation method
Determinate attitude on yaw axis

Figure 30: Overall Mission Scheme.

V.Results and Conclusion

In summary, this study represents a comprehensive effort aimed at exploring the understanding of lunar surface analysis and satellite positioning through a multifaceted exploration of detection methodologies. By harnessing the power of cutting-edge algorithms and innovative techniques within the immersive environment of Unity, a good framework for simulating and scrutinizing the lunar landscape have established.

Through the integration of Unity's advanced 3D modeling and visualization capabilities, this research has explored the possible ways of lunar satellite positioning missions. The utilization of well-established algorithms in idealized Unity conditions provided a foundational understanding of their performance, while the transition to non-idealized conditions using satellite imagery enabled a more realistic assessment. This iterative approach not only enhances the fidelity of the simulations but also underscores the adaptability and versatility of the methodology.

One of the key contributions of this work lies in its ability to address challenges unique to lunar exploration. Distortions in polar regions of Moon and refining crater detection processes on uniform colored areas, these efforts are crucial for overcoming the inherent complexities of lunar missions and giving motivations for the future exploration endeavors.

The implementation of complicated algorithms such as CDA and CDI adds depth to the analysis, offering a good insight into satellite positioning and attitude determination. However, for simplicity some disruptive factors are eliminated such as shadows, reflections, optical problems, different point of views to surface, polar region distortions etc.

As conclusion of this research, it is essential to recognize the broader implications and potential applications of the findings. The methodologies studied in this study have far-reaching implications for space science and engineering, offering valuable insights into the dynamics of lunar exploration and satellite missions.

As a result, the following steps can be outlined in this study:

- Modeling a 3D lunar surface in Unity and assigning global coordinates (georeferencing) to the Moon.
- Placing a satellite in orbit around the Moon and observing its surface with the satellite camera.
- Detecting craters on the lunar surface using Edge Detection and Hough Circle Detection methods.
- Grouping these craters into sets of three (triads) and storing the interior angles of all triad combinations.
- Matching craters from the catalog based on the stored angles' magnitudes and sequences.
- Utilizing the Template Matching method to reduce the scanned area in the catalog for easier and more effective catalog scanning.
- Assigning latitude and longitude to each crater after matching them from the catalog.
- Calculating the satellite altitude by using the coordinates of a point on the displayed lunar surface and the projection point coordinates of the satellite.
- Implementing the triangulation method using the assigned latitude and longitude data of two craters, along with their actual positions and the apparent positions on the camera's display, to determine the satellite's 3D location.
- Determining the attitude of the satellite on yaw axis by using straight line approximation.

In this study the followings are discussed and searched:

- Distortions may occur when a map is superimposed on a globe especially on the poles. 3D Moon model on Unity has also distortions on its poles. However, they can be fixed by using Blender program. In this study, distortions were neglected and equatorial regions were selected to be studied.
- What is the best image processing method for detecting craters on the lunar surface? Concepts such as Edge Detection, Corner Detection, Contour Detection, Feature Matching, Template Matching, Image Thresholding, Blurring, and Smoothing and Color Mapping were investigated, with some serving as auxiliary elements and others as fundamental elements.
- The Canny Edge Detection algorithm and the Sobel Edge Detection algorithm were explored, and the Canny algorithm was chosen.
- SIFT, SURF, and ORB algorithms were investigated but not preferred due to their expected slowness and unnecessary complexity.
- To assist in edge detection, the color map of the photograph was set to gray in order to enhancing the clarity of the edges with smoother transitions.
- Thresholding was applied within the Canny algorithm and optimal values were determined for edge detection.
- The Hough Circle method was employed to produce circles from the obtained edges and detect craters, with parameter adjustments aimed at achieving optimal results.
- Template matching was utilized to facilitate catalog scanning. It is reducing processor load and minimizing potential errors by constraining the scanned area. Six different methods were tested, revealing that the TM_SQDIFF and TM_SQDIFF_NORMED methods in the OpenCV library exhibited significantly higher accuracy compared to other methods for lunar surface examples.
- The use of the Triangulation method was investigated, with discussions on its two types: Resection and Intersection. It was determined that Intersection was the suitable method for our study, and the reasons were deliberated.
- To ensure accurate crater identification and triad comparison algorithms, the satellite's roll and pitch angles are assumed to be 0°. This assumption is necessary because any deviation would result in a non-perpendicular viewpoint, causing circles to appear as ellipses and rendering the Hough Circle method ineffective. While there are alternative methods to detect craters from different viewpoints, they are more complex than the Hough Circle method. This study aims to focus on the overall mission structure; therefore, craters that appear elliptical due to the angle of view are considered beyond the scope for simplicity.
- The straight-line approximation for attitude determination is appropriate when craters are relatively close together. If they are not, the margin of error increases significantly due to the spherical nature of the surface. However, since the mission involves observing a very narrow area of the Moon's surface, this issue is mitigated. As previously mentioned, wider areas would complicate the triad formation and crater catalog comparison processes, making them cumbersome and error-prone. Therefore, applying this approach is justified.

VI.Acknowledgments

I would like to express my sincere gratitude to my friend Burak Ay for his consistent support and valuable assistance throughout my undergraduate studies. His encouragements and insights have been greatly appreciated. Thank you, Burak, for being a dependable and inspiring friend.

VII.References

[1] Z. Zhang, Y. Xu, J. Song, Q. Zhou, J. Rasol and L. Ma, "Planet Craters Detection Based on Unsupervised Domain Adaptation," in IEEE Transactions on Aerospace and Electronic Systems, vol. 59, no. 5, pp. 7140-7152, Oct. 2023, https://doi.org/10.1109/TAES.2023.3285512.

[2] Mu, L., Xian, L., Li, L., Liu, G., Chen, M., & Zhang, W. (2023). YOLO-Crater Model for Small Crater Detection. Remote Sensing, 15(20), 5040. https://doi.org/10.3390/rs15205040.

[3] Thrasher, A.; Christian, J.A.; Molina, G.; Hansen, M.; Pelgrift, J.Y.; and Nelson, D.S.; “Lunar Crater Identification using Triangle Reprojection,” AAS/AIAA Astrodynamics Specialist Conference, Big Sky, MT, August 2023.

[4] Christian, J. A., Derksen, H., & Watkins, R. (2021). Lunar crater identification in digital images. The Journal of the Astronautical Sciences, 68(4), 1056-1144.

[5] Abernot, M., Gauthier, S., Gonos, T., & Todri-Sanial, A. (2023, April). SIFT-ONN: SIFT Feature Detection Algorithm Employing ONNs for Edge Detection. In Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference (pp. 100-107). https://doi.org/10.1145/3584954.3584999.

[6] Vijayarani, S., & Vinupriya, M. (2013). Performance analysis of canny and sobel edge detection algorithms in image mining. *International Journal of Innovative Research in Computer and Communication Engineering*, *1*(8), 1760-1767.

[7] Liu, H., Qian, Y., & Lin, S. (2010, May). Detecting persons using hough circle transform in surveillance video. In *International Conference on Computer Vision Theory and Applications* (Vol. 2, pp. 267-270). SCITEPRESS.

[8] Borovicka, J. (2003). Circle detection using hough transforms documentation. *COMS30121-Image Processing and Computer Vision*, *48*.

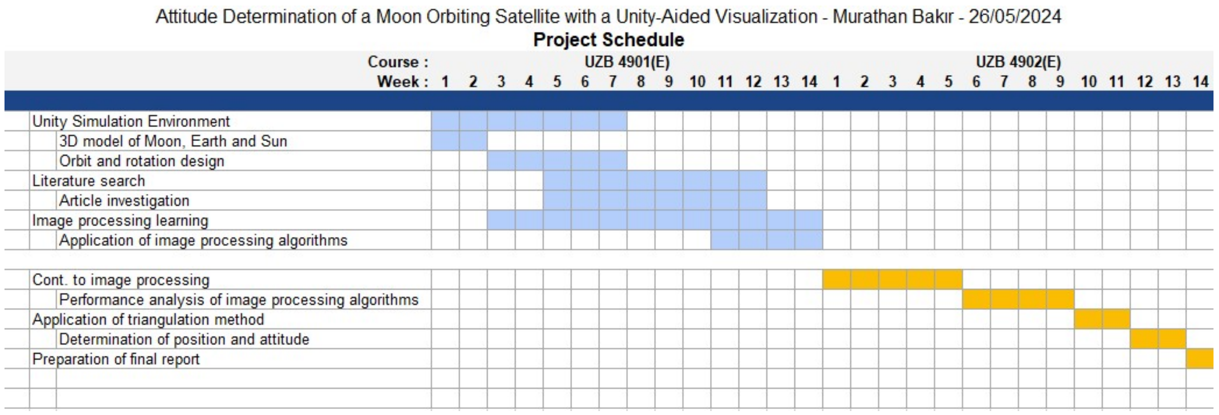
[9] Steven L Tanimoto, Template matching in pyramids, Computer Graphics and Image Processing, Volume 16, Issue 4, 1981, Pages 356-369, ISSN 0146-664X, https://doi.org/10.1016/0146-664X(81)90046-0.

[10] Wright, E. (2019, September 6). *Planets and Moons*. NASA SVS: CGI Moon Kit. Retrieved January 6, 2024, from https://svs.gsfc.nasa.gov/cgi-bin/details.cgi?aid=4720.

[11] Curtis, H. D. (2020). Orbital mechanics for engineering students: Revised Reprint. Butterworth-Heinemann.

VIII.Appendices

A. Project Schedule



Attitude Determination of a Moon Orbiting Satellite with a Unity-Aided Visualization - Murathan Bakır - 26/05/2024

Project Schedule

Course :	UZB 4901(E)														UZB 4902(E)													
Week :	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Unity Simulation Environment	🟦	🟦	🟦	🟦	🟦	🟦	🟦																					
3D model of Moon, Earth and Sun	🟦	🟦																										
Orbit and rotation design			🟦	🟦	🟦	🟦	🟦																					
Literature search					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦																
Article investigation					🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦																
Image processing learning			🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦														
Application of image processing algorithms											🟦	🟦	🟦	🟦														
Cont. to image processing															🟨	🟨	🟨	🟨	🟨									
Performance analysis of image processing algorithms																				🟨	🟨	🟨	🟨					
Application of triangulation method																								🟨	🟨			
Determination of position and attitude																										🟨	🟨	
Preparation of final report																												🟨

B. Statement of Ethics

I officially certify that all content in the thesis was obtained and presented in line with academic standards of ethics and ethical behavior, that all non-original sources utilized in this study were properly acknowledged, and that I take complete legal obligation in the event that the opposite occurs.

Murathan Bakır

[illegible]