Gebze Technical University

CSE462-CSE562 Augmented Reality in Fall 2022

Basic Camera and Scene Calibration

Homework 3 Report

Muhammed Bedir ULUCAY

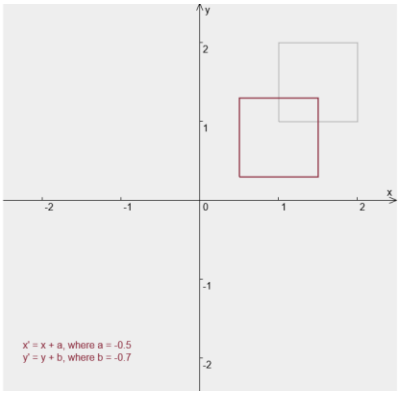
1901042697

Part 1:

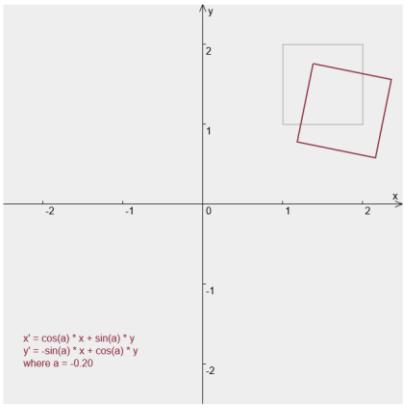
In part 1 implementatin about the projection about scene and the camera calibration. In the 3D world when we capture a photo we are loosing a dimension about depth a point with no different change its x, y coordinate according to the z (depth coordinate) in case of that when we capture a photo. The wanted thing is seeing it as Ortographic Projection. So that for a point when we want to convert homogenious coordinate system we are doing some calculation to set the camera and scene calibration.

We are converting point Cartesian coordinate system to homogenious coordinate system.

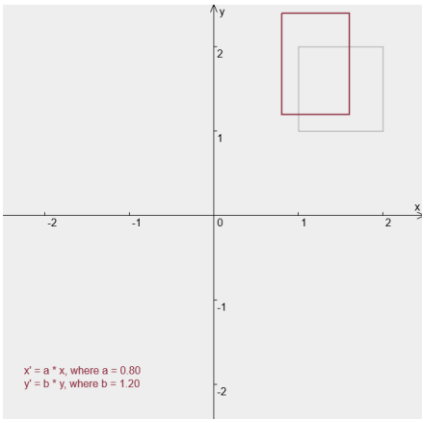
Translate a point is easy one just some the tranlate value then see the new point.



Rotation a point need some calculation using trigonometry and multiplication;



Scale is just multiplication;



Basic formula for converting cartesian coordinate system to homogenious coordinate system formula;

$$x' = Ax + By + C$$
$$y' = Dx + Ey + F$$

translation is:

$x' = x + C$ *(A = 1, B = 0)*
$y' = y + F$ *(D = 0, E = 1)*

$$\widehat{P} = HP$$

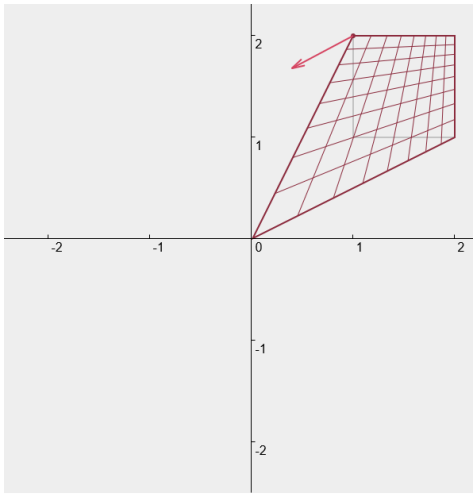
A rotation is:

$x' = sin(r)x + cos(r)y$ *(A = sin(r), B = cos(r), C = 0)*
$y' = cos(r)x - sin(r)y$ *(D = cos(r), E = -sin(r), F = 0)*

$$\widehat{P} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} P$$

And a scale is:

$x' = Ax$ *(B = 0, C = 0)*
$y' = Ey$ *(D = 0, F = 0)*



$$\begin{bmatrix} -4.049 & -2.013 & 6.05 \\ -2.024 & -4.043 & 6.061 \\ -1.012 & -1.006 & 1 \end{bmatrix}$$

For our program the calculation function like below;

We are calculating A matrix first then B matrix and we are making point to point pairing on matrix.

```
static double[,] CalculateHomograpy(double[,] src, double[,] dst){
    double[,] A = new double[8,8];
    double[,] B = new double[8,1];
    double[,] H = new double[8,1];

    for(int i = 0; i < 4; i++){
        A[2*i,0] = src[i,0];
        A[2*i,1] = src[i,1];
        A[2*i,2] = 1;
        A[2*i,3] = 0;
        A[2*i,4] = 0;
        A[2*i,5] = 0;
        A[2*i,6] = -src[i,0]*dst[i,0];
        A[2*i,7] = -src[i,1]*dst[i,0];
        A[2*i+1,0] = 0;
        A[2*i+1,1] = 0;
        A[2*i+1,2] = 0;
        A[2*i+1,3] = src[i,0];
        A[2*i+1,4] = src[i,1];
        A[2*i+1,5] = 1;
        A[2*i+1,6] = -src[i,0]*dst[i,1];
        A[2*i+1,7] = -src[i,1]*dst[i,1];
    }

    for(int i = 0; i < 4; i++){
        B[2*i,0] = dst[i,0];
        B[2*i+1,0] = dst[i,1];
    }

    Matrix<double> A_matrix = Matrix<double>.Build.DenseOfArray(A);
    Matrix<double> B_matrix = Matrix<double>.Build.DenseOfArray(B);

    Matrix<double> H_matrix = A_matrix.Solve(B_matrix);

    for(int i = 0; i < 8; i++){
        H[i,0] = H_matrix[i,0];
    }

    return H;
}
```

Single point calculation projection founding;

```
static double[,] CalculateProjectionOfPoint(double[] point, double[,] H){

    double[,] dst_point = new double[2,1];
    double srcX = point[0];
    double srcY = point[1];

    double dst1 = H[0,0] * srcX + H[1,0] * srcY + H[2,0];
    double dst2 = H[3,0] * srcX + H[4,0] * srcY + H[5,0];
    double dst3 = H[6,0] * srcX + H[7,0] * srcY + 1;

    dst_point[0,0] = dst1 / dst3;
    dst_point[1,0] = dst2 / dst3;

    return dst_point;
}
```

Inverse of a homography;

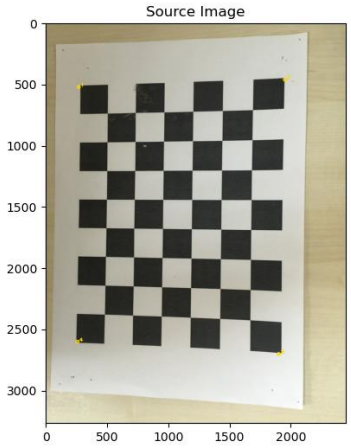
We are doing reverse of the matrices and then doing same homograpy process

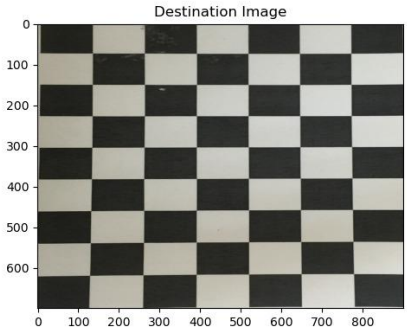
```
static double[,] ApplyInverseHomography(double[,] source_matrices, double[,] destination_matrices)
{
    MathNet.Numerics.LinearAlgebra.Matrix<double> inverse_matrix = DenseMatrix.OfArray(source_matrices).Inverse();
    double[,] inverse_matrix_array = inverse_matrix.ToArray();
    double[,] homographyMatrix = CalculateHomograpy(inverse_matrix_array, destination_matrices);

   return homographyMatrix;
}

static double[,] CalculateInverseProjection(double[,] hm, double[,] uv)
{
    double[,] match = ApplyInverseHomography(hm, uv);

    Console.WriteLine("(u,v) : " + uv[0, 0] + " , " + uv[1, 0] + " , " + uv[2, 0]);
    Console.WriteLine("(x,y) : " + match[0, 0] + " , " + match[1, 0] + " , " + match[2, 0]);

    return match;
}
```

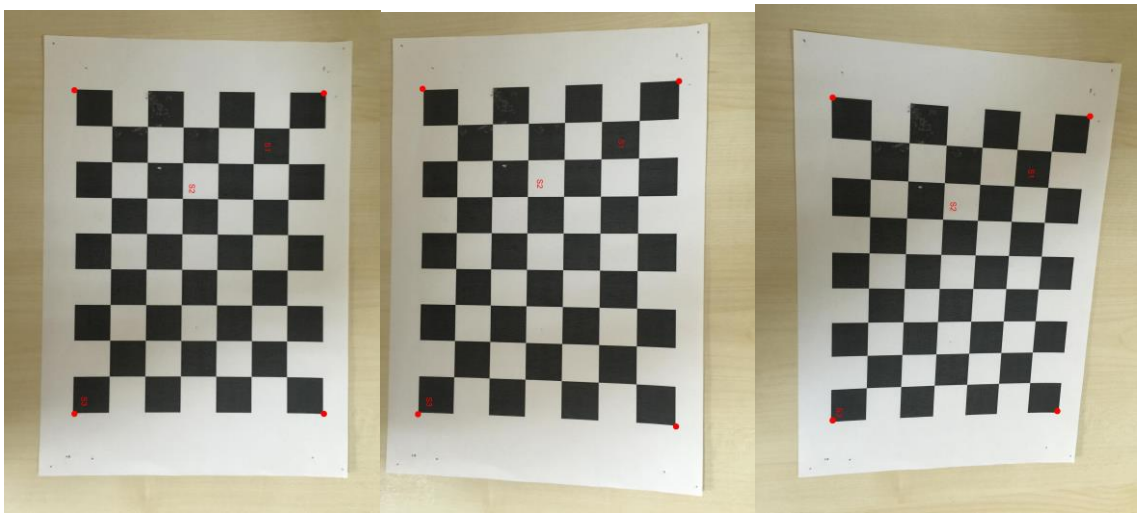
Finding the source point of a point from projection matrix;

```
static double[,] CalculateSourcePoints(double[,] destination, double[,] source)
{
    int dRows = destination.GetLength(0);
    int dCols = destination.GetLength(1);

    int sRows = source.GetLength(0);
    int sCols = source.GetLength(1);

    double[,] target = new double[dRows, dCols];

    for (int i = 0; i < dRows; ++i)
    {
        for (int j = 0; j < sCols; ++j)
        {
            for (int k = 0; k < dCols; ++k)
            {
                target[i,j] += destination[i, k] * source[k, j];
            }

        }

    }

    double[,] result = new double[2, 1];

    result[0, 0] = target[0, 0] / target[2, 0];
    result[1, 0] = target[1, 0] / target[2, 0];

    Console.Write("x = " + result[0, 0].ToString("0.0000") + "    ");
    Console.Write("y = " + result[1, 0].ToString("0.0000"));
    Console.WriteLine();

    return result;
}
```

Result of the program;







```
Homography Matrix for Image 1
1,858342539680262E-16
-0,5567120680425862
1096,7227740438946
0,33723904121810516
0,004163444953309941
-176,82150716707318
1,2183017520941E-19
5,947778504728516E-06

Projection of S1, S2, S3 for Image 1
S1: 850 710
S1: 698,507462686567 112,31343283582089
S1: 1120 1200
S2: 425,63042579578325 204,4233154195949
S1: 2500 1900
S3: 38,534376286537835 666,6529435981886
```

```
Homography Matrix for Image 2
0,00751598965980327
-0,531129935959433
1138,3216872760754
0,3087388080460341
-0,009243676887605834
-128,32072255374382
3,4079994773108005E-06
-3,4045607010315564E-05

Projection of S1, S2, S3 for Image 2
S1: 820 850
S1: 711,6293375025731 120,12864271778807
S1: 1080 1380
S2: 432,19466684558574 201,06760544835072
S1: 2500 2100
S3: 44,54398813070467 666,0602363640448
```

```
Homography Matrix for Image 3
0
-0,5265399438595152
1016,2220916488643
0,30952247868829225
0,022510725722784907
-222,96873828418433
-7,095365790528947E-05
5,353853908147805E-05

Projection of S1, S2, S3 for Image 3
S1: 1050 650
S1: 701,834878461124 121,48494514444812
S2: 1260 1150
S2: 422,45916169662706 198,4399576994726
S3: 2580 1880
S3: 28,69139740940302 673,4141211601759
```

References:

https://ros-developer.com/2017/12/26/finding-homography-matrix-using-singular-value-decomposition-and-ransac-in-opencv-and-matlab/

https://towardsdatascience.com/understanding-homography-a-k-a-perspective-transformation-cacaed5ca17

https://wordsandbuttons.online/interactive_guide_to_homogeneous_coordinates.html

Part 2:

Accept the first image source by going through the images is done, then the H matrix of all other images is first It is calculated according to the image. Teapot from the first image for each image by giving the x, y of the center it creates. target points are found.
Get scale by proportioning the distances between while being rotated from their positions to each other is obtained (applied for each image). Finally, this Using the information, lines are drawn for our 3D object.

First do the calculation for the R,T and S matrices. The distance between the origin of the reference image and one of the selected points is found and the origin is found.
The distance between the projection of the selected point and the projection of the selected point is found and these distances are calculated.
was proportioned and thus the scale factor was obtained. Same 2 for the rotation calculation.
There is a direction vector between the points and the angle between these two vectors is with the dot product.
found (making Unity dot product). Using the found rotation angle, the teapot
rotated around the calculated origin axis. The point where the Teapot projects is found and the cylinder
moved to this point.