Dani Lerner
ENPM673
HW1 Writeup

Problem 1:

- Resolution -> 9MP
- Square camera with length 14 mm
- Focal length -> 15mm

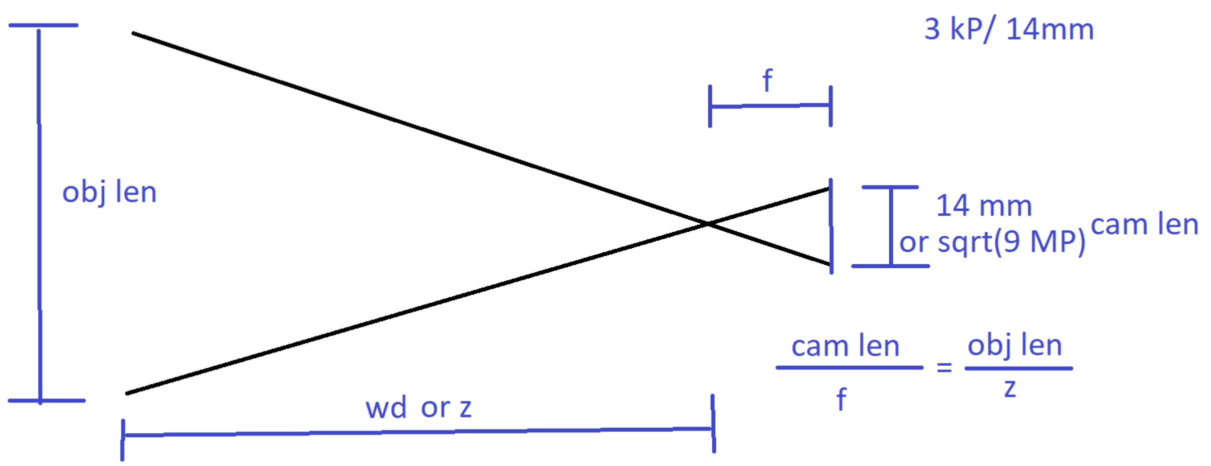
1. Compute Field of View

$$FOV = 2 * \tan^{-1}\left(\frac{w}{2f}\right) = 2 * \tan^{-1}\left(\frac{14}{2*15}\right) = 50.03° *$$

The field of view (FOV) will be identical horizontally and vertically since the camera is square.

*50.03 was computed using the atan function not atan2. With atan2, I received 130° but after research, the typical camera has a FOV of about 1 radian so I went with 50 deg.

2. Compute min. number of pixels a 5cm square obj at 20m will occupy.



*The "cam len" on this diagram is vague. For the similar triangles, the "cam len" corresponds to the length of the object on the camera sensor, not the total length of the sensor itself.

$$\frac{cam\ len}{f} = \frac{obj\ len}{z} \rightarrow cam\ len = f\frac{obj\ len}{z} = 15\ mm * \frac{5e1\ mm}{20e3\ mm} = .0375\ mm$$

The camera has a resolution of 9MP which is the total number of pixels. Since we know that the camera is square, each side length must be $\sqrt{9e6}\ px = 3{,}000\ pixels$. Therefore, the pixel density must be 3000 px per 14 mm of camera length.

$$px\ density = \frac{3000\ px}{14\ mm} = 214.3\ \frac{px}{mm} \rightarrow cam\ len\ px = px\ density * cam\ len\ mm$$

$$\therefore cam\ len\ px = 214.3 * .0375 \approx 8px \rightarrow Object\ area = cam\ len^2 = 64.5\ px$$
$$\rightarrow \min area = 64\ pixels$$

Problem 2:

For this problem, I needed to implement computer vision and path estimation for a red ball undergoing a parabolic arc in a video feed.

1. Computer Vision

With a red ball on a white background, the computer vision was relatively straightforward. I used the OpenCV library to separate the video frame into individual frames. Then, I converted each frame into an HSV (hue-saturation-value) image since it is easier and more practical to use for thresholding. The HSV color scheme is a color wheel (as opposed to the RGB cube) and red falls at a hue of about 0 or 180. I then thresholded the image. The OpenCV inRange function takes an image along with threshold values and returns a black and white image. Any pixel that falls into the threshold will be converted to white and all pixels are converted to black. Therefore, to ensure the ball color fell into the threshold, I created two thresholds (160-179 and 0-20) and then used a bitwise-or to combine them. With a binary black and white image, the next goal was to find the top and bottom of the ball. To do this, I merely had to take the maximum and minimum y-values of the white pixels. I then noticed that the top and bottom datapoints did not look accurate. This was because the ball was heavily pixelated and was not a perfect circle. Because of this, I took the average x-value of all white pixels at that max/min y-value and rounded it to the nearest integer.

2. Path Estimation

The computer vision algorithm returned two arrays – one containing the upper trajectory of the ball and the other containing the lower trajectory from each frame in the video. Using these datapoints, I could estimate an equation for the path the ball followed. First, I created the Polynomial Regression class as a base class. This class generates the proper X matrix to prepare the least squares parabolic estimation, uses a given least squares estimation technique, and predicts the y for all given x. This class takes in the x,y data as inputs as well as the degree polynomial and the estimation technique. The three estimation techniques were Ordinary Least Squares (OLS), Total Least Squares (TLS), and RANSAC. Each technique is truly attempting to estimate a line of best fit. However, we add an additional column of x squared to the X-matrix to force a higher order estimation as shown below.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ 1 & x_3 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}$$[1]

Consequently, the 2D result is a parabolic estimation.

The OLS method is the simplest. It follows the formula $w = (X^T X)^{-1} X^T y$ where X is the above-mentioned matrix, y is the list of y data points and I used the pseudoinverse rather than the inverse itself. This method assumes no measurement error and simply optimizes the error in the y-direction. The TLS method is very similar but de-regularizes the least squares the inner product first. The equation for TLS is $(X^T X - \sigma_{n+1}^2 I)^{-1} X^T y$ where σ_{n+1}^2 is the smallest eigen value of the [X y] matrix.[2] Lastly, RANSAC is an iterative method where we select a few random datapoints and fit them with another fitting technique such as OLS. We then give the algorithm a threshold and if a data point falls within that threshold from the estimated line, it is considered an inlier. For each random point selection, we count the number of total inliers and finally, select the iteration with the largest number of inliers. The data and estimation results for each scenario are shown in the graphs below.

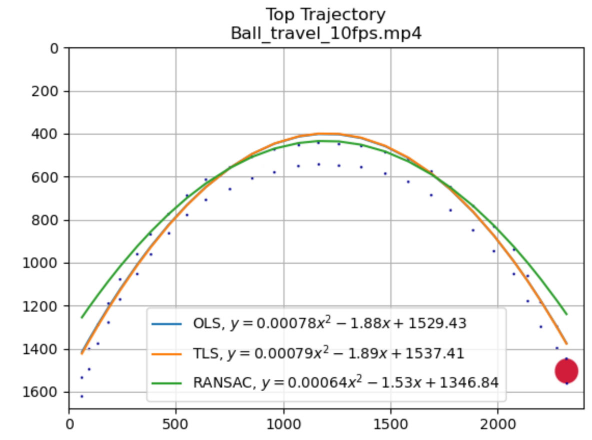


Figure 1: Top trajectory of low-noise video

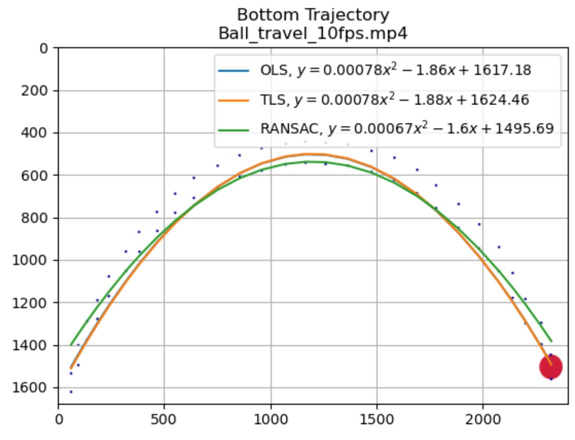


Figure 2: Bottom trajectory of low-noise video

[1]https://en.wikipedia.org/wiki/Polynomial_regression#:~:text=In%20statistics%2C%20polynomial%20regression%20is,nth%20degree%20polynomial%20in%20x.
[2] http://people.duke.edu/~hpgavin/SystemID/References/Markovsky+VanHuffel-SP-2007.pdf

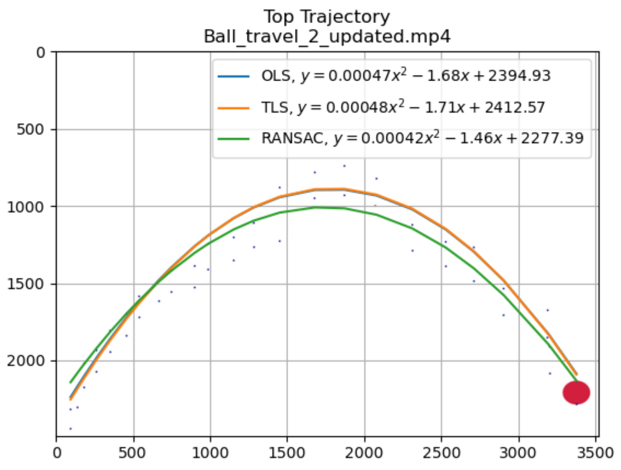


Figure 3: Top trajectory of high-noise video

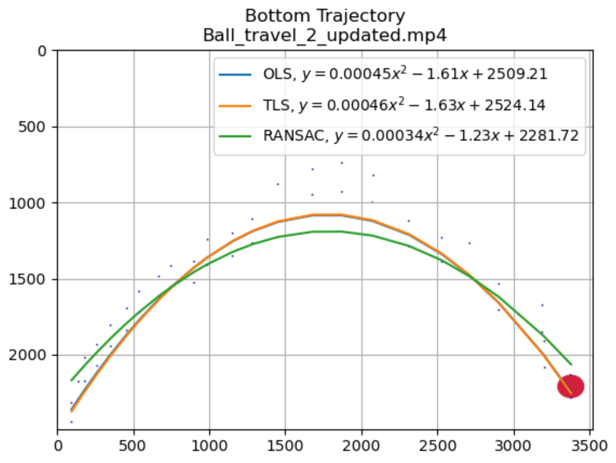


Figure 4: Bottom trajectory of high-noise video

*larger images can be seen in the submission folder.

The OLS and TLS estimation equations are very similar. This is because without much noise, the smallest eigen value of the [X y] matrix is very small. Notice that the two equation diverge a little more in the high-noise video than in the low-noise video. If we had extremely unclear lines, I believe that TLS and OLS would diverge far more. Compared to RANSAC, however, the LS methods were significantly better for the low-noise video. You can see that the RANSAC estimate tails off towards either end of the parabola while the LS methods have good results throughout. However, in the higher noise video, the middle noisy section throws off the LS methods while the RANSAC method works quite well. Thus, I can draw the conclusion that RANSAC is better with noisy data while least squares are better if we are dealing with a low noise, exact equations. TLS will handle more noisy, high-variance data than OLS but here, the difference between the two is small.

Problem 3:

To compute the SVD, the first step is to get the eigen values and vectors of the transpose(A)*A matrix using the numpy.linalg eig function. I then sorted the eigen values and its corresponding vectors according to descending order. Lastly, I calculated the U, Sigma, and V matrices. The V matrix is simply the matrix of sorted eigen vectors while sigma is the diagonal matrix of sorted singular values i.e.,

$$\sum = \sigma_{sorted} * I \ where\ I\ is\ the\ identity\ matrix,$$

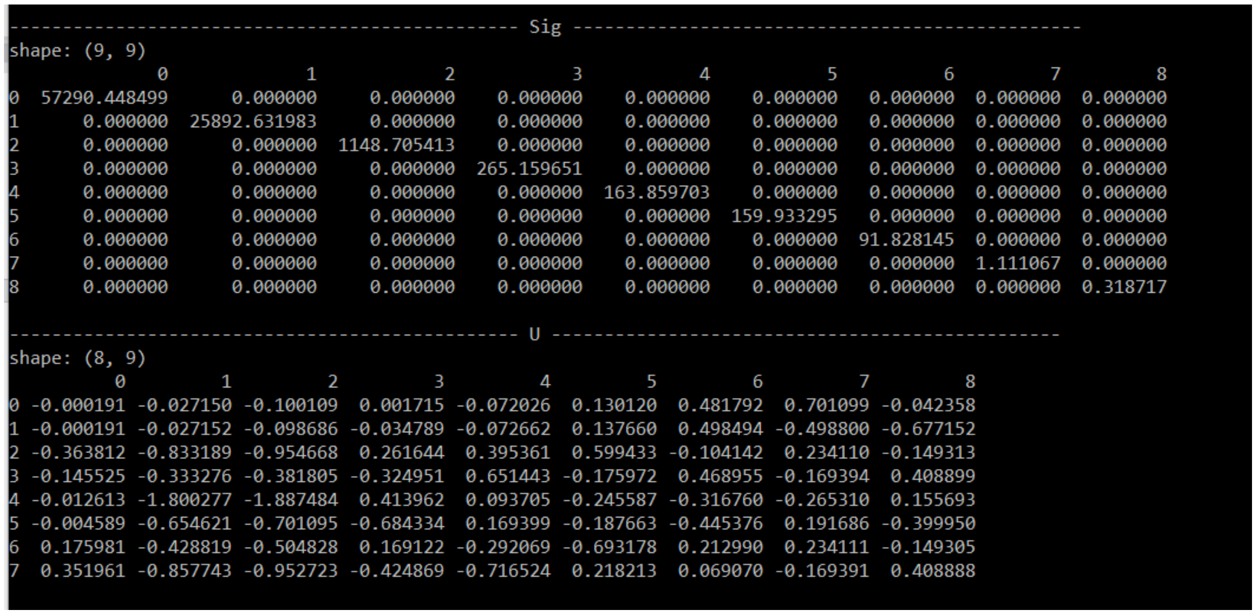
$$\sigma = singular\_vals \ = \sqrt{|\lambda|}\ and\ \lambda\ is\ a\ list\ of\ eigen\ values.^{3}$$

[3] https://courses.physics.illinois.edu/cs357/fa2019/assets/lectures/16-SVD-Intro.pdf

While the U matrix is the sorted eigen vectors of the A*transpose(A) matrix, it can also be calculated without finding these eigen vectors. When n > m, the equation is merely

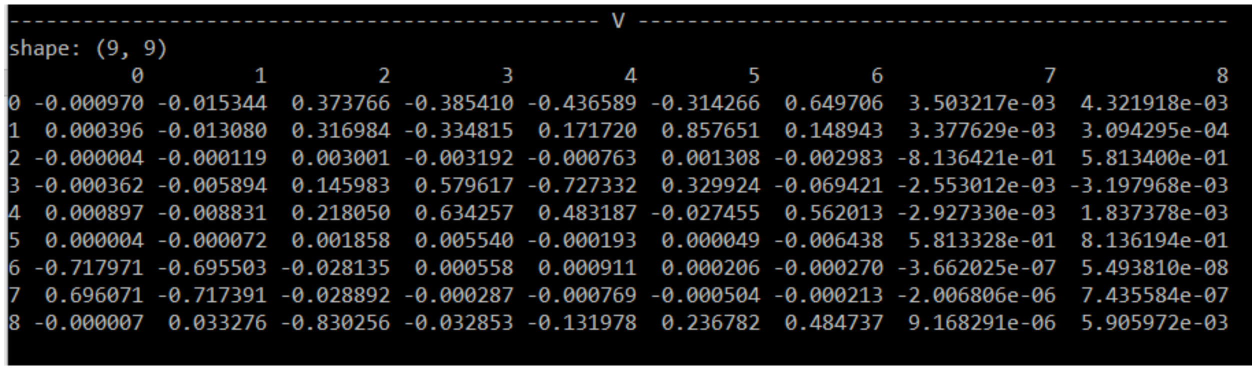
$$U = [\frac{1}{\sigma_1}Av_1, \frac{1}{\sigma_2}Av_2, \ldots]^4$$

To check if the SVD is correct, we can multiply U*∑*V^T and it should equal the original A matrix. The results are shown below.



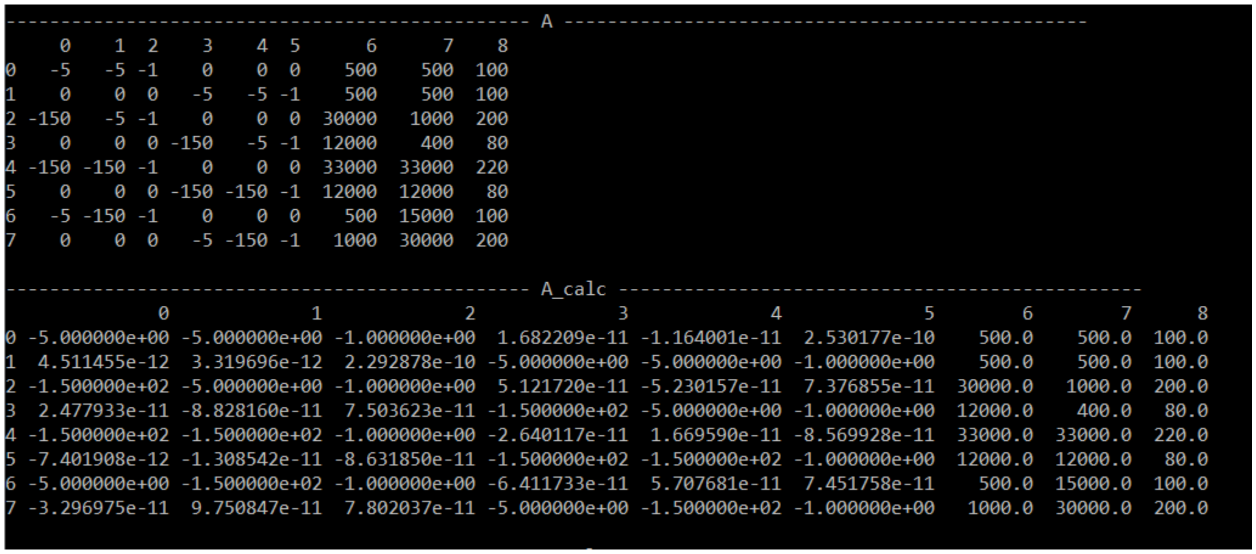
```
---------------------------------------------- Sig ----------------------------------------------
shape: (9, 9)
              0            1            2           3           4           5          6         7         8
0  57290.448499     0.000000     0.000000    0.000000    0.000000    0.000000   0.000000  0.000000  0.000000
1      0.000000 25892.631983     0.000000    0.000000    0.000000    0.000000   0.000000  0.000000  0.000000
2      0.000000     0.000000  1148.705413    0.000000    0.000000    0.000000   0.000000  0.000000  0.000000
3      0.000000     0.000000     0.000000  265.159651    0.000000    0.000000   0.000000  0.000000  0.000000
4      0.000000     0.000000     0.000000    0.000000  163.859703    0.000000   0.000000  0.000000  0.000000
5      0.000000     0.000000     0.000000    0.000000    0.000000  159.933295   0.000000  0.000000  0.000000
6      0.000000     0.000000     0.000000    0.000000    0.000000    0.000000  91.828145  0.000000  0.000000
7      0.000000     0.000000     0.000000    0.000000    0.000000    0.000000   0.000000  1.111067  0.000000
8      0.000000     0.000000     0.000000    0.000000    0.000000    0.000000   0.000000  0.000000  0.318717

---------------------------------------------- U ----------------------------------------------
shape: (8, 9)
          0         1         2         3         4         5         6         7         8
0 -0.000191 -0.027150 -0.100109  0.001715 -0.072026  0.130120  0.481792  0.701099 -0.042358
1 -0.000191 -0.027152 -0.098686 -0.034789 -0.072662  0.137660  0.498494 -0.498800 -0.677152
2 -0.363812 -0.833189 -0.954668  0.261644  0.395361  0.599433 -0.104142  0.234110 -0.149313
3 -0.145525 -0.333276 -0.381805 -0.324951  0.651443 -0.175972  0.468955 -0.169394  0.408899
4 -0.012613 -1.800277 -1.887484  0.413962  0.093705 -0.245587 -0.316760 -0.265310  0.155693
5 -0.004589 -0.654621 -0.701095 -0.684334  0.169399 -0.187663 -0.445376  0.191686 -0.399950
6  0.175981 -0.428819 -0.504828  0.169122 -0.292069 -0.693178  0.212990  0.234111 -0.149305
7  0.351961 -0.857743 -0.952723 -0.424869 -0.716524  0.218213  0.069070 -0.169391  0.408888
```



```
---------------------------------------------- V ----------------------------------------------
shape: (9, 9)
          0         1         2         3         4         5         6             7             8
0 -0.000970 -0.015344  0.373766 -0.385410 -0.436589 -0.314266  0.649706  3.503217e-03  4.321918e-03
1  0.000396 -0.013080  0.316984 -0.334815  0.171720  0.857651  0.148943  3.377629e-03  3.094295e-04
2 -0.000004 -0.000119  0.003001 -0.003192 -0.000763  0.001308 -0.002983 -8.136421e-01  5.813400e-01
3 -0.000362 -0.005894  0.145983  0.579617 -0.727332  0.329924 -0.069421 -2.553012e-03 -3.197968e-03
4  0.000897 -0.008831  0.218050  0.634257  0.483187 -0.027455  0.562013 -2.927330e-03  1.837378e-03
5  0.000004 -0.000072  0.001858  0.005540 -0.000193  0.000049 -0.006438  5.813328e-01  8.136194e-01
6 -0.717971 -0.695503 -0.028135  0.000558  0.000911  0.000206 -0.000270 -3.662025e-07  5.493810e-08
7  0.696071 -0.717391 -0.028892 -0.000287 -0.000769 -0.000504 -0.000213 -2.006806e-06  7.435584e-07
8 -0.000007  0.033276 -0.830256 -0.032853 -0.131978  0.236782  0.484737  9.168291e-06  5.905972e-03
```

4 https://www.youtube.com/watch?v=Ls2TgGFfZnU&ab_channel=JonathanDavid%27sNovels

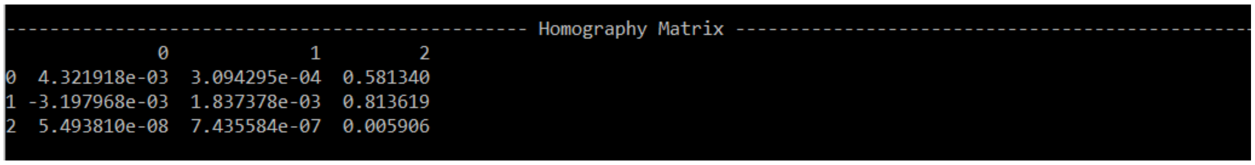


```
---------------------------------------------- A ---------------------------------------------
     0    1  2    3    4  5      6      7    8
0   -5   -5 -1    0    0  0    500    500  100
1    0    0  0   -5   -5 -1    500    500  100
2 -150   -5 -1    0    0  0  30000   1000  200
3    0    0  0 -150   -5 -1  12000    400   80
4 -150 -150 -1    0    0  0  33000  33000  220
5    0    0  0 -150 -150 -1  12000  12000   80
6   -5 -150 -1    0    0  0    500  15000  100
7    0    0  0   -5 -150 -1   1000  30000  200

---------------------------------------------- A_calc ----------------------------------------------
             0             1             2             3             4             5        6        7      8
0 -5.000000e+00 -5.000000e+00 -1.000000e+00  1.682209e-11 -1.164001e-11  2.530177e-10    500.0    500.0  100.0
1  4.511455e-12  3.319696e-12  2.292878e-10 -5.000000e+00 -5.000000e+00 -1.000000e+00    500.0    500.0  100.0
2 -1.500000e+02 -5.000000e+00 -1.000000e+00  5.121720e-11 -5.230157e-11  7.376855e-11  30000.0   1000.0  200.0
3  2.477933e-11 -8.828160e-11  7.503623e-11 -1.500000e+02 -5.000000e+00 -1.000000e+00  12000.0    400.0   80.0
4 -1.500000e+02 -1.500000e+02 -1.000000e+00 -2.640117e-11  1.669590e-11 -8.569928e-11  33000.0  33000.0  220.0
5 -7.401908e-12 -1.308542e-11 -8.631850e-11 -1.500000e+02 -1.500000e+02 -1.000000e+00  12000.0  12000.0   80.0
6 -5.000000e+00 -1.500000e+02 -1.000000e+00 -6.411733e-11  5.707681e-11  7.451758e-11    500.0  15000.0  100.0
7 -3.296975e-11  9.750847e-11  7.802037e-11 -5.000000e+00 -1.500000e+02 -1.000000e+00   1000.0  30000.0  200.0
```

A_calc is the matrix product of U∑*V^T while A is the original matrix.

To find the homogenous matrix, we merely take the last column of the V matrix which is also the eigen vector corresponding to the smallest eigen value. Then, we reshape the vector as a matrix. The result is shown below.



```
---------------------------------------------- Homography Matrix ----------------------------------------------
             0             1         2
0  4.321918e-03  3.094295e-04  0.581340
1 -3.197968e-03  1.837378e-03  0.813619
2  5.493810e-08  7.435584e-07  0.005906
```