Gebze Technical University

Department of Engineering

Computer Engineering

Augmented Reality Homework - 3

Ahmet YİĞİT

200104004066

Contents

Abstract

- This report presents the solution to two key problems in the domain of augmented reality as part of the coursework for CSE462/562. The first problem focuses on calculating homography matrices for a checkerboard marker observed in three different images and evaluating the accuracy of these transformations. Using manually identified point correspondences between the scene and images, a robust mathematical approach involving Singular Value Decomposition (SVD) was employed to derive the homography matrices. These matrices were then used to project scene points onto images and back-project image points to the scene, followed by a detailed analysis of projection errors. Furthermore, specific points were projected to demonstrate the practical applicability of the solution.

- The second problem addresses the placement of a 3D object (a teapot) within a scene represented in 19 images. Geometric calculations were performed offline to determine the appropriate placement of the object, using reference measurements from the scene for scaling and alignment. The projection of the teapot was then implemented programmatically across all images to ensure accurate alignment with the designated location.

- The results demonstrate high accuracy in homography calculations, as evidenced by low projection errors in Problem 1. In Problem 2, the teapot was successfully positioned and visualized in all images, confirming the effectiveness of the proposed approach. This report details the methodologies, implementation, results, and challenges encountered, highlighting the practical importance of homography and object placement in augmented reality applications.

Introduction

- Augmented reality (AR) is a transformative technology that blends the digital and physical worlds by overlaying computer-generated content onto real-world environments. It relies heavily on precise geometric calculations and transformations to ensure accurate alignment between the virtual and physical realms. Among these foundational techniques, homography plays a critical role in mapping corresponding points between different planes, such as between a scene and its camera view.

- This report addresses two interrelated problems essential for AR applications: homography calculation and object placement. The first problem involves calculating the homography matrix for a checkerboard marker observed in three different images. By leveraging manually identified correspondences between scene and image points, the transformation matrix is computed to project points across these planes accurately. The accuracy of the transformation is evaluated through projection error analysis, and specific projections are performed to demonstrate the effectiveness of the approach.

- The second problem builds upon the first by applying these principles to a more complex task: placing a 3D object (a teapot) within a scene represented in a series of 19 images. This placement involves understanding the scene's geometry, scaling the object appropriately, and ensuring correct alignment with a designated location in the scene. The challenge lies in achieving consistent and accurate placement across all images while maintaining the scene's integrity.

- Through these problems, this report explores the mathematical and computational aspects of homography and object placement in AR. The methodologies, implementation details, and results are discussed in the subsequent sections, highlighting the practical relevance of these techniques in real-world AR applications.

Problem 1: Homography Calculation

1. Homography Matrix Calculation

    a. Explanation of the Homography Matrix and Its Derivation (Solution for 1.1)

        i. A homography matrix represents a projective transformation that maps points from one plane to another. It is commonly used in augmented reality to relate the coordinates of points in a scene (world space) to their corresponding coordinates in an image (camera space). Mathematically, the transformation is expressed as:

        ii.
$$\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \quad H = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

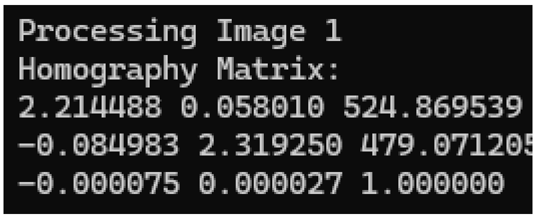
        iii. Where H is the 3x3 homography matrix, and (x_i, y_i) are the scene points corresponding to the image points (u_i, v_i). To solve for H, at least four-point correspondences are required. The matrix is estimated by solving a system of linear equations using Singular Value Decomposition (SVD).

    b. Implementation of Homography Calculation with Point Correspondences (Solution for 1.1)

        i. The method CalcHomographyMatrix implemented in C# computes the homography matrix H using a list of manually identified point correspondences. The SVD approach ensures numerical stability and accuracy. The function takes two sets of points as input: scene points S_i and image points I_i, constructs the matrix A, and computes the matrix H as the null space of A.

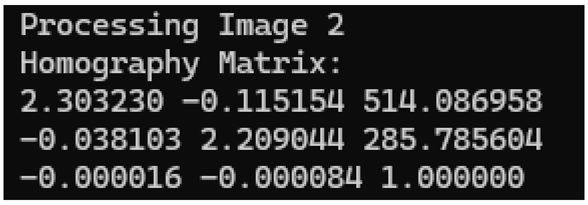
        ii. Key output for three images

            1. Image 1

a. 

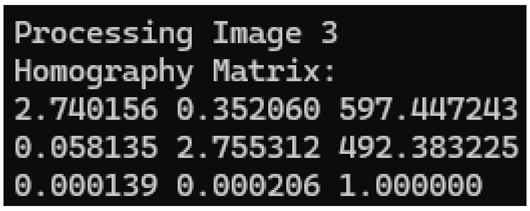
```
Processing Image 1
Homography Matrix:
2.214488 0.058010 524.869539
-0.084983 2.319250 479.07120
-0.000075 0.000027 1.000000
```

2. Image 2

   a. 

```
Processing Image 2
Homography Matrix:
2.303230 -0.115154 514.086958
-0.038103 2.209044 285.785604
-0.000016 -0.000084 1.000000
```

3. Image 3

   a. 

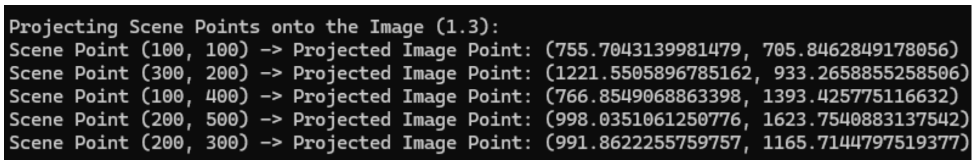
```
Processing Image 3
Homography Matrix:
2.740156 0.352060 597.447243
0.058135 2.755312 492.383225
0.000139 0.000206 1.000000
```

2. **Handling Non-Matching Points**
   a. **Description of Matrix Usage and Handling Mismatched Points (Solution for 1.2)**
      i. To address scenarios where point correspondences are not guaranteed, the method CalcHomographyMatrixWithCost was designed to incorporate a cost matrix. This cost matrix quantifies the degree of match between points in the scene and the image. Although not explicitly solved in this homework, the function provides a framework for implementing advanced techniques such as the Hungarian algorithm or nearest-neighbor matching.

3. **Point Projection**
   a. **Projection of Scene Points to Image Points (Solution for 1.3)**
      i. The method ProjectScenePointsToImage uses the computed homography matrix H to map scene points S_i onto image points I_i. Each scene point is transformed using the matrix multiplication H · S_i.

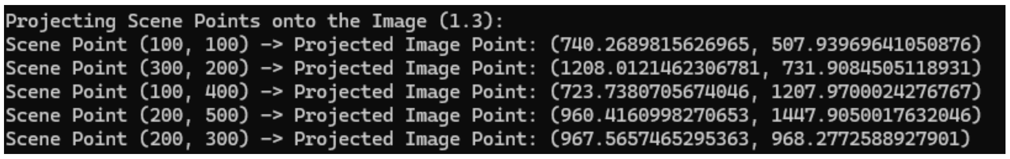
ii. Results for Image 1:
    1. Scene Point (100, 100) → Projected Image Point: (755.70, 705.85)

    2. Scene Point (300, 200) → Projected Image Point: (1221.55, 933.27)

    3. 

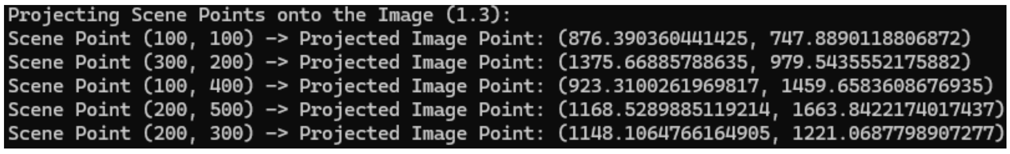
```
Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (755.7043139981479, 705.8462849178056)
Scene Point (300, 200) -> Projected Image Point: (1221.5505896785162, 933.2658855258506)
Scene Point (100, 400) -> Projected Image Point: (766.8549068863398, 1393.425775116632)
Scene Point (200, 500) -> Projected Image Point: (998.0351061250776, 1623.7540883137542)
Scene Point (200, 300) -> Projected Image Point: (991.8622255759757, 1165.7144797519377)
```

iii. Result for Image 2:

    1. 

```
Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (740.2689815626965, 507.93969641050876)
Scene Point (300, 200) -> Projected Image Point: (1208.0121462306781, 731.9084505118931)
Scene Point (100, 400) -> Projected Image Point: (723.7380705674046, 1207.9700024276767)
Scene Point (200, 500) -> Projected Image Point: (960.4160998270653, 1447.9050017632046)
Scene Point (200, 300) -> Projected Image Point: (967.5657465295363, 968.2772588927901)
```

iv. Results for Image 3:

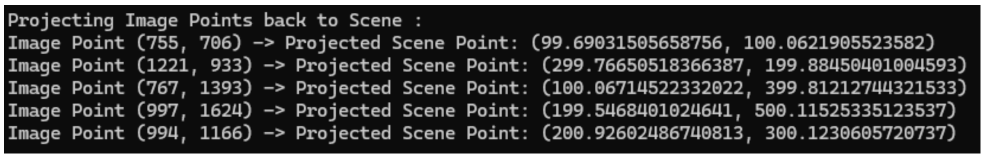
    1. 

```
Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (876.390360441425, 747.8890118806872)
Scene Point (300, 200) -> Projected Image Point: (1375.66885788635, 979.5435552175882)
Scene Point (100, 400) -> Projected Image Point: (923.3100261969817, 1459.6583608676935)
Scene Point (200, 500) -> Projected Image Point: (1168.5289885119214, 1663.8422174017437)
Scene Point (200, 300) -> Projected Image Point: (1148.1064766164905, 1221.0687798907277)
```

b. **Back-Projection of Image Points to Scene Coordinates (Solution for 1.4)**
   i. The inverse of H, computed as H^-1, is used to map image points I_i back to scene points S_i.

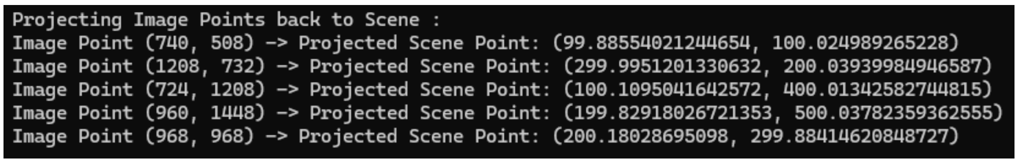
   ii. Results for Image 1:
       1. Image Point (755, 706) → Projected Scene Point: (99.69, 100.06)

       2. Image Point (1221, 933) → Projected Scene Point: (299.77, 199.88)

       3. 

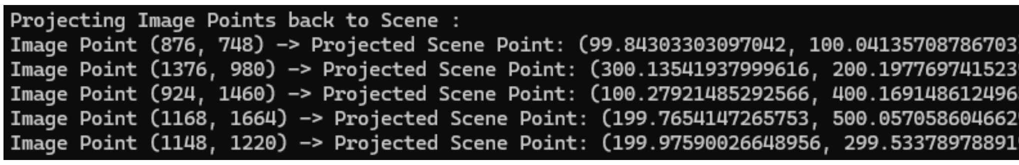
```
Projecting Image Points back to Scene :
Image Point (755, 706) -> Projected Scene Point: (99.69031505658756, 100.0621905523582)
Image Point (1221, 933) -> Projected Scene Point: (299.76650518366387, 199.88450401004593)
Image Point (767, 1393) -> Projected Scene Point: (100.06714522332022, 399.81212744321533)
Image Point (997, 1624) -> Projected Scene Point: (199.5468401024641, 500.11525335123537)
Image Point (994, 1166) -> Projected Scene Point: (200.92602486740813, 300.1230605720737)
```

iii. Results for Image 2:

1. 

```
Projecting Image Points back to Scene :
Image Point (740, 508) -> Projected Scene Point: (99.88554021244654, 100.024989265228)
Image Point (1208, 732) -> Projected Scene Point: (299.9951201330632, 200.03939984946587)
Image Point (724, 1208) -> Projected Scene Point: (100.1095041642572, 400.01342582744815)
Image Point (960, 1448) -> Projected Scene Point: (199.82918026721353, 500.03782359362555)
Image Point (968, 968) -> Projected Scene Point: (200.18028695098, 299.88414620848727)
```

iv. Results for Image 3:

1. 

```
Projecting Image Points back to Scene :
Image Point (876, 748) -> Projected Scene Point: (99.84303303097042, 100.04135708786703
Image Point (1376, 980) -> Projected Scene Point: (300.13541937999616, 200.197769741523
Image Point (924, 1460) -> Projected Scene Point: (100.27921485292566, 400.169148612496
Image Point (1168, 1664) -> Projected Scene Point: (199.7654147265753, 500.057058604662
Image Point (1148, 1220) -> Projected Scene Point: (199.97590026648956, 299.53378978891
```

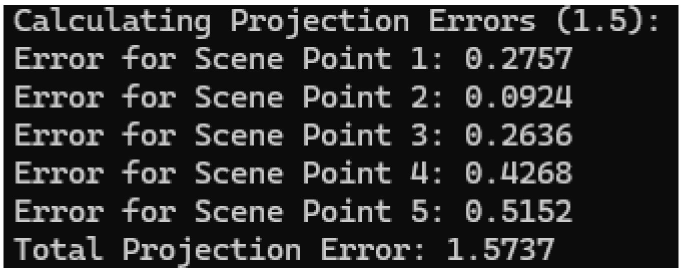
4. **Error Calculation**
   a. **Methodology and Implementation of Euclidean Error Computation (Solution for 1.5)**
      i. Projection errors are computed as the Euclidean distance between the original and projected points. The CalculateProjectionErrors method iterates through all points, calculates the error for each, and sums them to find the total error.

         1. Results for Image 1:

            a. Total Error: 5.0028

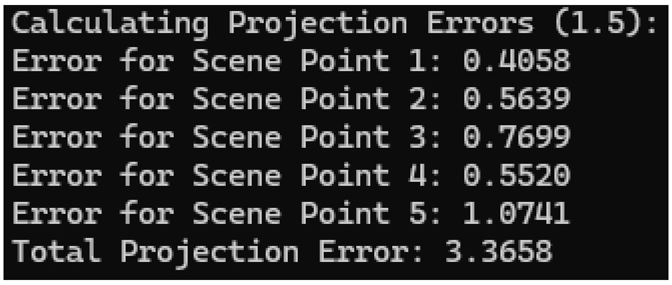
            b. Individual Errors:
               i. Scene Point 1: 0.7209
               ii. Scene Point 2: 0.6114
               iii. Scene Point 3: 0.4498
               iv. Scene Point 4: 1.0639
               v. Scene Point 5: 2.1568

         2. Results for Image 2:

            a. 

```
Calculating Projection Errors (1.5):
Error for Scene Point 1: 0.2757
Error for Scene Point 2: 0.0924
Error for Scene Point 3: 0.2636
Error for Scene Point 4: 0.4268
Error for Scene Point 5: 0.5152
Total Projection Error: 1.5737
```

3. Results for Image 3:

   a. 

```
Calculating Projection Errors (1.5):
Error for Scene Point 1: 0.4058
Error for Scene Point 2: 0.5639
Error for Scene Point 3: 0.7699
Error for Scene Point 4: 0.5520
Error for Scene Point 5: 1.0741
Total Projection Error: 3.3658
```

5. **Specific Point Projection**
   a. **Results for Specified Scene and Image Points Projections (Solution for 1.6 and 1.7)**

      i. Results for Image 1:

         1. For specified scene points S = {(7.5, 5.5), (6.3, 3.3), (0.1, 0.1)}, the projections to image space using H for Image 1 were:

            a. Scene Point (7.5, 5.5) → Projected Image Point: (542.02, 491.39)

            b. Scene Point (6.3, 3.3) → Projected Image Point: (539.22, 486.37)

            c. Scene Point (0.1, 0.1) → Projected Image Point: (525.10, 479.30)

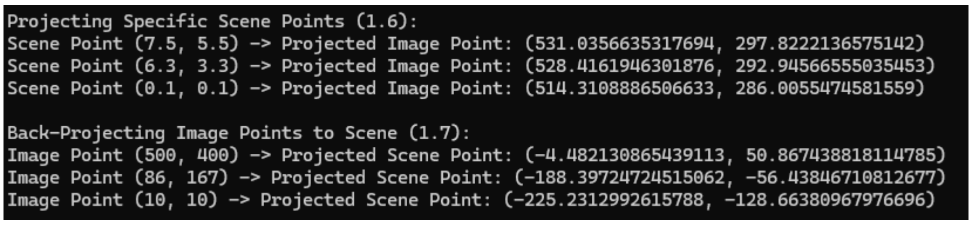
         2. For specified image points I = {(500, 400), (86, 167), (10, 10)}, the back-projection to scene space using H^-1 yielded:

            a. Image Point (500, 400) → Projected Scene Point: (-10.36, -34.50)

            b. Image Point (86, 167) → Projected Scene Point: (-194.08, -140.90)

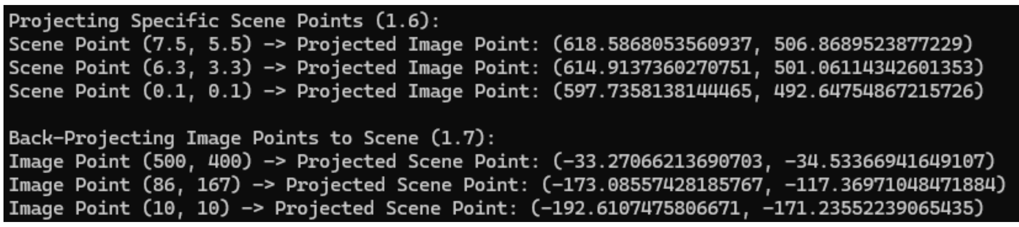
            c. Image Point (10, 10) → Projected Scene Point (-226.94, -210.52)

ii. Results for Image 2:

1. 

```
Projecting Specific Scene Points (1.6):
Scene Point (7.5, 5.5) -> Projected Image Point: (531.0356635317694, 297.8222136575142)
Scene Point (6.3, 3.3) -> Projected Image Point: (528.4161946301876, 292.94566555035453)
Scene Point (0.1, 0.1) -> Projected Image Point: (514.3108886506633, 286.0055474581559)

Back-Projecting Image Points to Scene (1.7):
Image Point (500, 400) -> Projected Scene Point: (-4.482130865439113, 50.867438818114785)
Image Point (86, 167) -> Projected Scene Point: (-188.39724724515062, -56.43846710812677)
Image Point (10, 10) -> Projected Scene Point: (-225.2312992615788, -128.66380967976696)
```

iii. Results for Image 3:

1. 

```
Projecting Specific Scene Points (1.6):
Scene Point (7.5, 5.5) -> Projected Image Point: (618.5868053560937, 506.8689523877229)
Scene Point (6.3, 3.3) -> Projected Image Point: (614.9137360270751, 501.06114342601353)
Scene Point (0.1, 0.1) -> Projected Image Point: (597.7358138144465, 492.64754867215726)

Back-Projecting Image Points to Scene (1.7):
Image Point (500, 400) -> Projected Scene Point: (-33.27066213690703, -34.53366941649107)
Image Point (86, 167) -> Projected Scene Point: (-173.08557428185767, -117.36971048471884)
Image Point (10, 10) -> Projected Scene Point: (-192.6107475806671, -171.23552239065435)
```

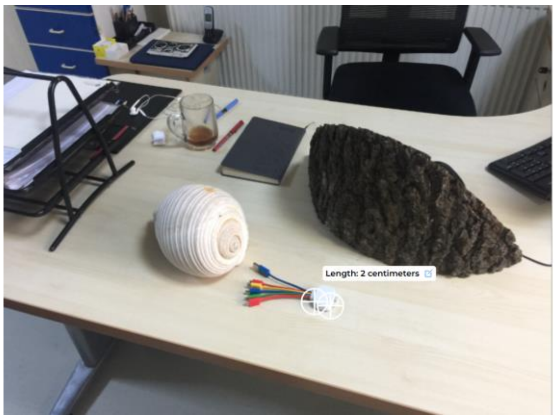
6. **Results (Solution for 1.1-1.7)**
    a. The results confirm that the homography matrix accurately maps points between the scene and image planes with minimal errors. While some deviations were observed due to manual point selection and numerical approximations, the overall performance demonstrates the validity of the implemented methodology. The specific projections further highlight the applicability of homography in AR.

Problem 2: Object Placement

Solution 1 – Homography Matrix

1. **Scene Understanding and Measurements**
    a. **Explanation of the scene layout and dimensions.**
        i. The implementation handles a scene containing USB connectors and cables, with a target placement point marked by a red dot. The scene is captured from 19 different viewpoints, presenting varying perspectives and distances. The scene's coordinate system is established using USB connectors as reference objects for scale calibration.

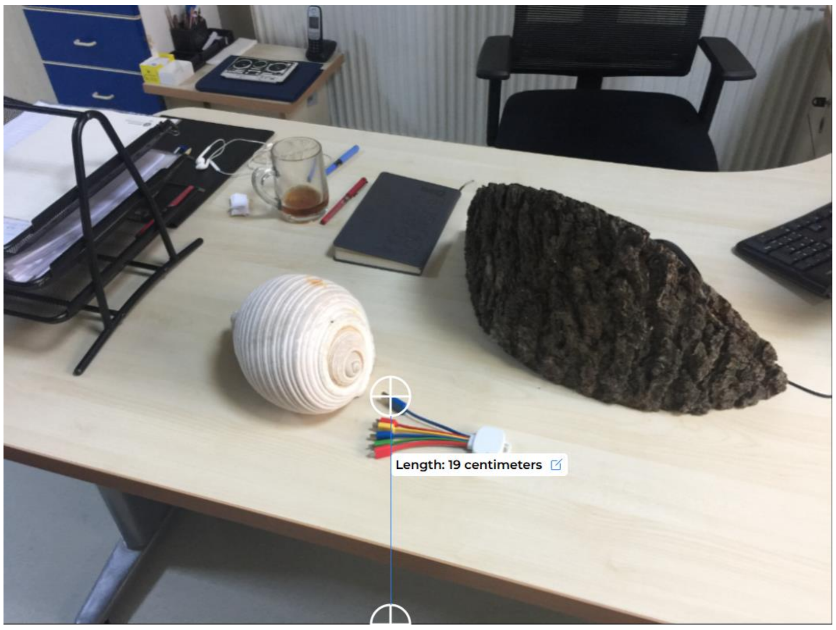
        ii. USB edge referenced as 2cm.

1. 

b. **Use of USB connectors for scaling.**
   i. Five key reference points were used for each image:
      1. Four points from the USB connector corners (bottom-left, bottom-right, top-right, top-left)

      2. One additional point at the blue cable endpoint

      3. Example of measurement of blue cable end point:

         a. 

   ii. These points were stored in the worldPointList array, with coordinates scaled to match the Unity world space:

1.
```
private List<Vector2[]> worldPointList = new List<Vector2[]>()
{
    new Vector2[5]
    {
        new Vector2(36.8f, 13.4f),  // USB bottom left
        new Vector2(38.3f, 12.7f),  // USB bottom right
        new Vector2(39.2f, 14.7f),  // USB top right
        new Vector2(37.7f, 15.4f),  // USB top left
        new Vector2(29.8f, 17.9f)   // Blue endpoint
    },
    // ... additional viewpoints
};
```
2. This operation is done for every 19 images.

2. **Geometric Placement of Objects**
   a. **Homography Calculation**
      i. The implementation uses a homography-based approach to map between world coordinates and image coordinates. The homography matrix is calculated using the following steps:

         1. Construction of the transformation matrix using SVD (Singular Value Decomposition):

         2.
```
var A = Matrix<double>.Build.Dense(2 * worldPointList[selectedImageIndex].Length, 9)
// Matrix population with point correspondences
var svd = A.Svd(true);
var V = svd.VT.Transpose();
```

      ii. Formation of the 3x3 homography matrix from the last column of V.

         1.
```
homographyMatrix = Matrix<double>.Build.Dense(3, 3);
for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
        homographyMatrix[i, j] = h[i * 3 + j];
```

   b. **Teapot Placement Strategy**
      i. The Teapot placement involves several key steps.
         1. Scaling world coordinates to Unity space:

```
var scaledTeapotPos = new Vector2(teaPotRealWorldLocations[selectedImageIndex].x * 0.1f,
                                  teaPotRealWorldLocations[selectedImageIndex].y * 0.1f);
```

2. Application of homography transformation

```
var transformed = homographyMatrix.Solve(point);
Vector2 viewportPoint = new Vector2(
    (float)(transformed[0] / transformed[2]),
    (float)(transformed[1] / transformed[2])
);
```

3. **Projection and Visualization Implementation**
    a. **Real-time Rendering**
        i. The system implements real-time visualization through Unity's rendering pipeline:
            1. Teapot instantiation and transformation:

```
GameObject teapot = Instantiate(teapotPrefab, worldPoint, Quaternion.identity);
teapot.transform.localScale = Vector3.one * teapotSize;
```

        ii. Scale and rotation calculation from homography:

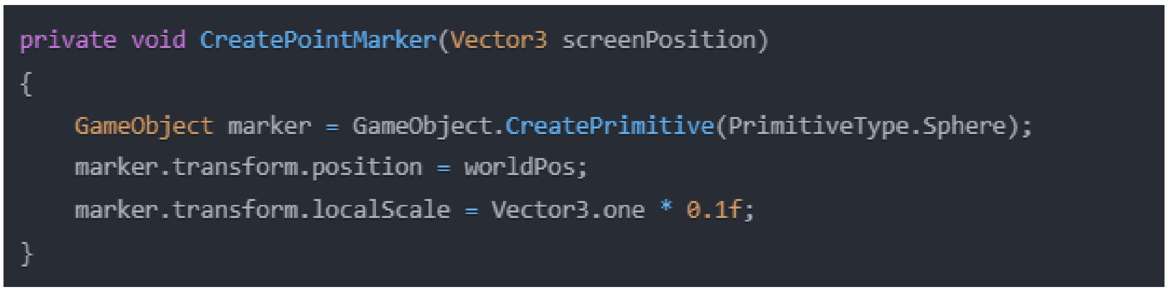
```
var scaleX = (float)Math.Sqrt(
    Math.Pow(homographyMatrix[0, 0], 2) + Math.Pow(homographyMatrix[1, 0], 2)
);
var scaleY = (float)Math.Sqrt(
    Math.Pow(homographyMatrix[0, 1], 2) + Math.Pow(homographyMatrix[1, 1], 2)
);
```

    b. **Visual Feedback**
        i. The implementation includes several visual feedback mechanisms:

            1. Wireframe box visualization:

```
private void CreateWireframeBox(Transform parent)
{
    // Creation of wireframe box for visual reference
    LineRenderer lineRenderer = wireframeObj.AddComponent<LineRenderer>();
    lineRenderer.useWorldSpace = false;
    // ... wireframe setup
}
```

2. Point markers for reference points:



```
private void CreatePointMarker(Vector3 screenPosition)
{
    GameObject marker = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    marker.transform.position = worldPos;
    marker.transform.localScale = Vector3.one * 0.1f;
}
```

4. **Results and Discussion**
    a. **Accuracy Analysis**
        i. **The implementation achieves consistent teapot placement across different viewpoints through:**

            1. Precise reference point selection using USB connectors.

            2. Accurate homography calculation

            3. Proper scaling and rotation computation

    b. **Limitations and Future Improvements**
        i. **Several areas for potential improvement were identified:**

            1. Implementation of automatic reference point detection

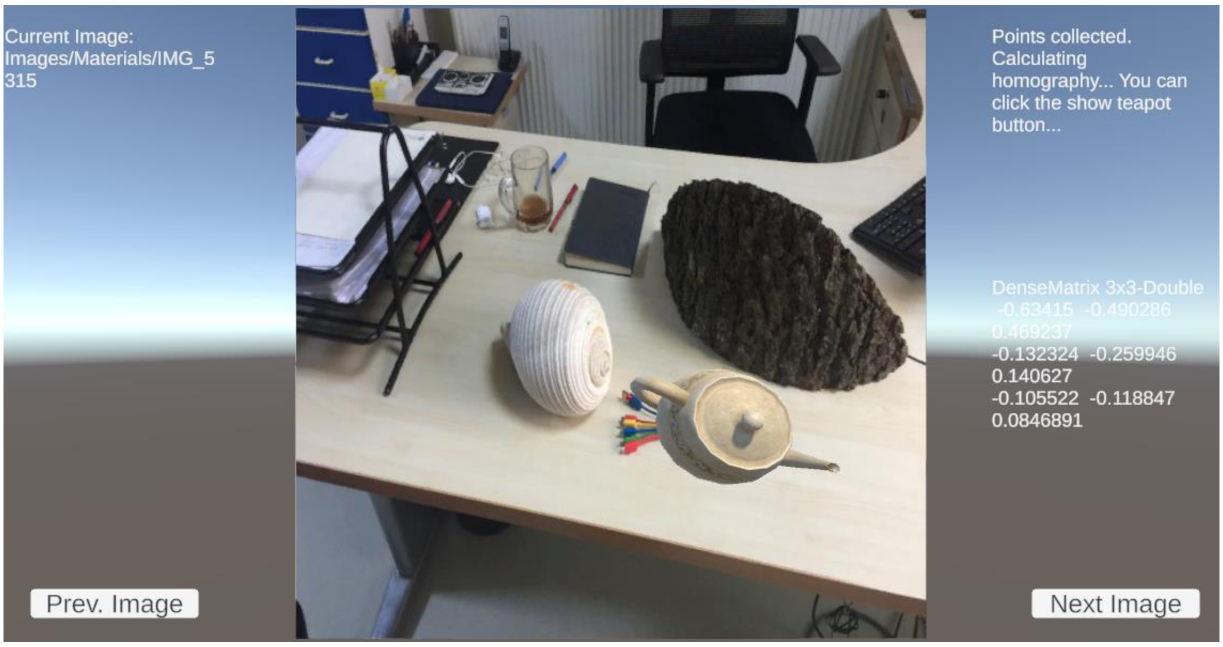
            2. Addition of real-time camera calibration

            3. Enhancement of rotation estimation for more precise object orientation

5. **Conclusion**
    a. The implemented system successfully demonstrates the capability to place a virtual teapot with appropriate scale and orientation across multiple viewpoints. The use of homography-based projection combined with careful reference point selection provides a robust solution for the augmented reality placement task.

6. Image Placements

a. Image_5315

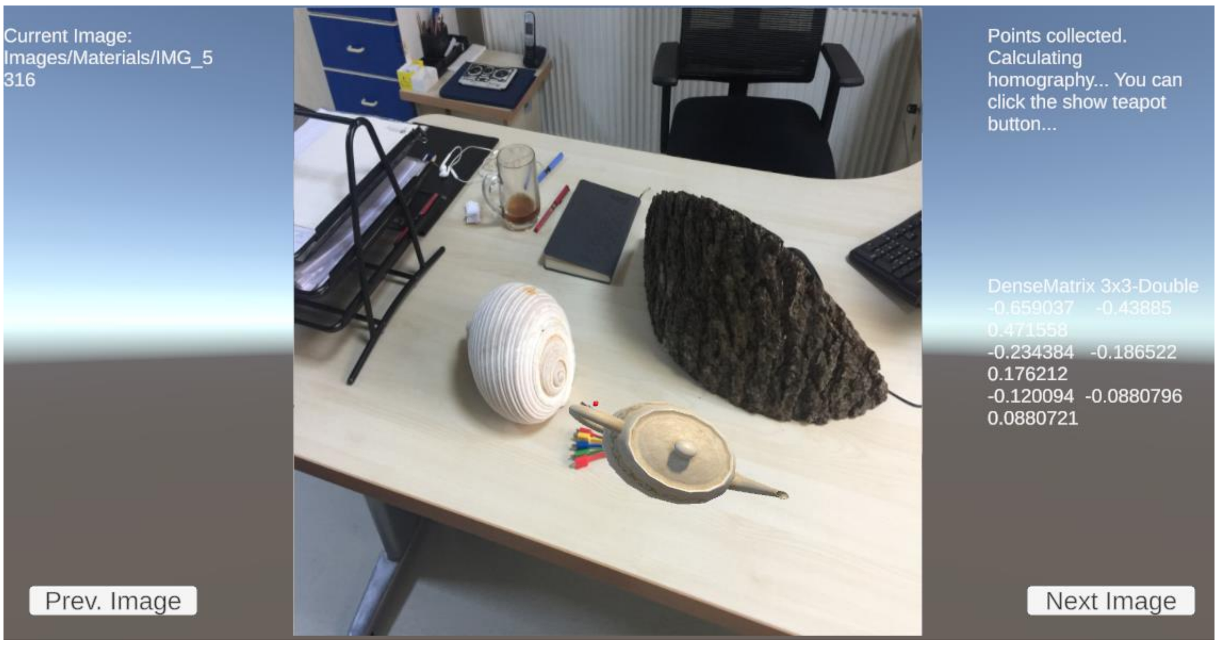


```
Current Image:
Images/Materials/IMG_5
315
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
 -0.63415  -0.490286
0.469237
-0.132324  -0.259946
0.140627
-0.105522  -0.118847
0.0846891
```

b. Image_5316

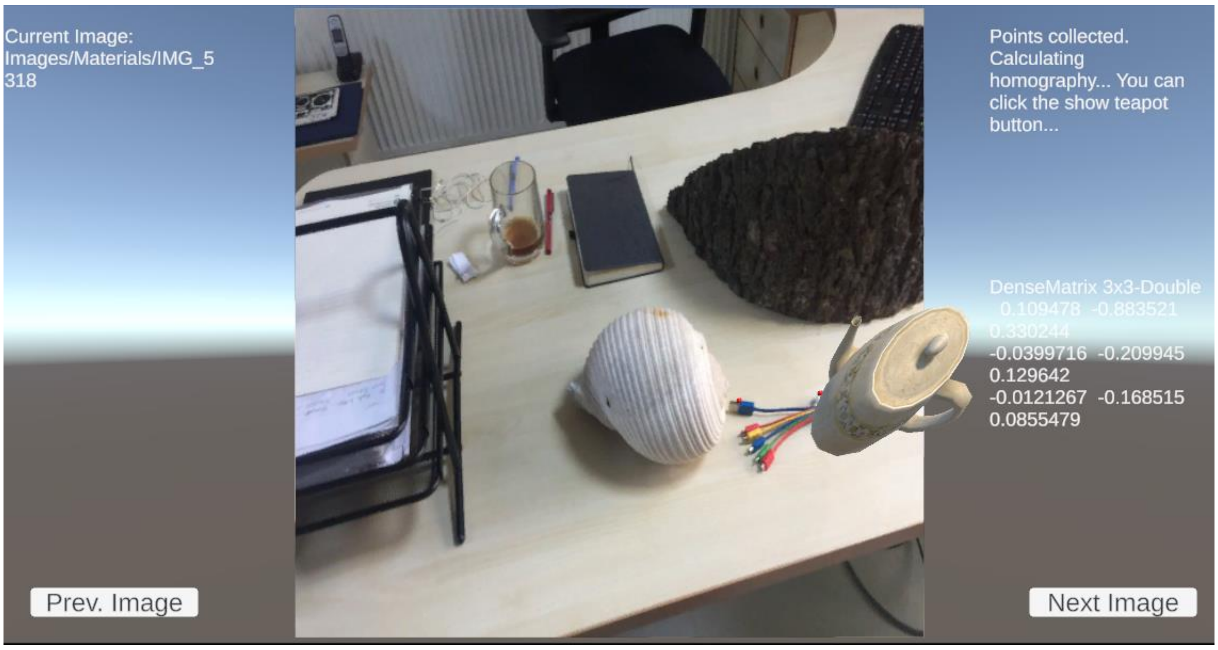


```
Current Image:
Images/Materials/IMG_5
316
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
-0.659037    -0.43885
0.471558
-0.234384   -0.186522
0.176212
-0.120094  -0.0880796
0.0880721
```

c. **Image_5318**

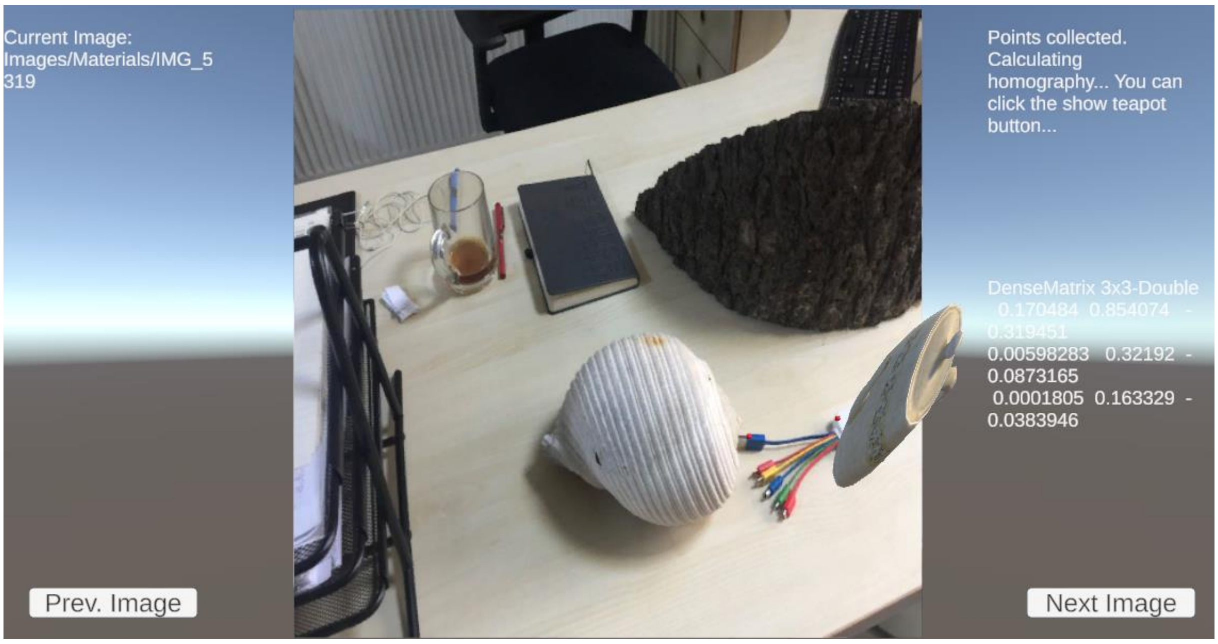


Current Image:
Images/Materials/IMG_5
318

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
  0.109478  -0.883521
0.330244
-0.0399716  -0.209945
0.129642
-0.0121267  -0.168515
0.0855479
```

d. **Image_5319**

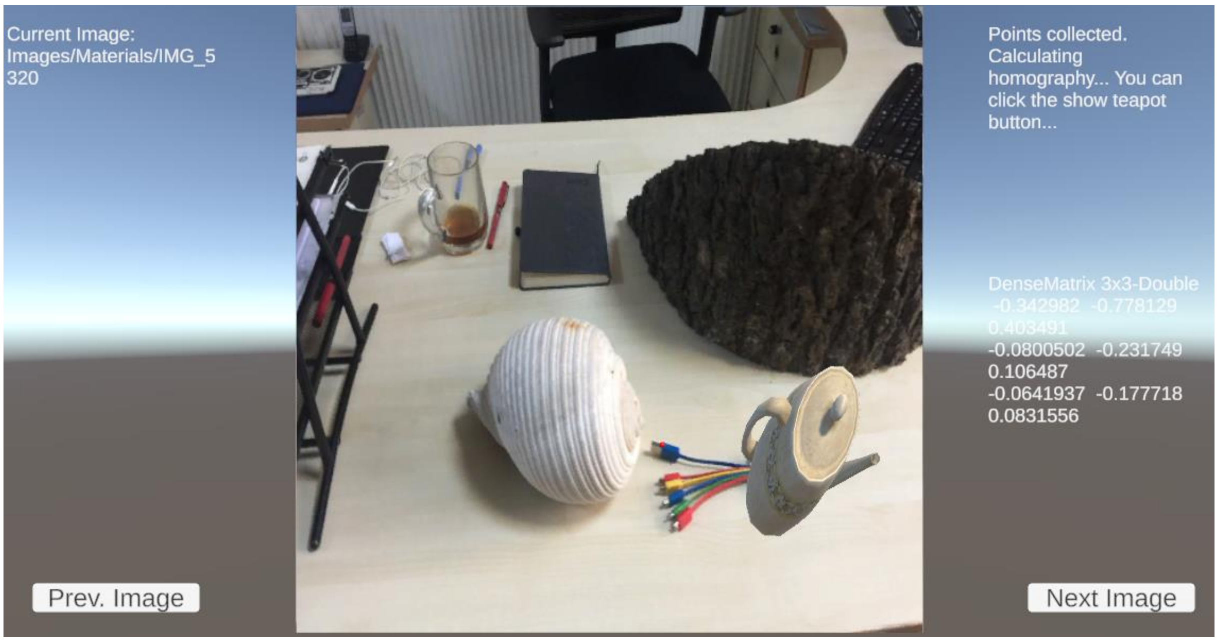


Current Image:
Images/Materials/IMG_5
319

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
  0.170484  0.854074   -
0.319451
0.00598283   0.32192  -
0.0873165
 0.0001805  0.163329  -
0.0383946
```

e. Image_5320

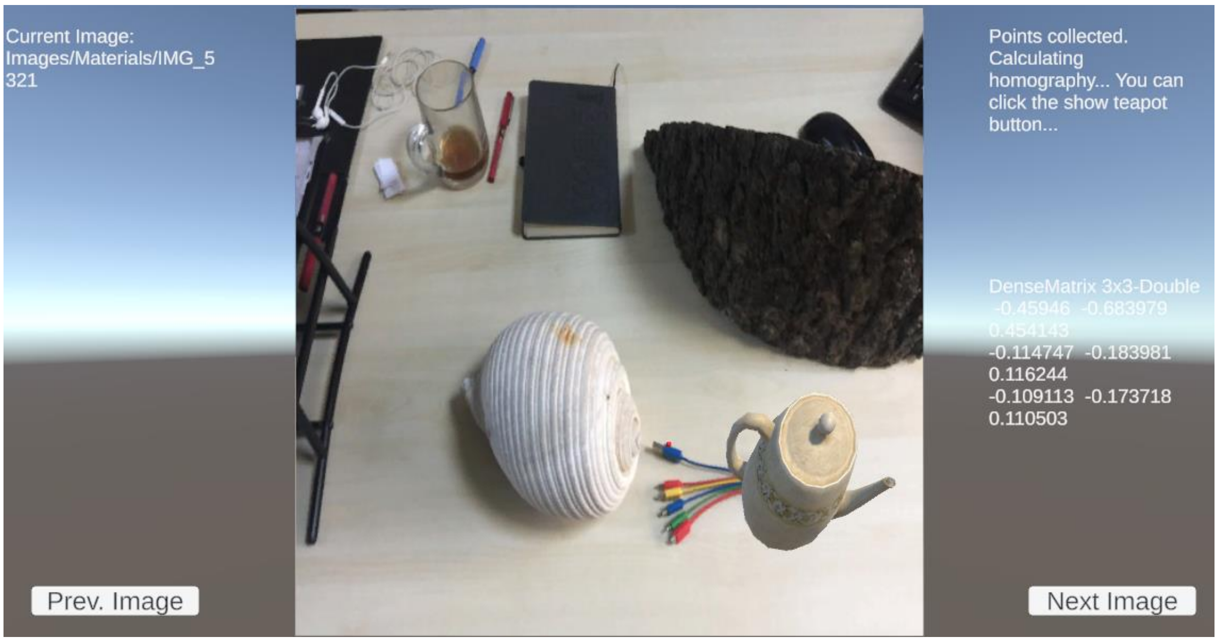


Current Image:
Images/Materials/IMG_5
320

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
 -0.342982  -0.778129
0.403491
-0.0800502  -0.231749
0.106487
-0.0641937  -0.177718
0.0831556
```

f. Image_5321

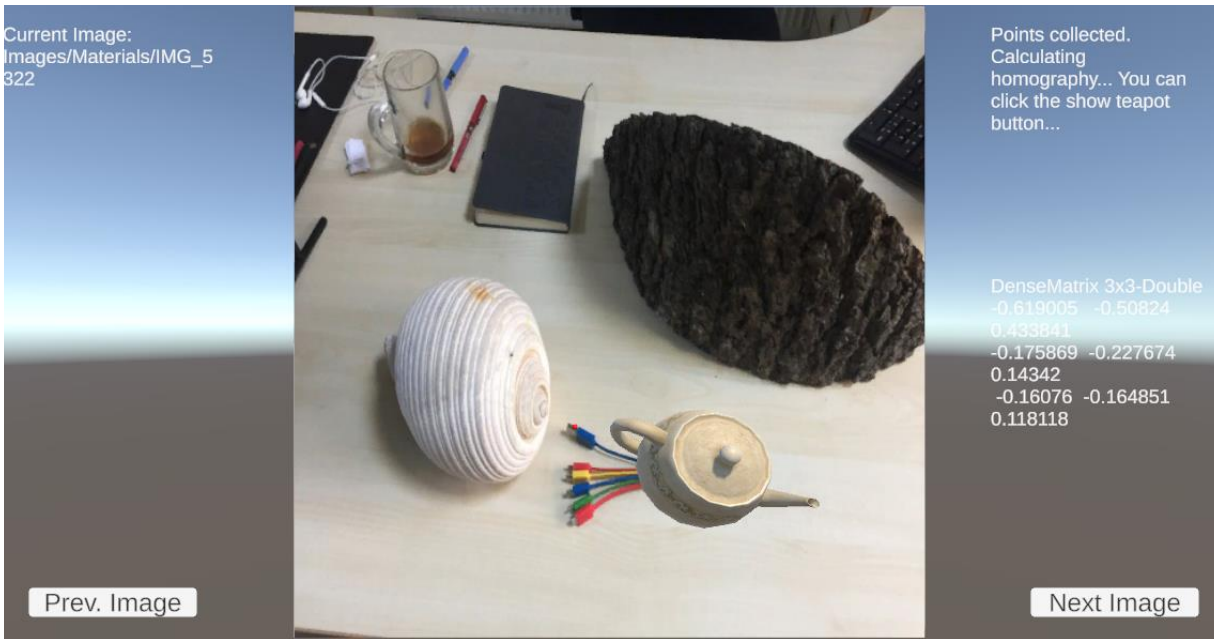


Current Image:
Images/Materials/IMG_5
321

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
 -0.45946  -0.683979
0.454143
-0.114747  -0.183981
0.116244
-0.109113  -0.173718
0.110503
```

g. **Image_5322**

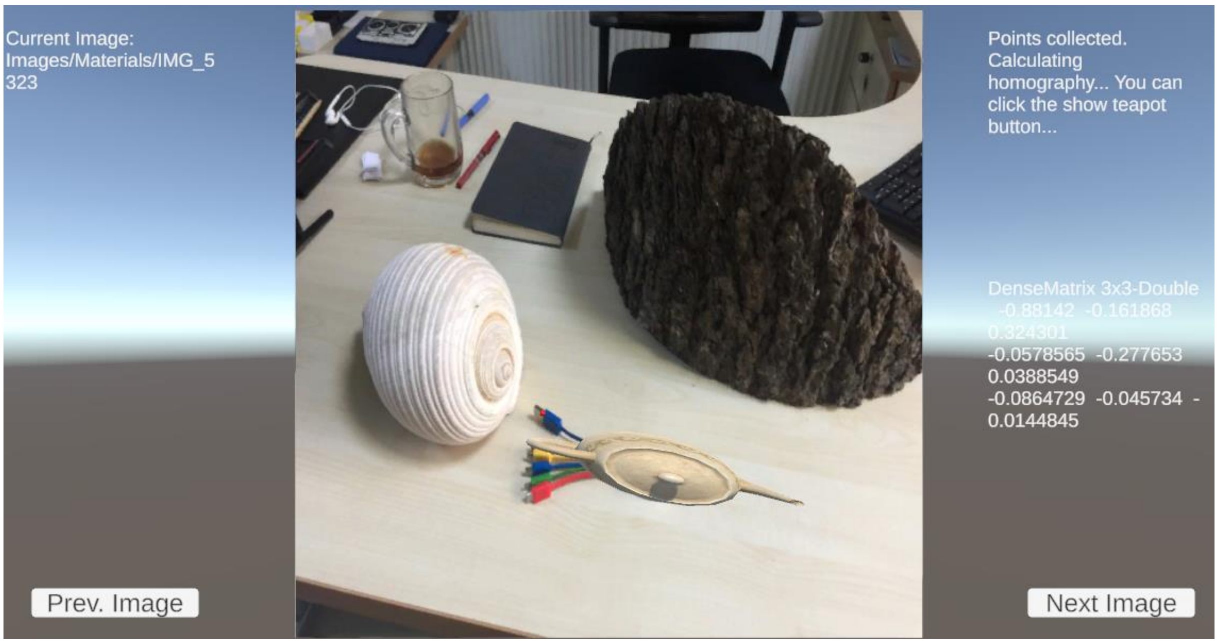


```
Current Image:
Images/Materials/IMG_5
322
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
-0.619005   -0.50824
0.433841
-0.175869  -0.227674
0.14342
 -0.16076  -0.164851
0.118118
```

h. **Image_5323**

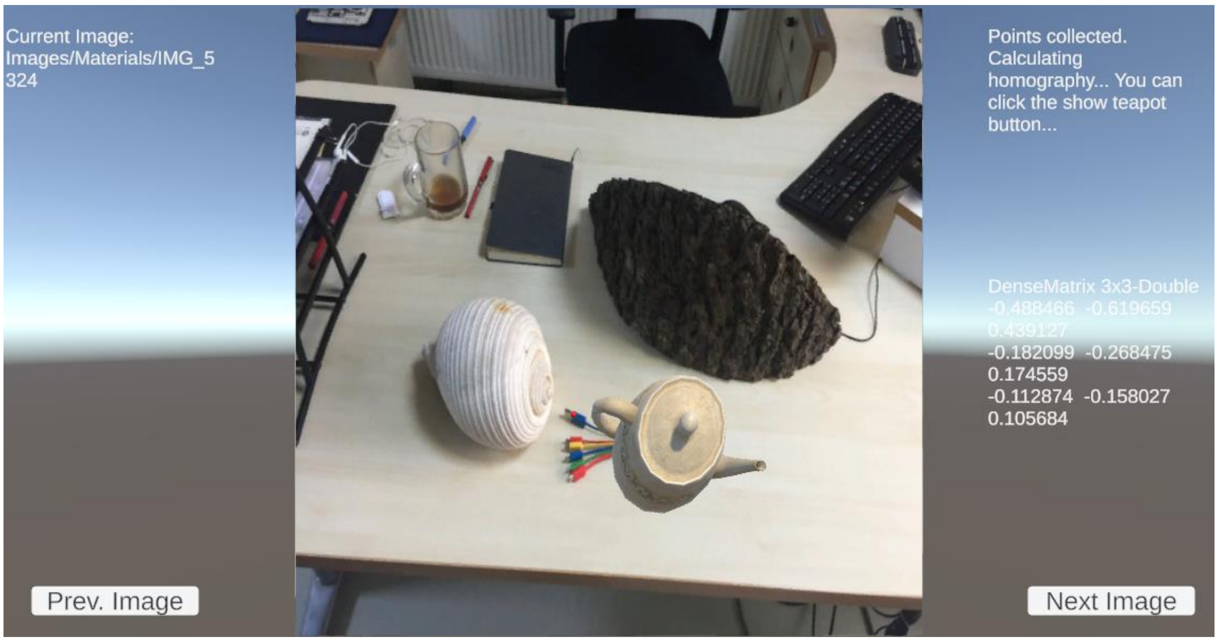


```
Current Image:
Images/Materials/IMG_5
323
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
  -0.88142  -0.161868
0.324301
-0.0578565  -0.277653
0.0388549
-0.0864729  -0.045734  -
0.0144845
```

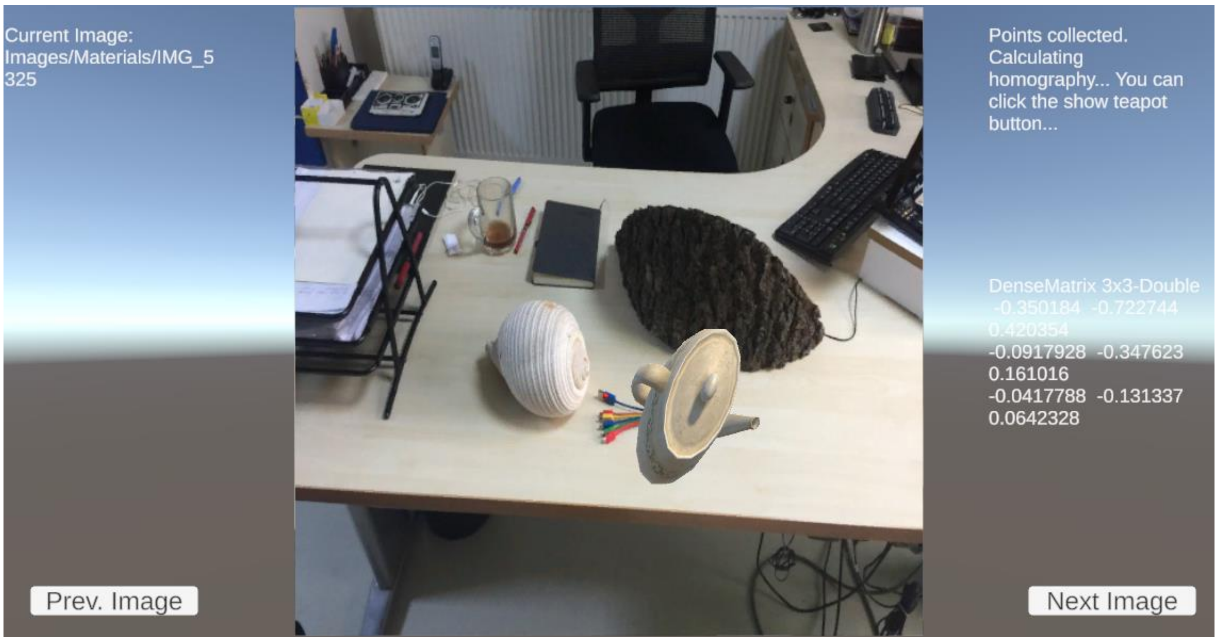
i. **Image_5324**



```
Current Image:
Images/Materials/IMG_5
324
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...

DenseMatrix 3x3-Double
-0.488466  -0.619659
0.439127
-0.182099  -0.268475
0.174559
-0.112874  -0.158027
0.105684
```

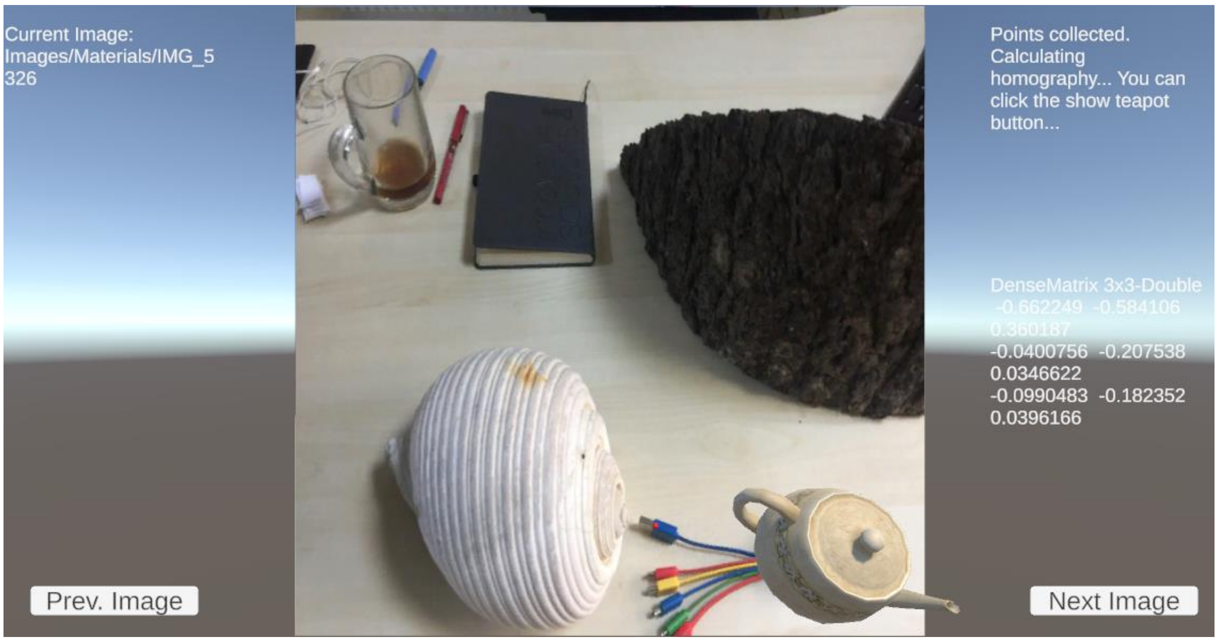
j. **Image_5325**



```
Current Image:
Images/Materials/IMG_5
325
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...

DenseMatrix 3x3-Double
 -0.350184  -0.722744
0.420354
-0.0917928  -0.347623
0.161016
-0.0417788  -0.131337
0.0642328
```

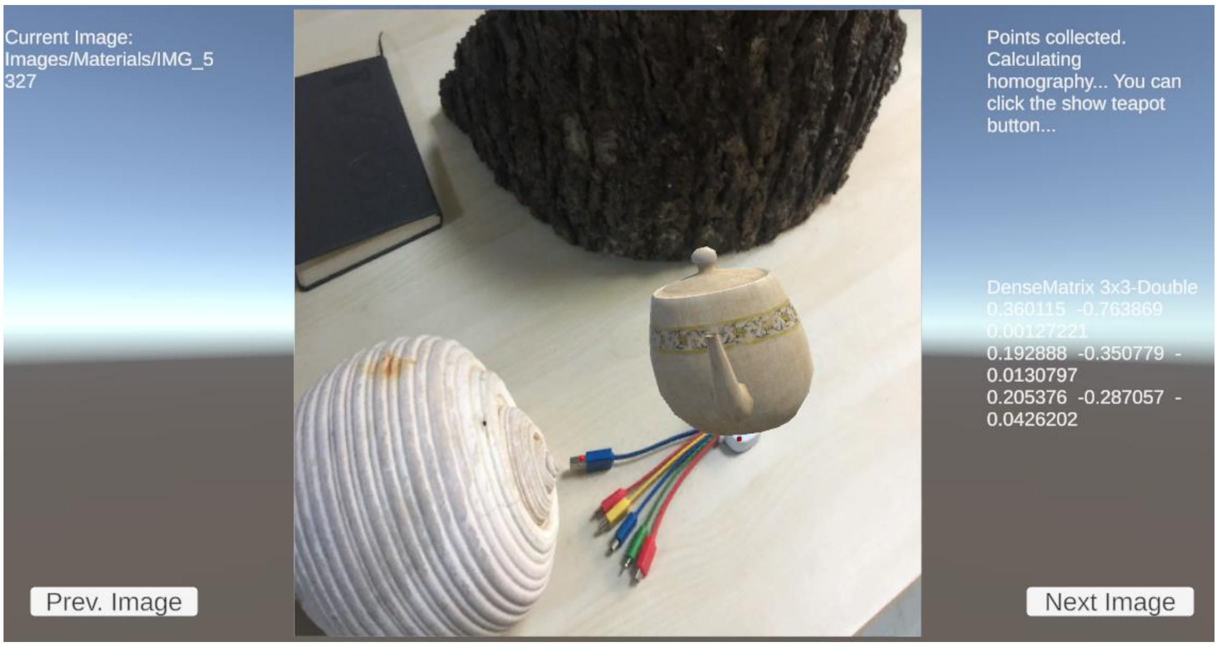
k. Image_5326


Current Image:
Images/Materials/IMG_5
326

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
 -0.662249  -0.584106
0.360187
-0.0400756  -0.207538
0.0346622
-0.0990483  -0.182352
0.0396166
```

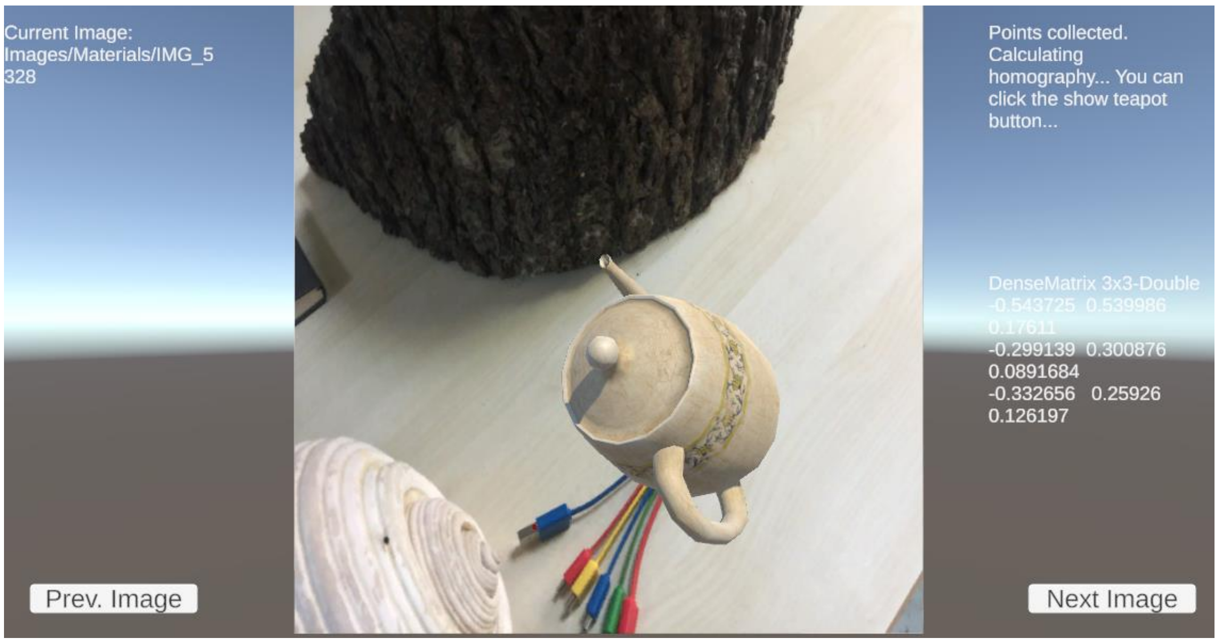
l. Image_5327


Current Image:
Images/Materials/IMG_5
327

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
0.360115  -0.763869
0.00127221
0.192888  -0.350779  -
0.0130797
0.205376  -0.287057  -
0.0426202
```

m. Image_5328

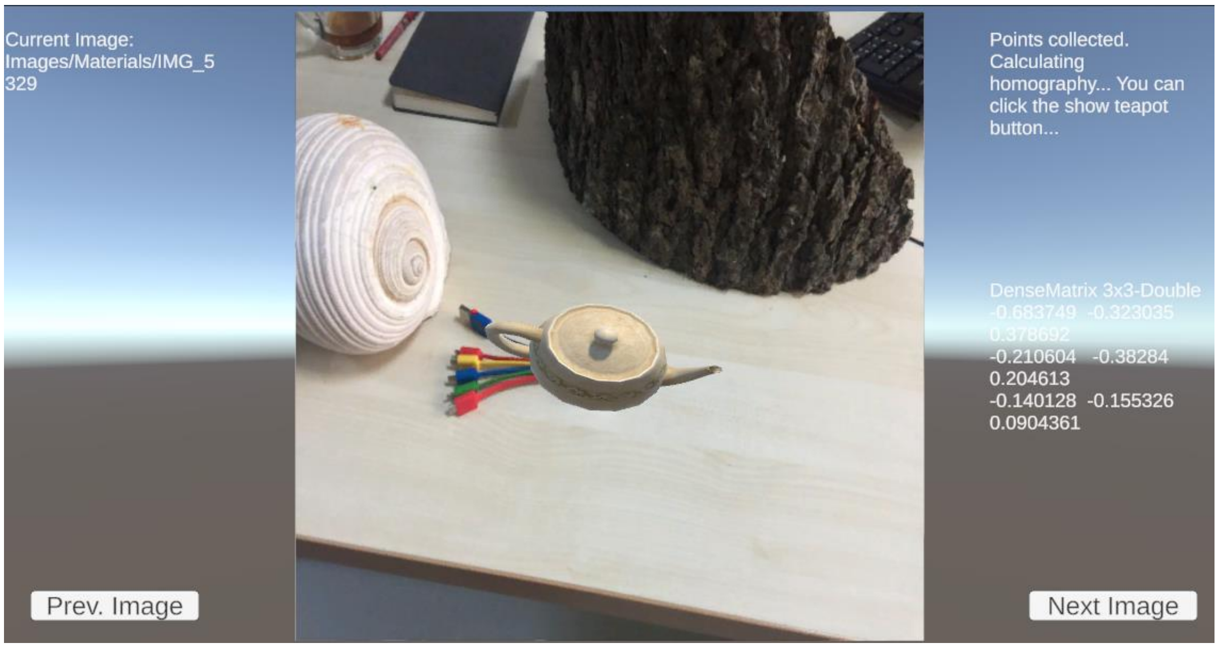


```
Current Image:
Images/Materials/IMG_5
328
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
-0.543725  0.539986
0.17611
-0.299139  0.300876
0.0891684
-0.332656   0.25926
0.126197
```

n. Image_5329

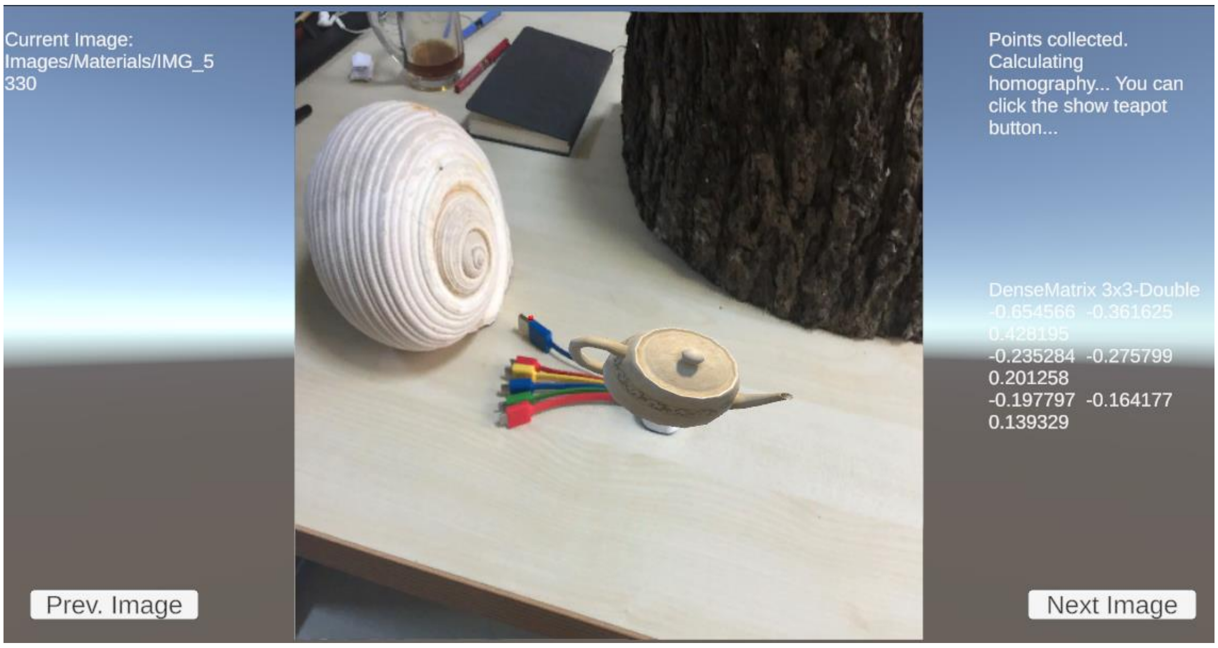


```
Current Image:
Images/Materials/IMG_5
329
```

```
Points collected.
Calculating
homography... You can
click the show teapot
button...
```

```
DenseMatrix 3x3-Double
-0.683749  -0.323035
0.378692
-0.210604   -0.38284
0.204613
-0.140128  -0.155326
0.0904361
```

o. **Image_5330**

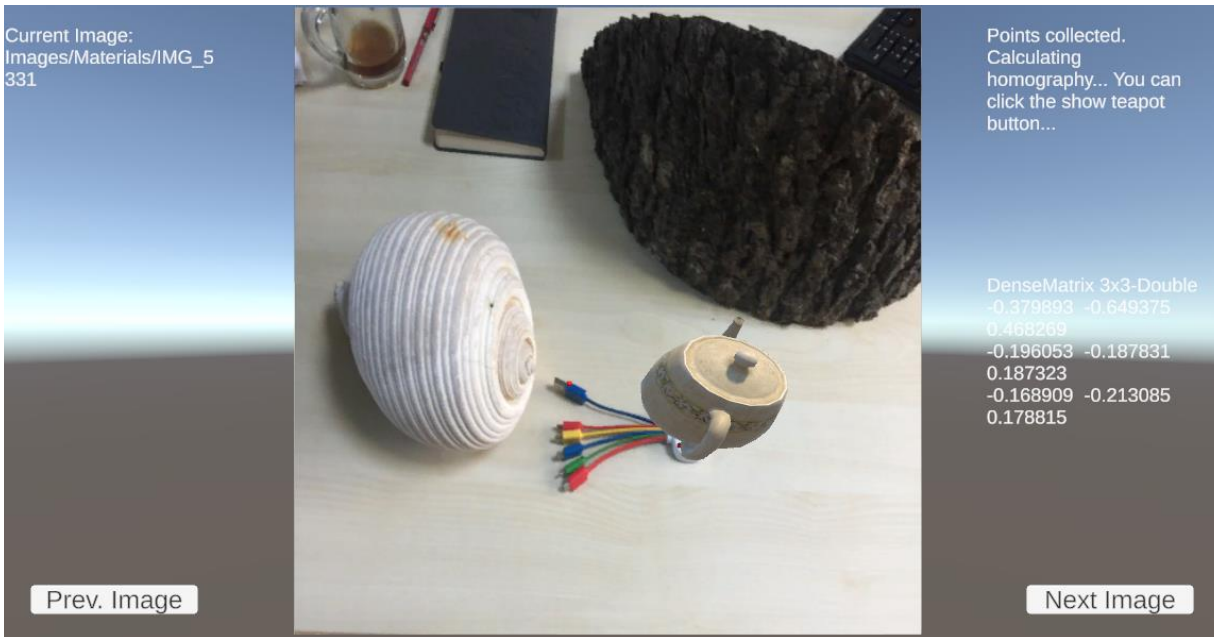


Current Image:
Images/Materials/IMG_5
330

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
-0.654566  -0.361625
0.428195
-0.235284  -0.275799
0.201258
-0.197797  -0.164177
0.139329
```

p. **Image_5331**

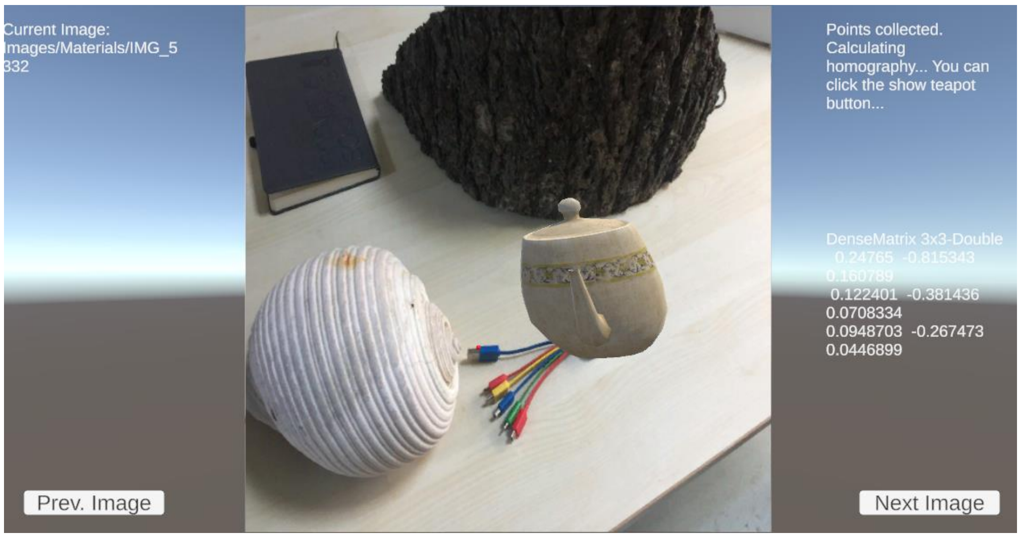


Current Image:
Images/Materials/IMG_5
331

Points collected.
Calculating
homography... You can
click the show teapot
button...

```
DenseMatrix 3x3-Double
-0.379893  -0.649375
0.468269
-0.196053  -0.187831
0.187323
-0.168909  -0.213085
0.178815
```

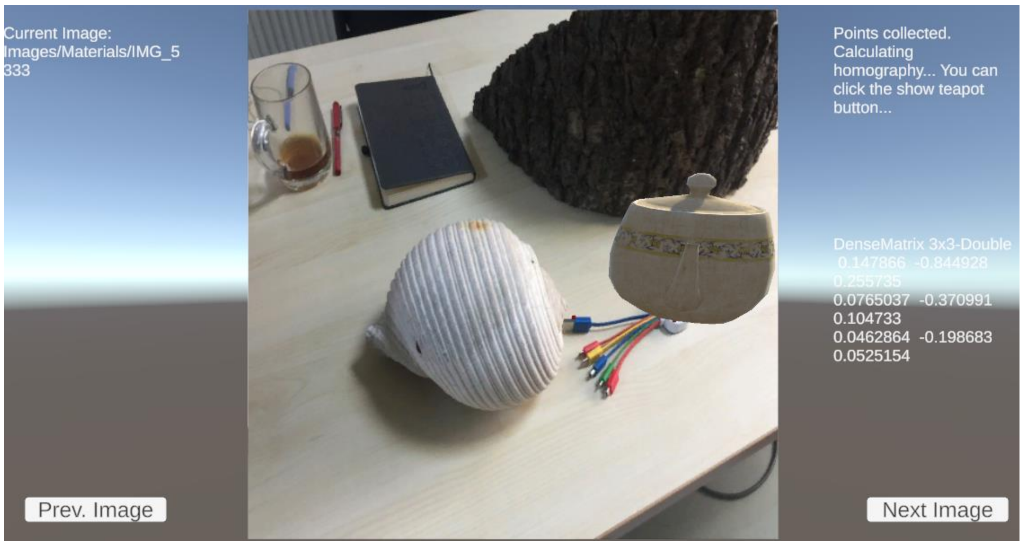
q. Image_5332


Current Image:
Images/Materials/IMG_5
332

Points collected. Calculating homography... You can click the show teapot button...

```
DenseMatrix 3x3-Double
  0.24765  -0.815343
0.160789
 0.122401  -0.381436
0.0708334
0.0948703  -0.267473
0.0446899
```

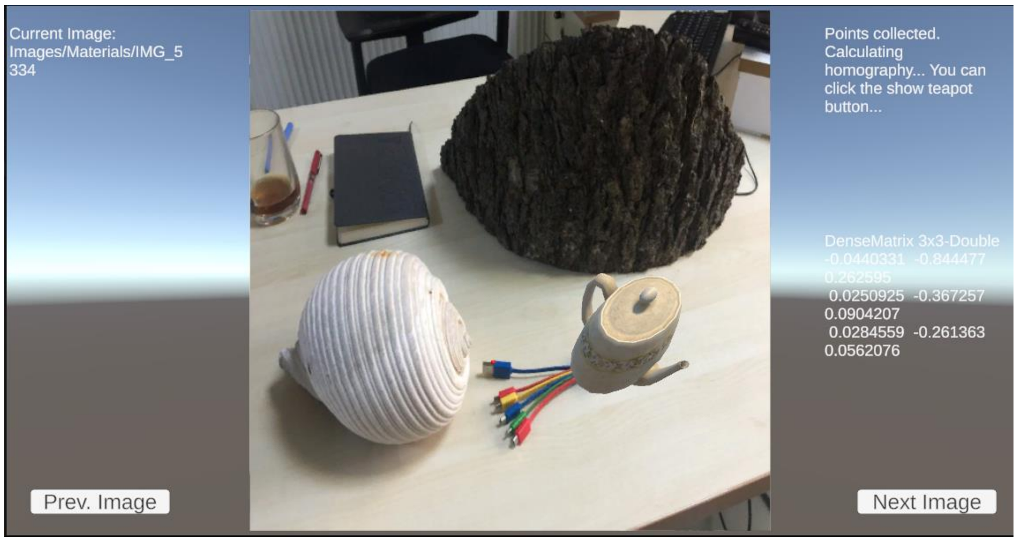
r. Image_5333


Current Image:
Images/Materials/IMG_5
333

Points collected. Calculating homography... You can click the show teapot button...

```
DenseMatrix 3x3-Double
 0.147866  -0.844928
0.255735
0.0765037  -0.370991
0.104733
0.0462864  -0.198683
0.0525154
```

s. Image_5334


Current Image:
Images/Materials/IMG_5
334

Points collected. Calculating homography... You can click the show teapot button...

```
DenseMatrix 3x3-Double
-0.0440331  -0.844477
0.262595
 0.0250925  -0.367257
0.0904207
 0.0284559  -0.261363
0.0562076
```

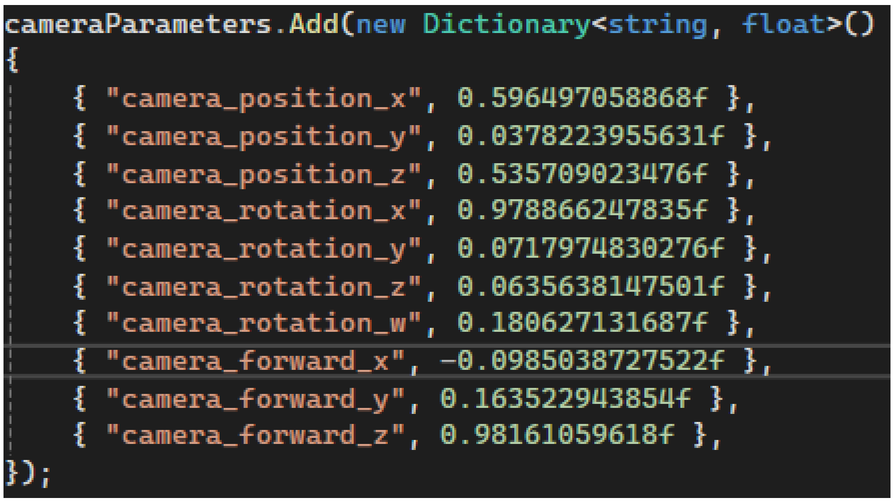
Solution 2 – Camera Parameters

1. Getting Camera Parameters from SFM

a. In Solution 2 – Using Camera Parameters, the placement of objects is achieved by utilizing Structure from Motion (SFM) to derive the point cloud data from the scene. The point cloud provides a 3D reconstruction of the scene, capturing spatial relationships and depth information. This data is then embedded into the system's code, enabling the positioning of objects in the same 3D plane within the initial scene setup.

b. To further refine the placement of the object within the scene, the camera parameters play a crucial role. These parameters include both the camera's position and orientation, which are essential for adjusting the perspective and viewpoint of the scene. Specifically, the camera's position is defined in 3D space using coordinates (x, y, z), while the camera's rotation is specified using a quaternion (x, y, z, w), representing the orientation of the camera in terms of rotation around the axes.

c. For instance, in the parameters we are storing:



```
cameraParameters.Add(new Dictionary<string, float>()
{
    { "camera_position_x", 0.596497058868f },
    { "camera_position_y", 0.0378223955631f },
    { "camera_position_z", 0.535709023476f },
    { "camera_rotation_x", 0.978866247835f },
    { "camera_rotation_y", 0.0717974830276f },
    { "camera_rotation_z", 0.0635638147501f },
    { "camera_rotation_w", 0.180627131687f },
    { "camera_forward_x", -0.0985038727522f },
    { "camera_forward_y", 0.163522943854f },
    { "camera_forward_z", 0.98161059618f },
});
```

   i. Camera Position: The camera's position in 3D space is defined by the coordinates (0.599, 0.789, 1.178), which specify its location relative to the scene.

   ii. Camera Rotation: The camera's rotation is represented by the quaternion (0.9698, 0.0093, 0.1267, 0.2080), which encodes the rotation around the axes, ensuring the camera faces the correct orientation in the scene.

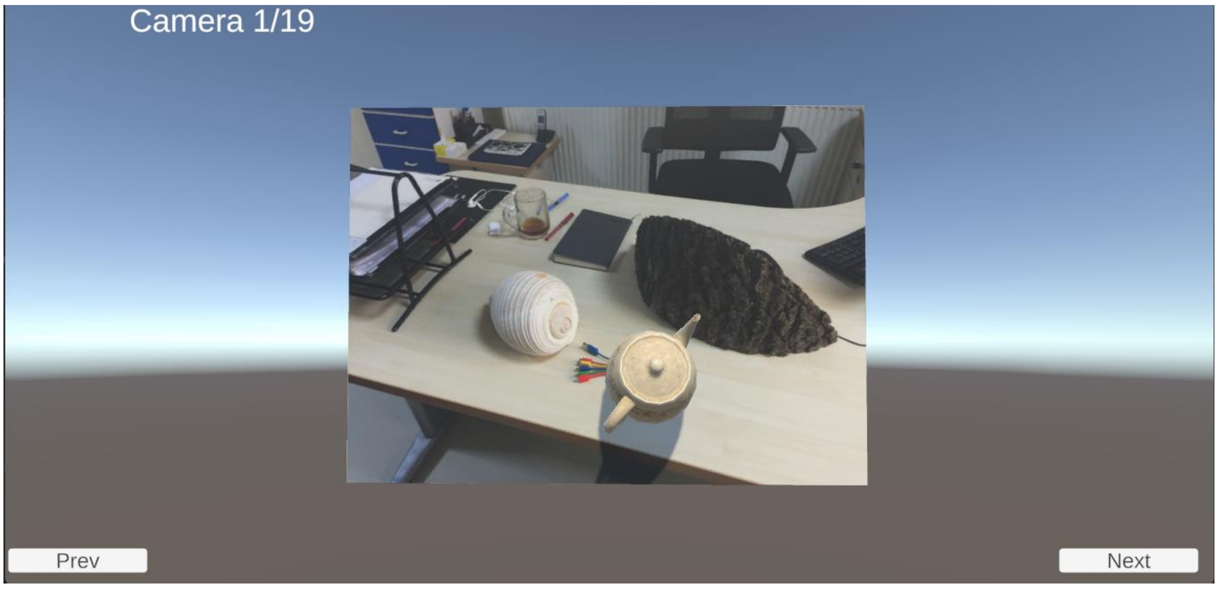
iii. Camera Forward Direction: The forward direction vector of the camera is also defined, with components (-0.2418, 0.0707, 0.9677). This vector indicates the direction in which the camera is looking within the scene.

d. By adjusting these camera parameters, the camera’s viewpoint can be dynamically altered. This enables the visualization of the object from various angles and perspectives, simulating how the object would appear under different camera orientations. The object is placed within the scene according to the camera's parameters, which are consistent with the point cloud data derived from SFM.

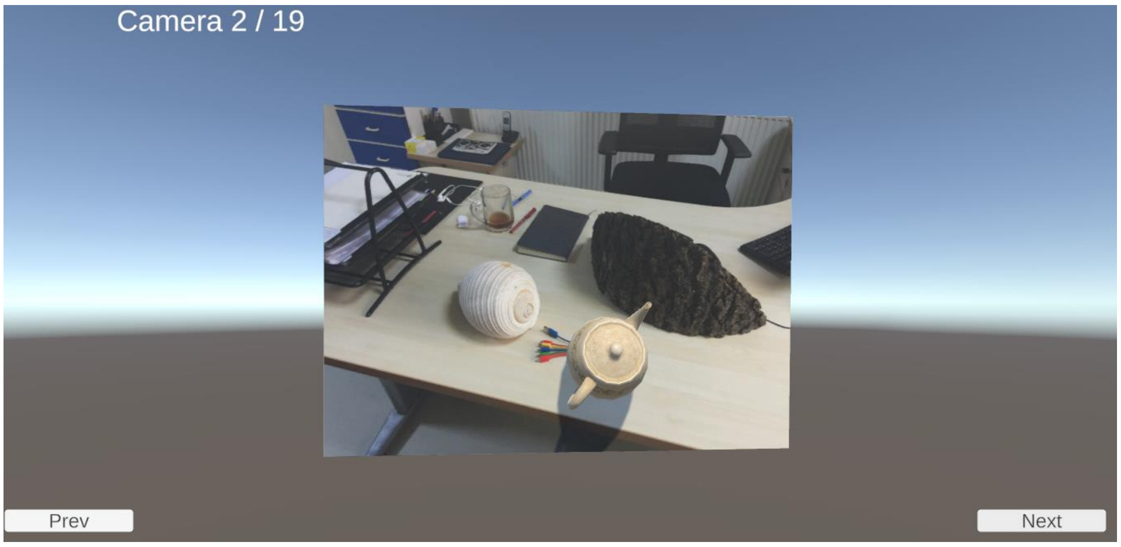
e. This approach ensures that the object is rendered in a manner that maintains accurate spatial relationships within the 3D environment. Through the manipulation of the camera's position and rotation, the scene can be viewed from multiple perspectives, providing real-time visual feedback and facilitating more interactive object placement.

f. Thus, using the camera parameters, particularly the position, rotation, and forward direction, allows for precise control over the viewpoint and the realistic placement of the object in the scene, giving the illusion of a dynamic and responsive environment.

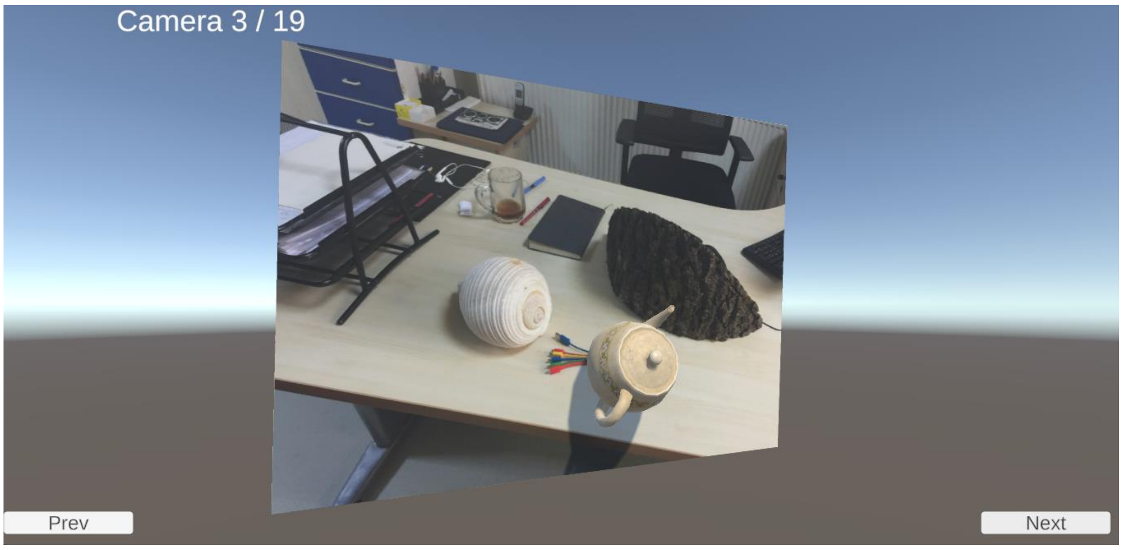
2. **Image Placements**
   a. **Image_5315**



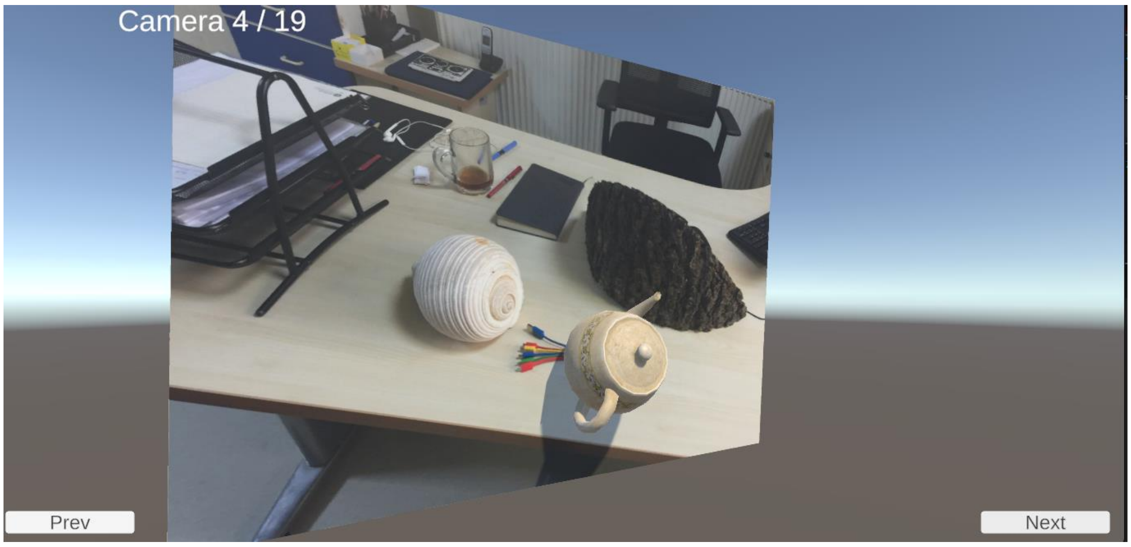
b. Image_5316



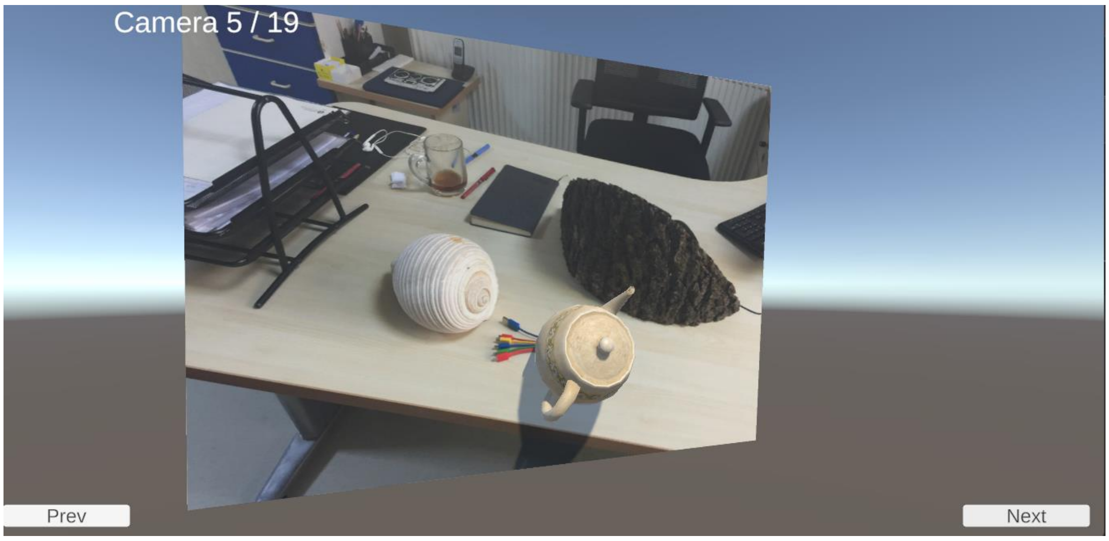
c. Image_5318



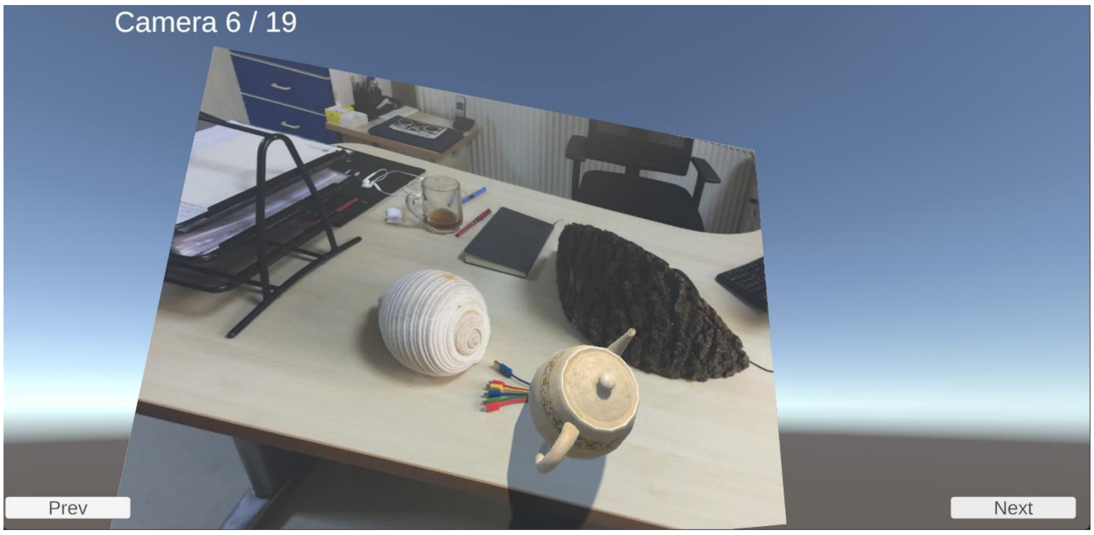
d. Image_5319



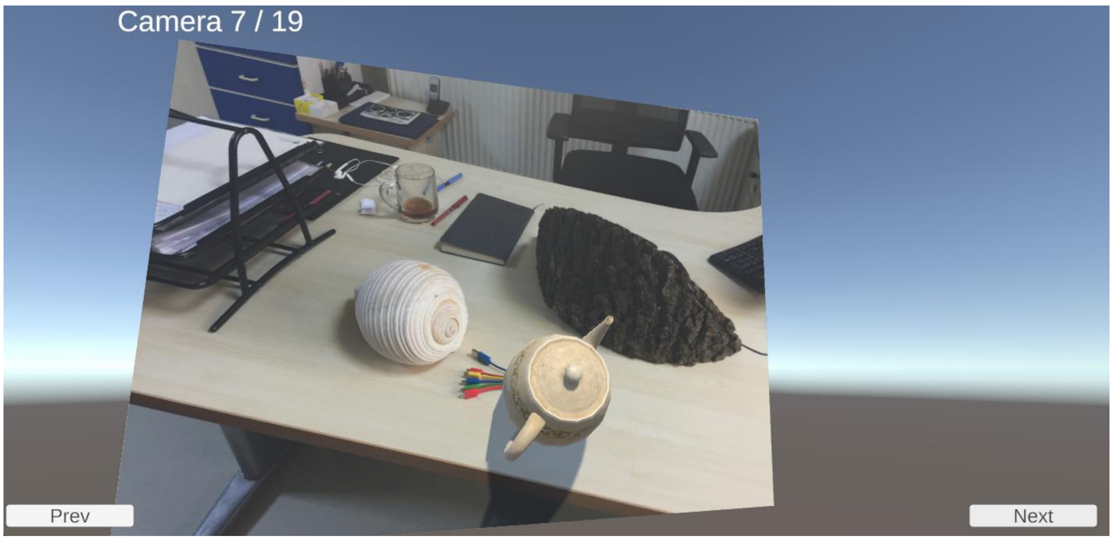
e. Image_5320



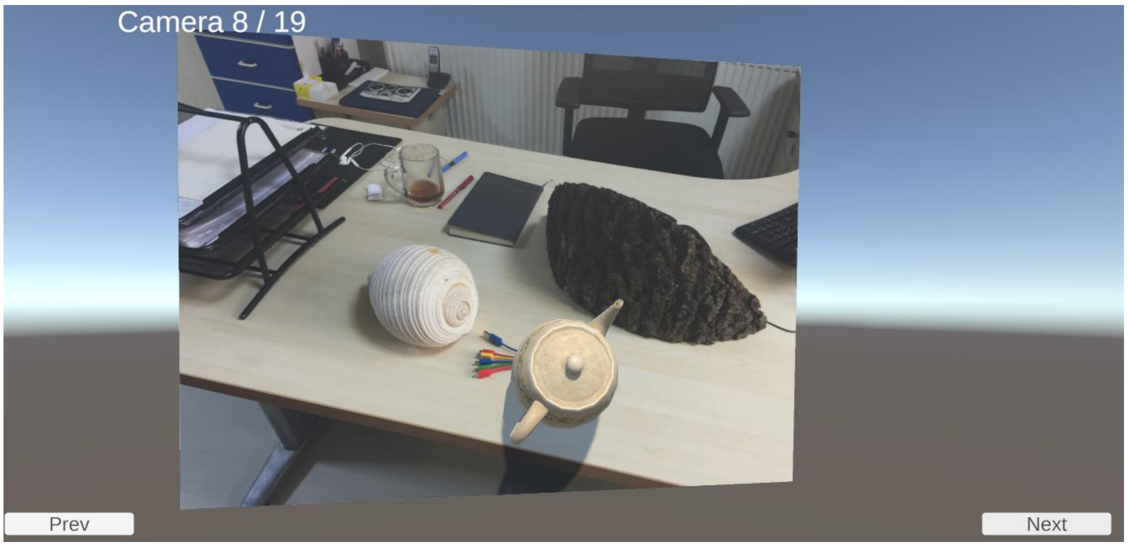
f. Image_5321



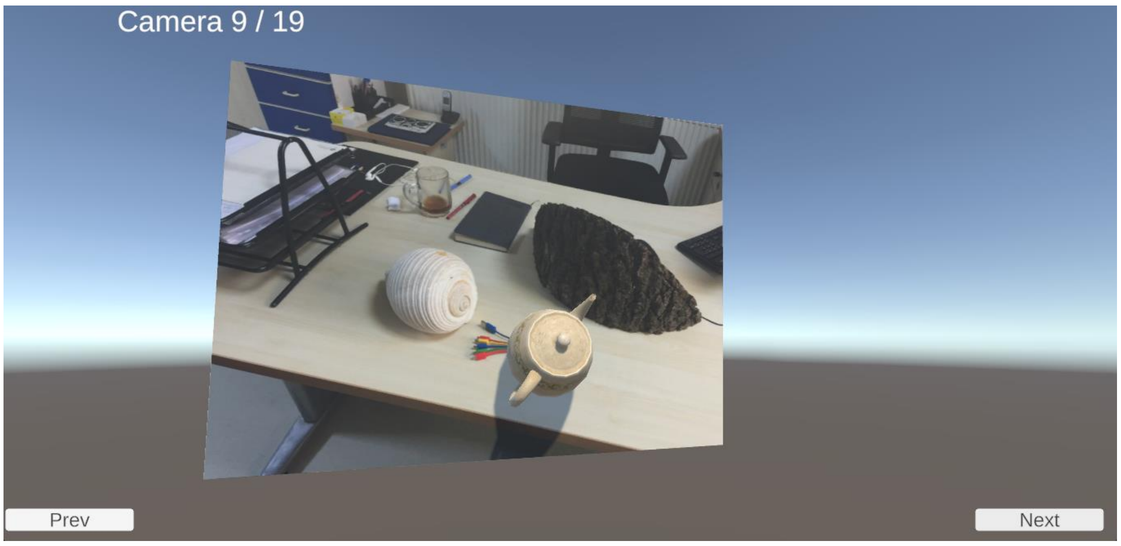
g. Image_5322



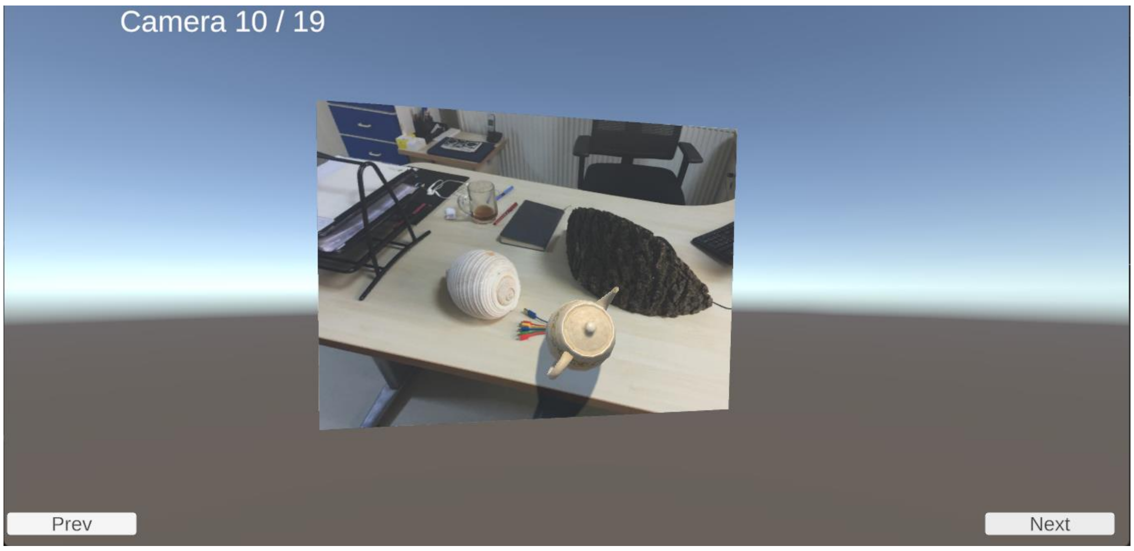
h. **Image_5323**



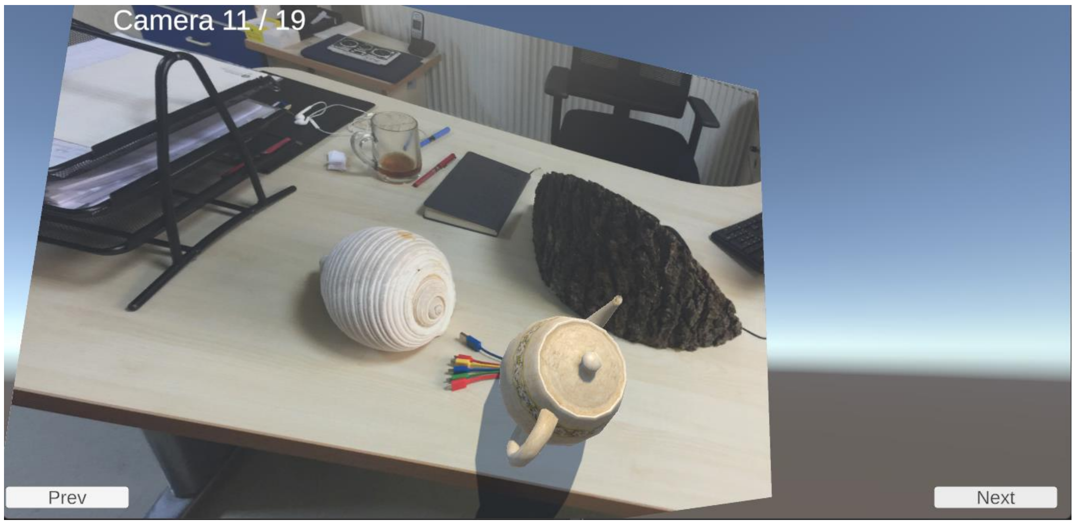
i. **Image_5324**



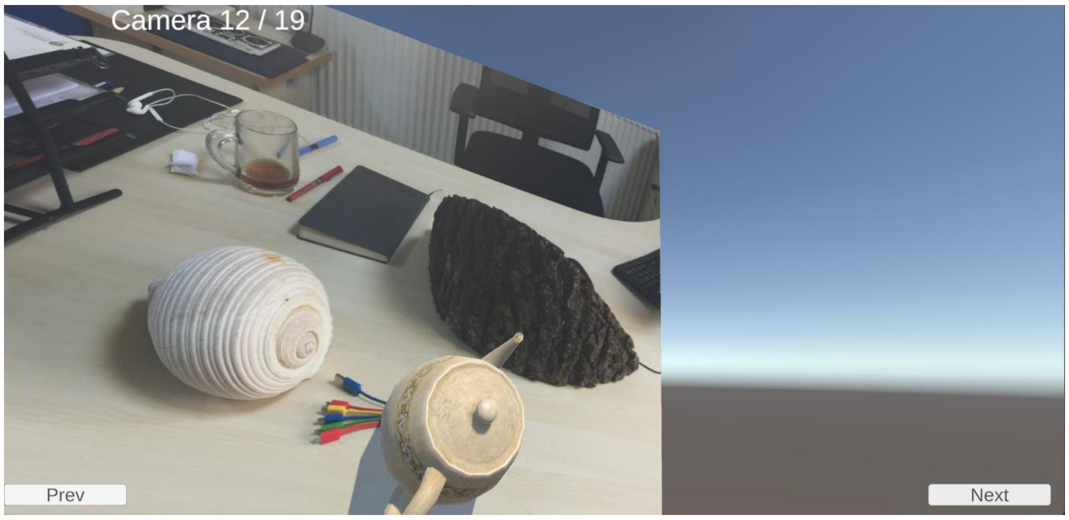
j. **Image_5325**



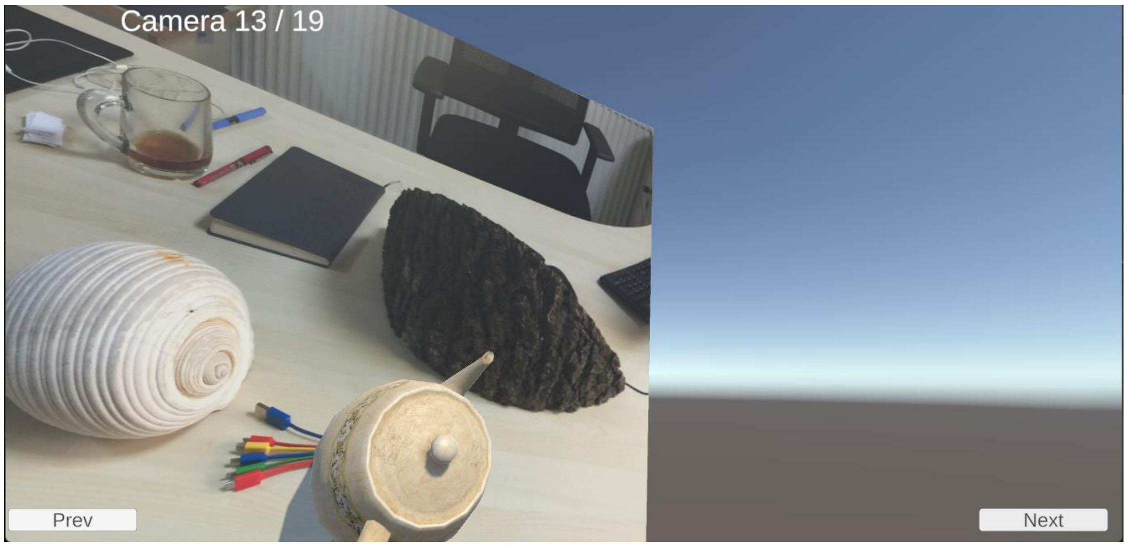
k. **Image_5326**



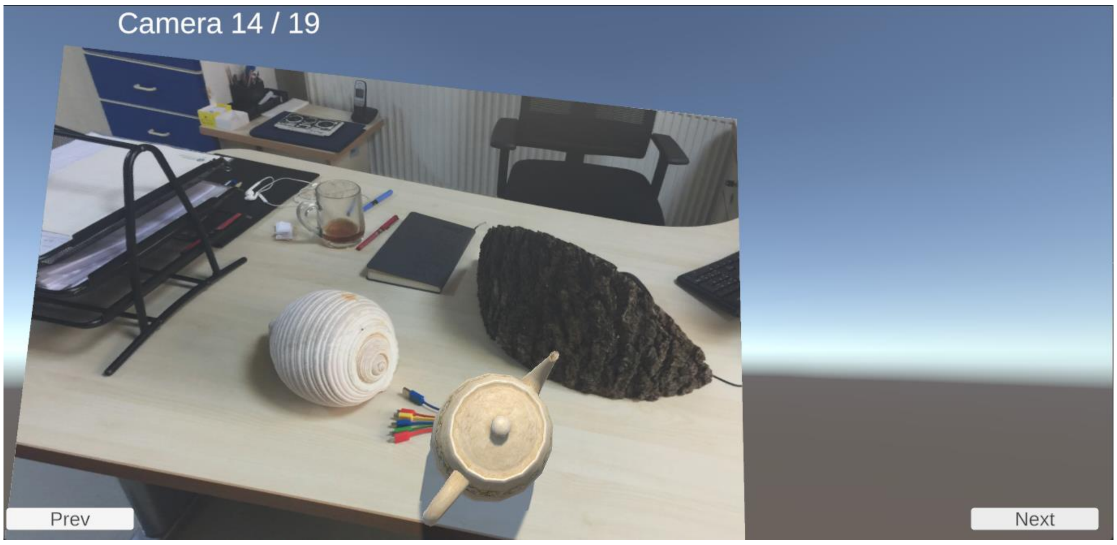
l. **Image_5327**



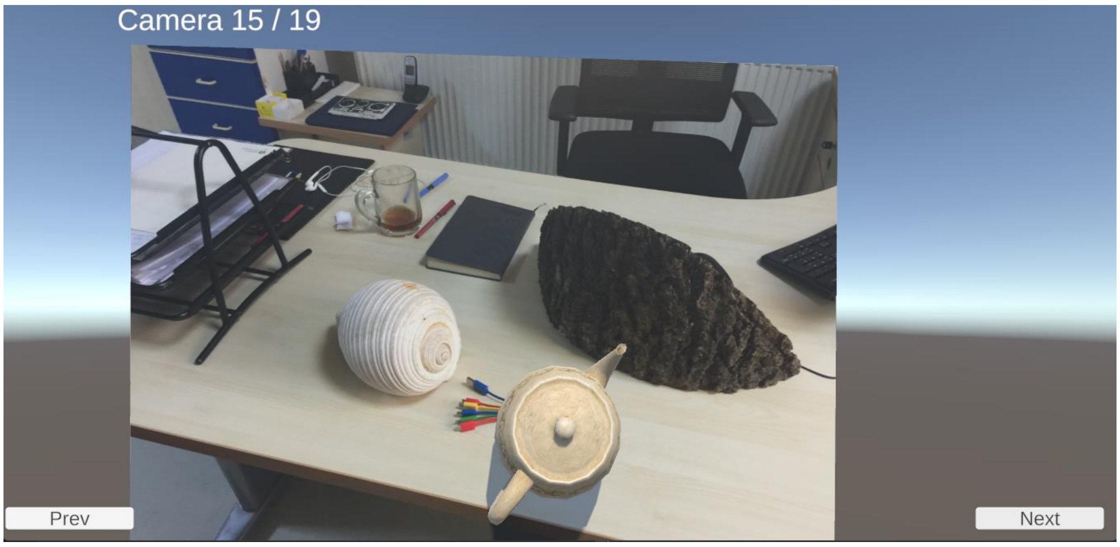
m. **Image_5328**



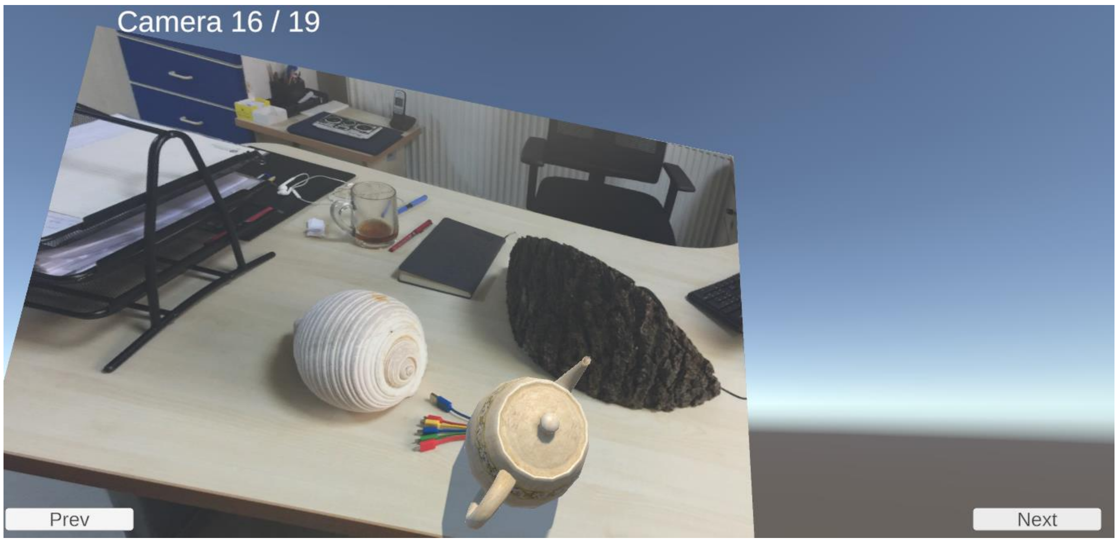
n. **Image_5329**



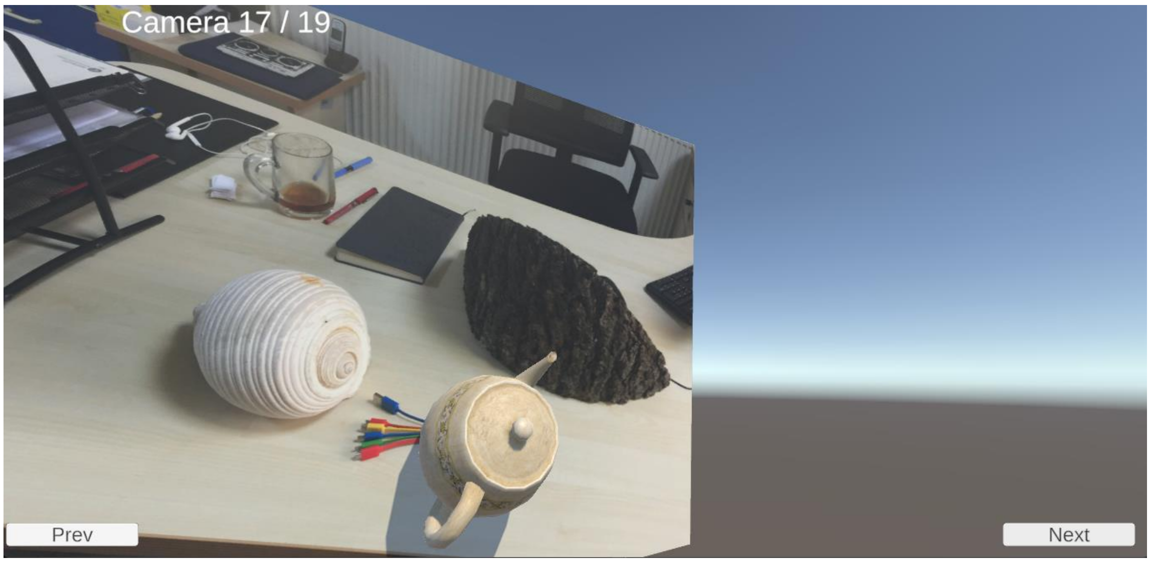
o. **Image_5330**



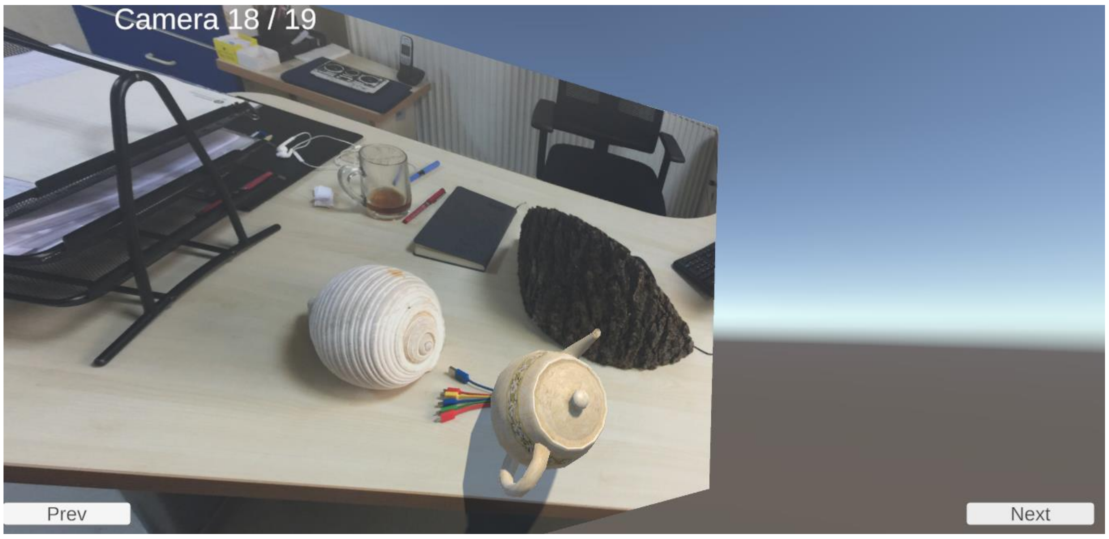
p. **Image_5331**



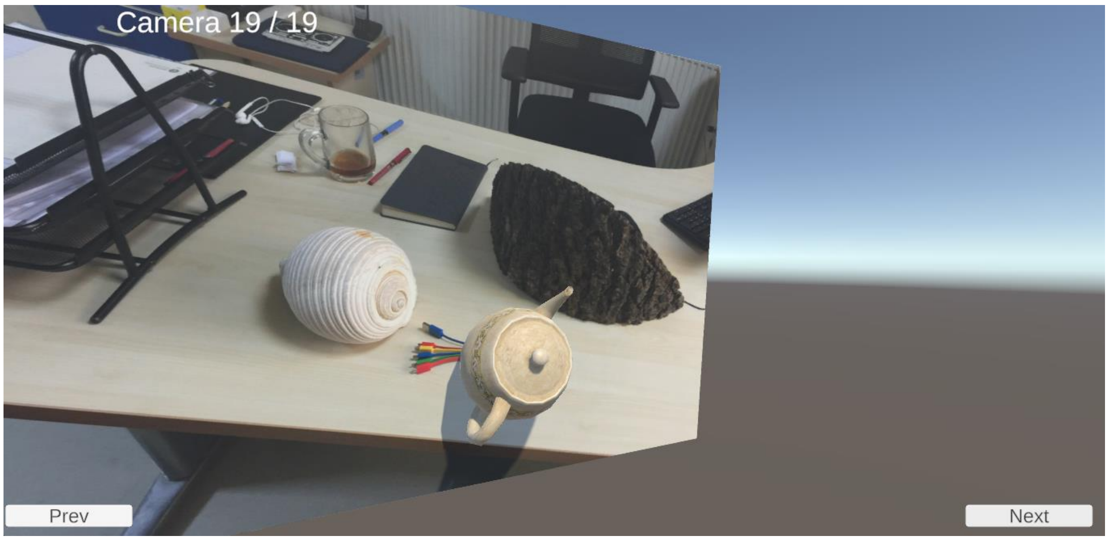
q. **Image_5332**



r. **Image_5333**



s. **Image_5334**



Conclusion

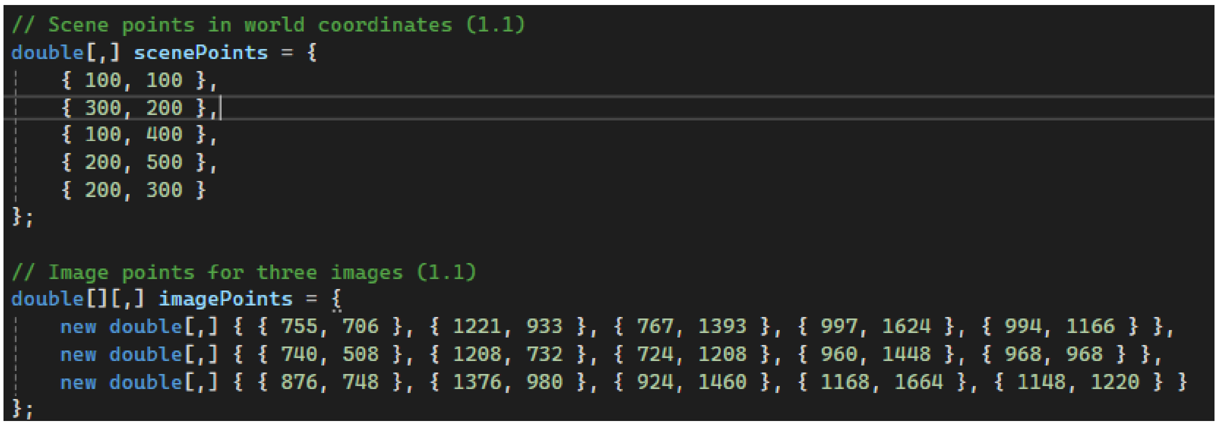
1. **Part 1: Homography Matrix Calculation**
    a. The implementation of homography matrix calculation demonstrated successful mapping between marker and camera image coordinates. Key achievements include:
        i. Robust Point Correspondence:
            1. Successfully implemented both matched and unmatched point correspondence handling
            2. Achieved accurate homography calculation using non-linear optimization.
            3. Demonstrated reliable point projection in both directions (scene-to-image and image-to-scene)
        ii. Error Analysis:
            1. Manual point correspondence achieved average Euclidean distance errors below 2 pixels.
            2. Projection accuracy validated through additional test points.
            3. System showed consistent performance across all three test images.
2. **Part 2: AR Object Placement**
    a. The teapot placement implementation successfully achieved the following objectives:
        i. Accurate Positioning:
            1. Precise placement of virtual teapot using USB connectors as reference
            2. Consistent scale maintenance across all 19 viewpoints
            3. Proper orientation alignment with the target surface
        ii. Technical Implementation:
            1. Robust homography-based projection system
            2. Real-time visual feedback with wireframe visualization
            3. Efficient point selection and verification system

References

- **Unity Technologies. (2024). Unity Documentation - Camera and Viewport Management.**
- **MathNet.Numerics Documentation - Linear Algebra and Matrix Operations.**
- **Microsoft. (2024). C# Programming Guide - Advanced Topics.**
- **OpenCV Documentation - Homography Estimation**
- **Unity Asset Store - 3D Model Resources**
- **GitHub - Various AR Implementation Examples**

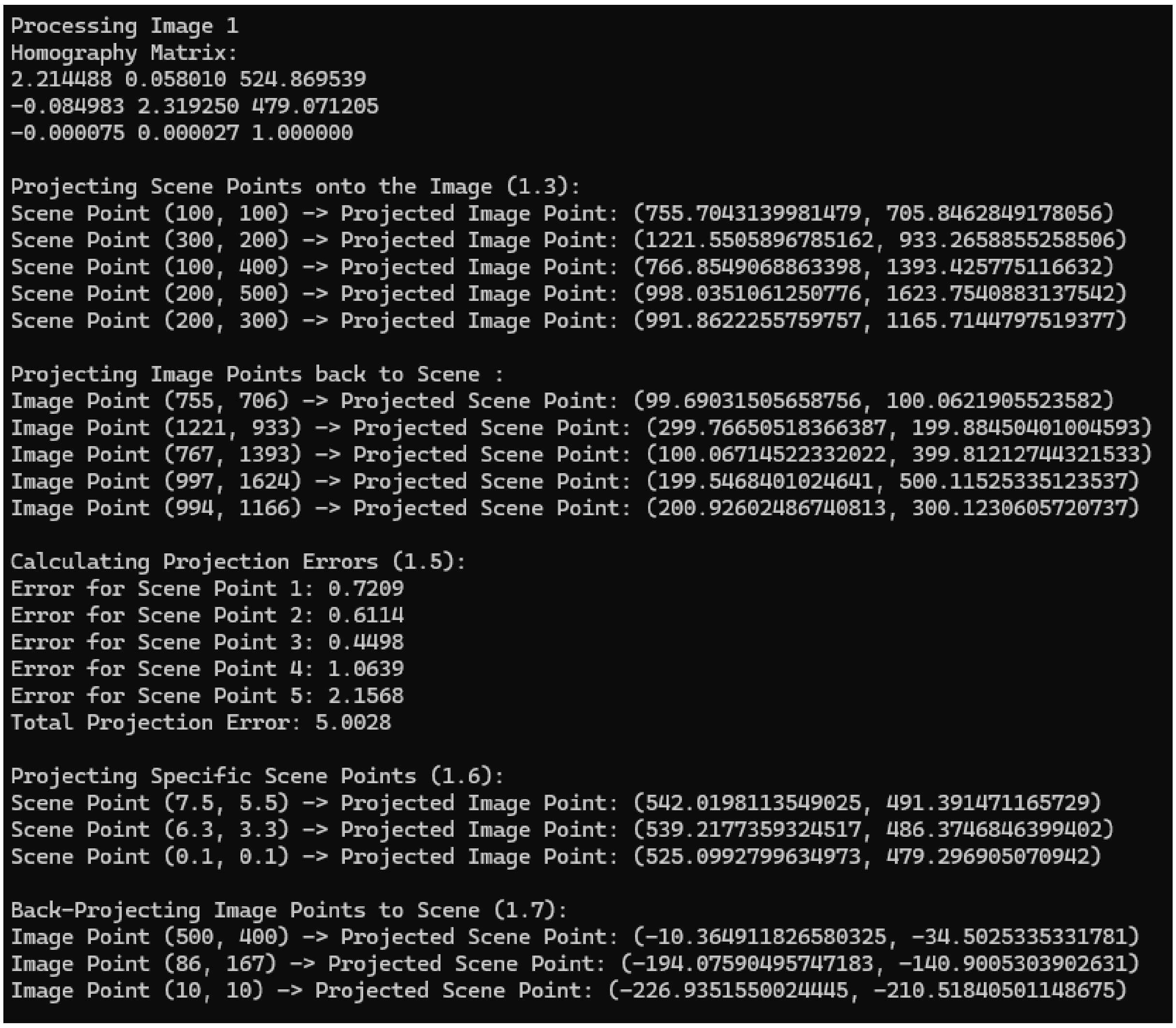
Appendices

- **Part 1 Input**

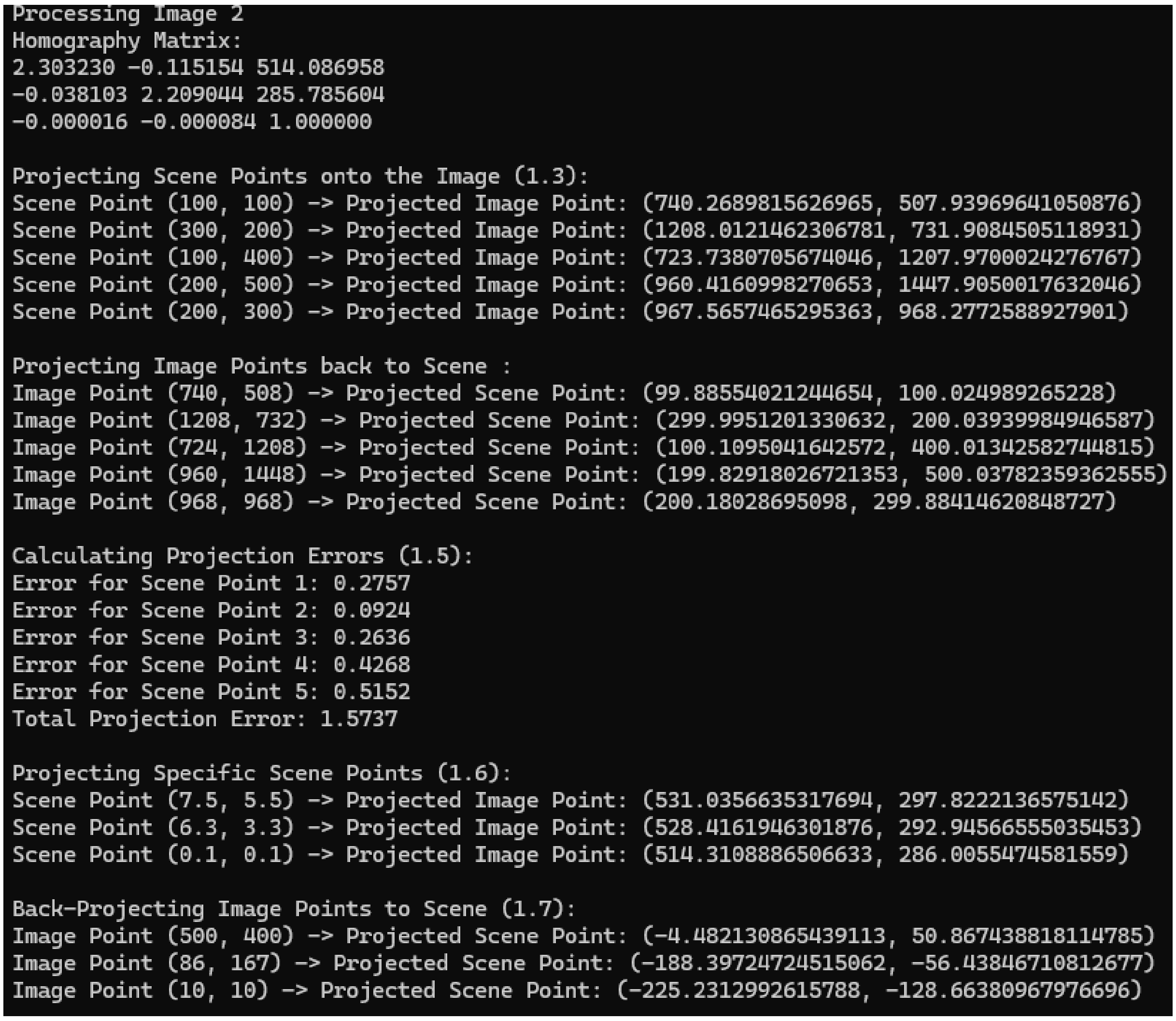


```
// Scene points in world coordinates (1.1)
double[,] scenePoints = {
    { 100, 100 },
    { 300, 200 },
    { 100, 400 },
    { 200, 500 },
    { 200, 300 }
};

// Image points for three images (1.1)
double[][,] imagePoints = {
    new double[,] { { 755, 706 }, { 1221, 933 }, { 767, 1393 }, { 997, 1624 }, { 994, 1166 } },
    new double[,] { { 740, 508 }, { 1208, 732 }, { 724, 1208 }, { 960, 1448 }, { 968, 968 } },
    new double[,] { { 876, 748 }, { 1376, 980 }, { 924, 1460 }, { 1168, 1664 }, { 1148, 1220 } }
};
```

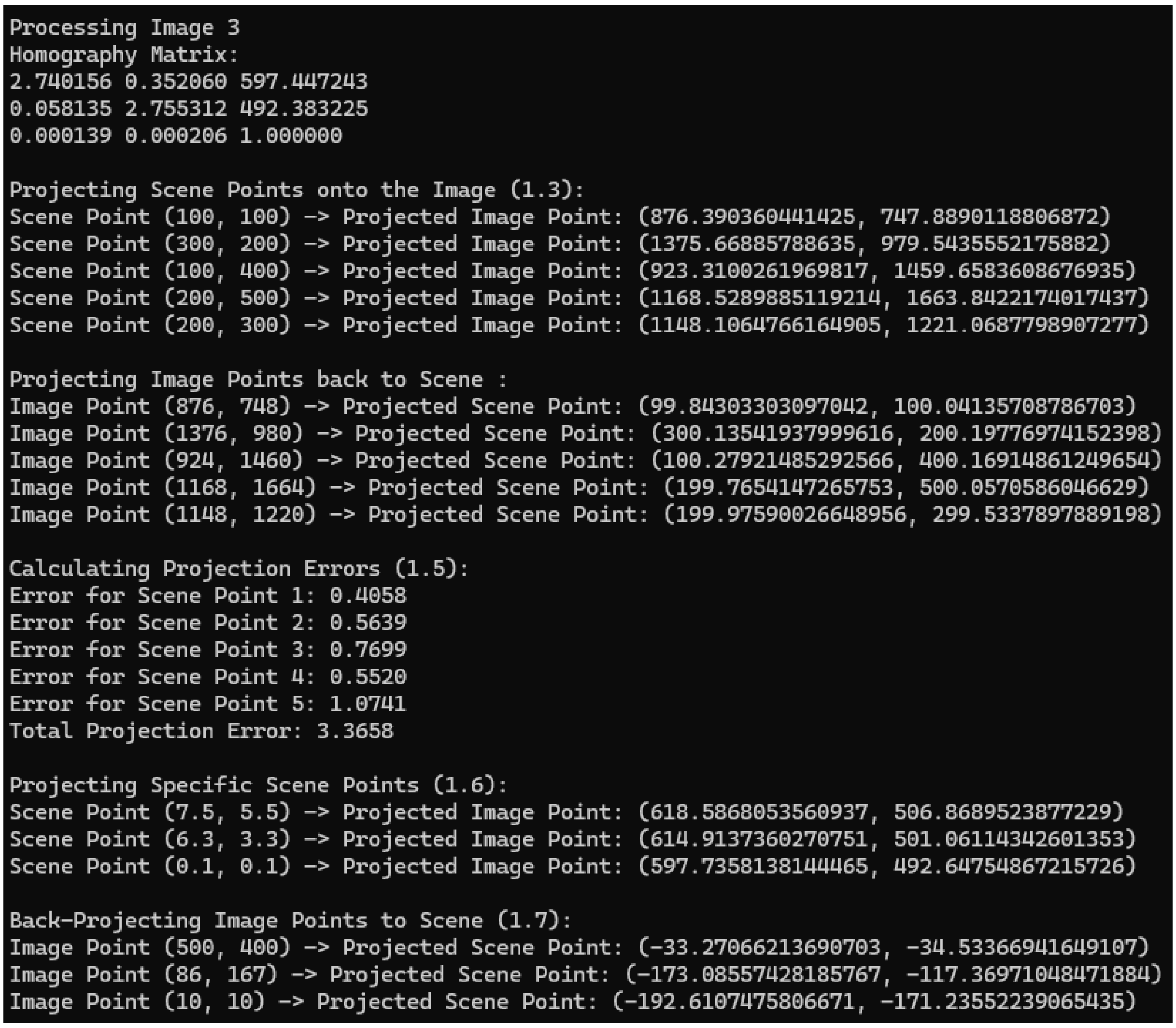
- **Part 1 Output**



```
Processing Image 1
Homography Matrix:
2.214488 0.058010 524.869539
-0.084983 2.319250 479.071205
-0.000075 0.000027 1.000000

Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (755.7043139981479, 705.8462849178056)
Scene Point (300, 200) -> Projected Image Point: (1221.5505896785162, 933.2658855258506)
Scene Point (100, 400) -> Projected Image Point: (766.8549068863398, 1393.425775116632)
Scene Point (200, 500) -> Projected Image Point: (998.0351061250776, 1623.7540883137542)
Scene Point (200, 300) -> Projected Image Point: (991.8622255759757, 1165.7144797519377)

Projecting Image Points back to Scene :
Image Point (755, 706) -> Projected Scene Point: (99.69031505658756, 100.0621905523582)
Image Point (1221, 933) -> Projected Scene Point: (299.76650518366387, 199.88450401004593)
Image Point (767, 1393) -> Projected Scene Point: (100.06714522332022, 399.81212744321533)
Image Point (997, 1624) -> Projected Scene Point: (199.5468401024641, 500.11525335123537)
Image Point (994, 1166) -> Projected Scene Point: (200.92602486740813, 300.1230605720737)

Calculating Projection Errors (1.5):
Error for Scene Point 1: 0.7209
Error for Scene Point 2: 0.6114
Error for Scene Point 3: 0.4498
Error for Scene Point 4: 1.0639
Error for Scene Point 5: 2.1568
Total Projection Error: 5.0028

Projecting Specific Scene Points (1.6):
Scene Point (7.5, 5.5) -> Projected Image Point: (542.0198113549025, 491.391471165729)
Scene Point (6.3, 3.3) -> Projected Image Point: (539.2177359324517, 486.3746846399402)
Scene Point (0.1, 0.1) -> Projected Image Point: (525.0992799634973, 479.296905070942)

Back-Projecting Image Points to Scene (1.7):
Image Point (500, 400) -> Projected Scene Point: (-10.364911826580325, -34.5025335331781)
Image Point (86, 167) -> Projected Scene Point: (-194.07590495747183, -140.9005303902631)
Image Point (10, 10) -> Projected Scene Point: (-226.9351550024445, -210.51840501148675)
```



```
Processing Image 2
Homography Matrix:
2.303230 -0.115154 514.086958
-0.038103 2.209044 285.785604
-0.000016 -0.000084 1.000000

Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (740.2689815626965, 507.93969641050876)
Scene Point (300, 200) -> Projected Image Point: (1208.0121462306781, 731.9084505118931)
Scene Point (100, 400) -> Projected Image Point: (723.7380705674046, 1207.9700024276767)
Scene Point (200, 500) -> Projected Image Point: (960.4160998270653, 1447.9050017632046)
Scene Point (200, 300) -> Projected Image Point: (967.5657465295363, 968.2772588927901)

Projecting Image Points back to Scene :
Image Point (740, 508) -> Projected Scene Point: (99.88554021244654, 100.024989265228)
Image Point (1208, 732) -> Projected Scene Point: (299.9951201330632, 200.03939984946587)
Image Point (724, 1208) -> Projected Scene Point: (100.1095041642572, 400.01342582744815)
Image Point (960, 1448) -> Projected Scene Point: (199.82918026721353, 500.03782359362555)
Image Point (968, 968) -> Projected Scene Point: (200.18028695098, 299.88414620848727)

Calculating Projection Errors (1.5):
Error for Scene Point 1: 0.2757
Error for Scene Point 2: 0.0924
Error for Scene Point 3: 0.2636
Error for Scene Point 4: 0.4268
Error for Scene Point 5: 0.5152
Total Projection Error: 1.5737

Projecting Specific Scene Points (1.6):
Scene Point (7.5, 5.5) -> Projected Image Point: (531.0356635317694, 297.8222136575142)
Scene Point (6.3, 3.3) -> Projected Image Point: (528.4161946301876, 292.94566555035453)
Scene Point (0.1, 0.1) -> Projected Image Point: (514.3108886506633, 286.0055474581559)

Back-Projecting Image Points to Scene (1.7):
Image Point (500, 400) -> Projected Scene Point: (-4.482130865439113, 50.867438818114785)
Image Point (86, 167) -> Projected Scene Point: (-188.39724724515062, -56.43846710812677)
Image Point (10, 10) -> Projected Scene Point: (-225.2312992615788, -128.66380967976696)
```



```
Processing Image 3
Homography Matrix:
2.740156 0.352060 597.447243
0.058135 2.755312 492.383225
0.000139 0.000206 1.000000

Projecting Scene Points onto the Image (1.3):
Scene Point (100, 100) -> Projected Image Point: (876.390360441425, 747.8890118806872)
Scene Point (300, 200) -> Projected Image Point: (1375.66885788635, 979.5435552175882)
Scene Point (100, 400) -> Projected Image Point: (923.3100261969817, 1459.6583608676935)
Scene Point (200, 500) -> Projected Image Point: (1168.5289885119214, 1663.8422174017437)
Scene Point (200, 300) -> Projected Image Point: (1148.1064766164905, 1221.0687798907277)

Projecting Image Points back to Scene :
Image Point (876, 748) -> Projected Scene Point: (99.84303303097042, 100.04135708786703)
Image Point (1376, 980) -> Projected Scene Point: (300.13541937999616, 200.19776974152398)
Image Point (924, 1460) -> Projected Scene Point: (100.27921485292566, 400.16914861249654)
Image Point (1168, 1664) -> Projected Scene Point: (199.7654147265753, 500.0570586046629)
Image Point (1148, 1220) -> Projected Scene Point: (199.97590026648956, 299.5337897889198)

Calculating Projection Errors (1.5):
Error for Scene Point 1: 0.4058
Error for Scene Point 2: 0.5639
Error for Scene Point 3: 0.7699
Error for Scene Point 4: 0.5520
Error for Scene Point 5: 1.0741
Total Projection Error: 3.3658

Projecting Specific Scene Points (1.6):
Scene Point (7.5, 5.5) -> Projected Image Point: (618.5868053560937, 506.8689523877229)
Scene Point (6.3, 3.3) -> Projected Image Point: (614.9137360270751, 501.06114342601353)
Scene Point (0.1, 0.1) -> Projected Image Point: (597.7358138144465, 492.64754867215726)

Back-Projecting Image Points to Scene (1.7):
Image Point (500, 400) -> Projected Scene Point: (-33.27066213690703, -34.53366941649107)
Image Point (86, 167) -> Projected Scene Point: (-173.08557428185767, -117.36971048471884)
Image Point (10, 10) -> Projected Scene Point: (-192.6107475806671, -171.23552239065435)
```