**3D Reconstruction – A Computer Vision Project**

Presenters:

Nisan Kogan

Daniel Spevak

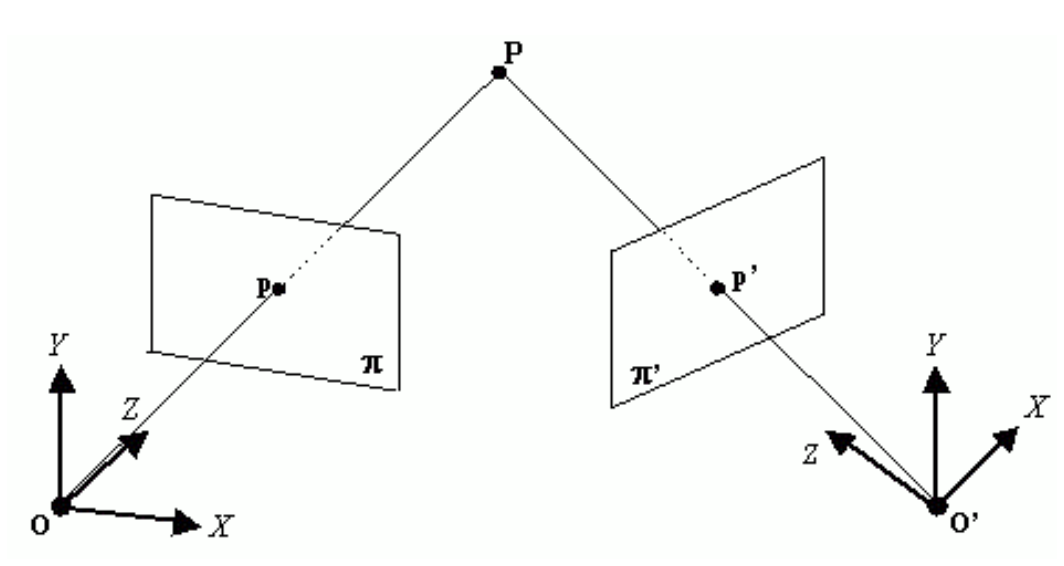


Figure Projection from 2 images

**Background**

Given a set of at least two images of the same object, with a small angle difference between each picture of the object, we can reconstruct the 3-dimensional object via those images.

We can realize this reconstruction using the techniques and theory learned in computer vision.

In our project, we reconstructed the object with 3 images.

**3D Reconstruction Steps:**

1. **Find keypoints using SIFT**

First step in the 3d reconstruction process we would want to find corresponding points between each pair of images.

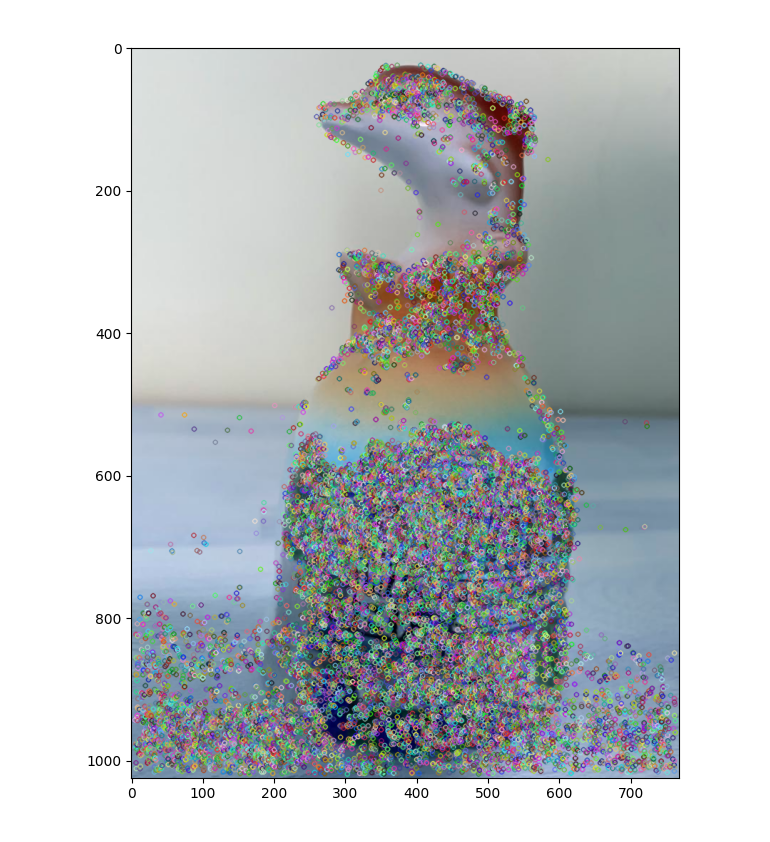
We did that with the SIFT algorithm that detects and describes features around the points for each image.

Parameters chosen:

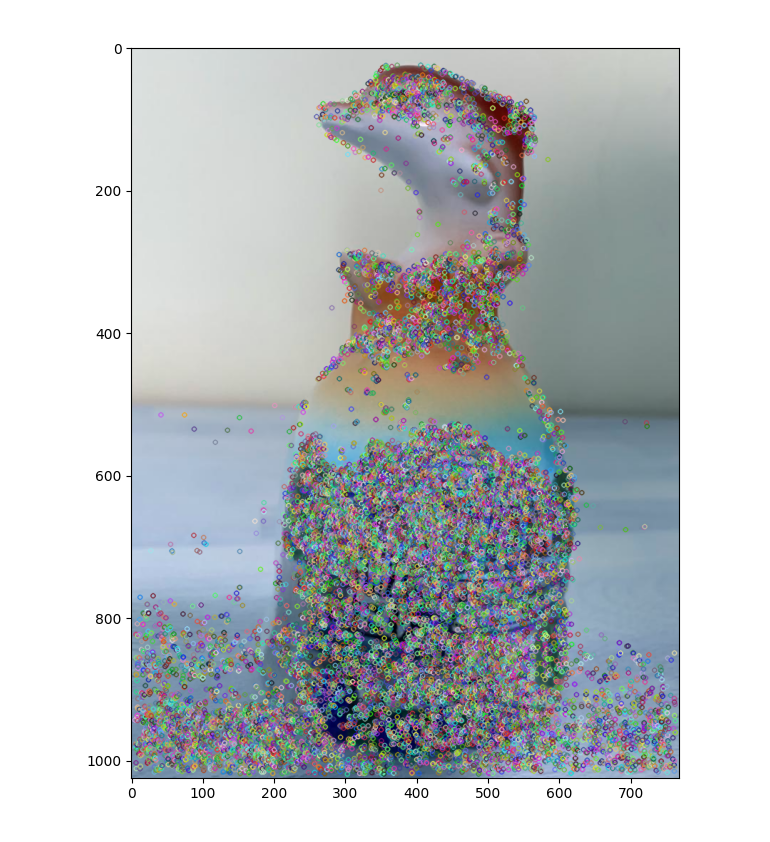
nfeatures=0, nOctaveLayers=3, contrastThreshold=0.01, edgeThreshold=20, sigma=0.5

Keypoints found by SIFT algorithm in:

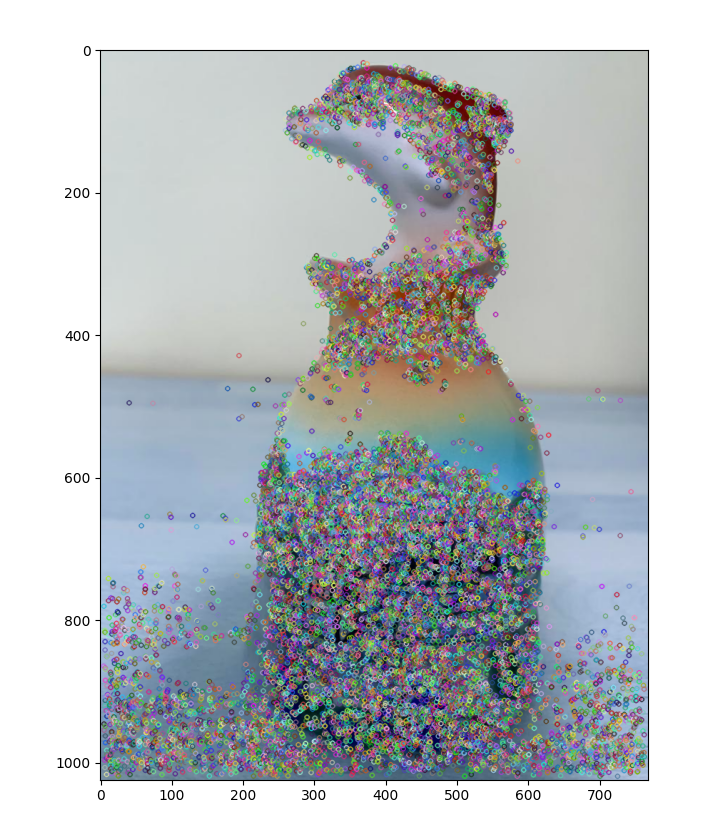
* Middle Image:



* Left Image:



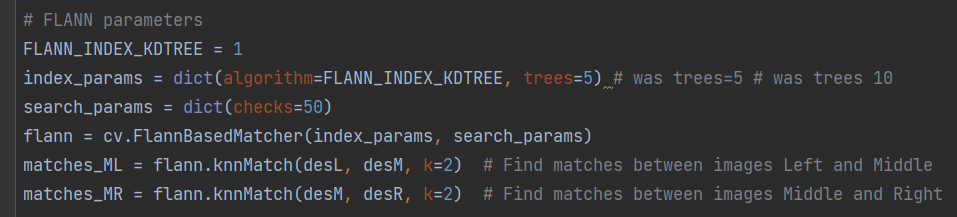
* Right Image:



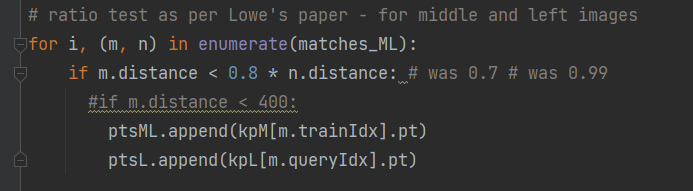
\*\* We know that the keypoints we found also contain noisy keypoints. Therefore we filter some of them later. Yet in the end some of the noise is still left in the reconstruction, but we left it this way because with too much filtering we will get a small amount of points in the final reconstruction.

2) Match points Based on KNN ( K Nearest Neighbors) .

In order for us to achieve the relation (rotation, translation etc.) between pair of images we would want to match those points – with the help of the SIFT descriptor we apply the FLANN matcher which is based on KNN search.

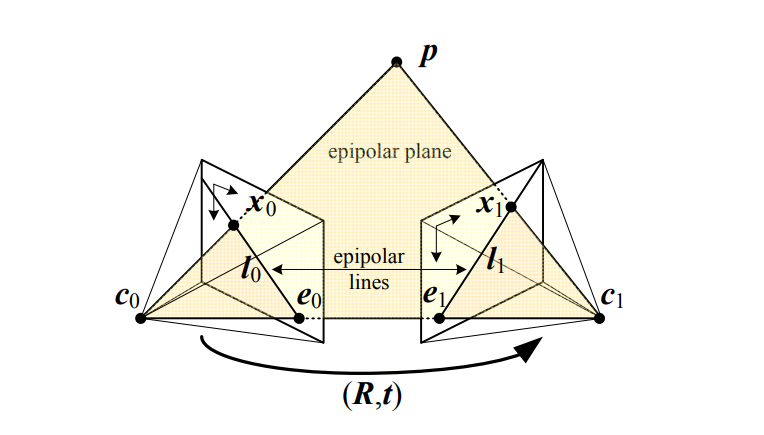


We would want to ensure the validity of these points with the Lowe ratio test, which proposes filtering these points by eliminating matches when the second-best match is almost as good.



3) Find the Fundamental matrix using the 7-point algorithm

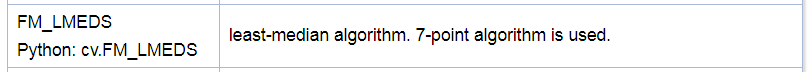
The epipolar constraint:



**The Epipolar Geometry constraint** says that given 2 cameras located at c0,c1 and we have a point p such that c0,c1 and p are coplanar – the points satisfy the condition:

Where F is the fundamental matrix.

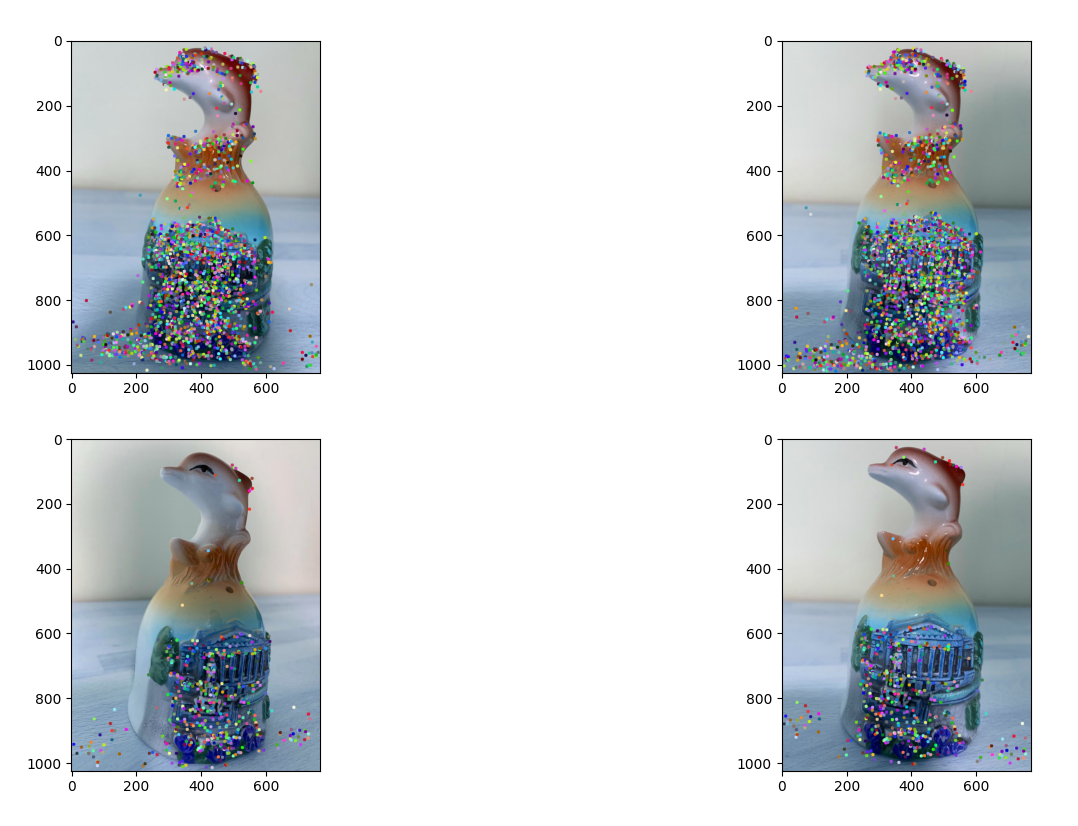
Using the epipolar constraint between points on the image planes, we can find F via the 7-point algorithm. This is done with applying the cv.FM\_LMEDS parameter inside the cv.findFundamentalMat function.



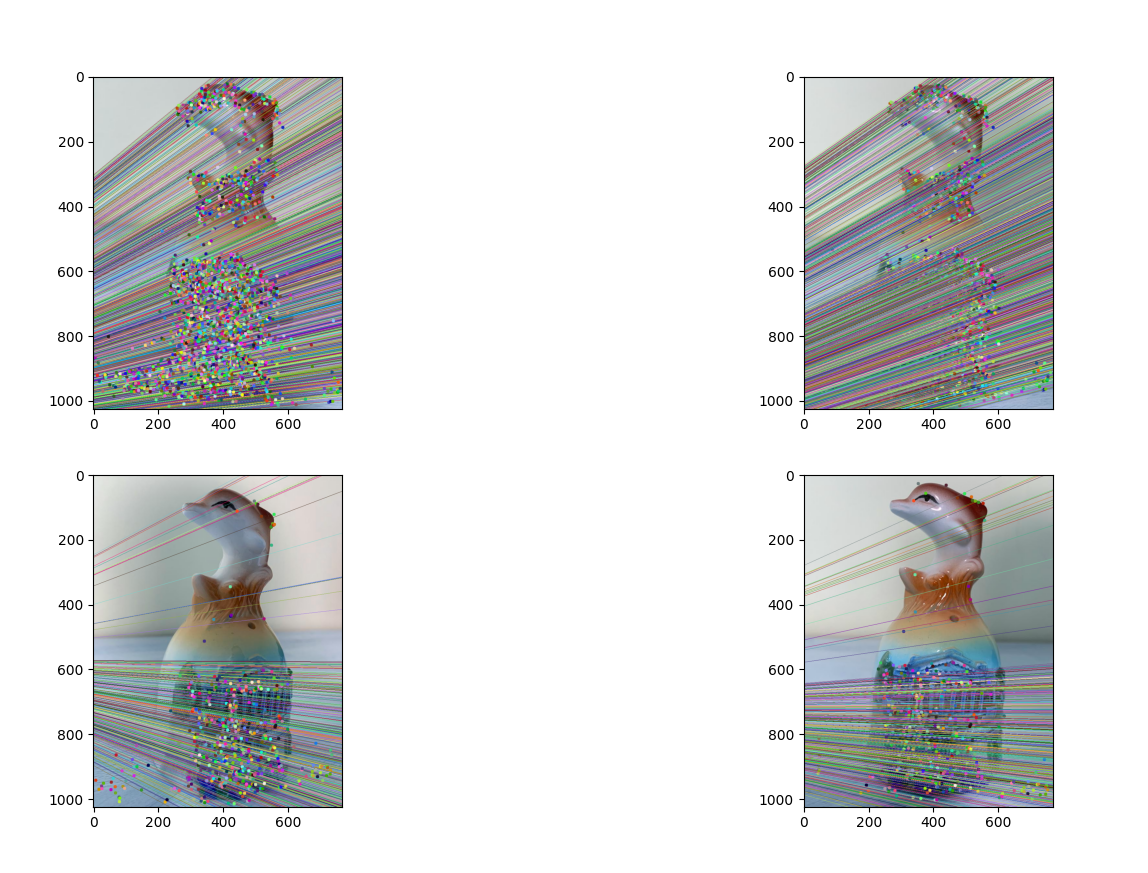
maxIters, confidence and ransacReprojThreshold parameters were set according to trial and error until we got the best results.

Now that we have the fundamental matrix, we can use it to draw the epipolar lines through the matching points in each image:

Matching points between left and middle images:

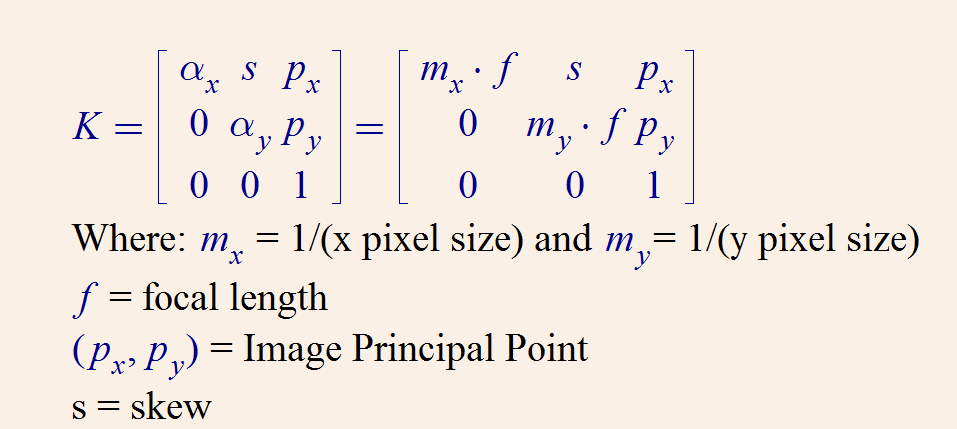


Epipolar lines between left and middle images:

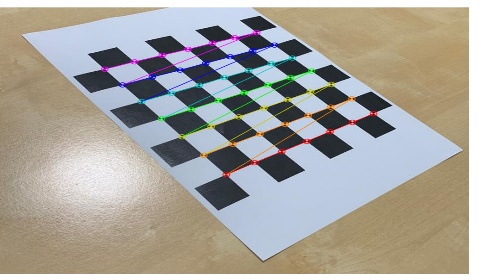


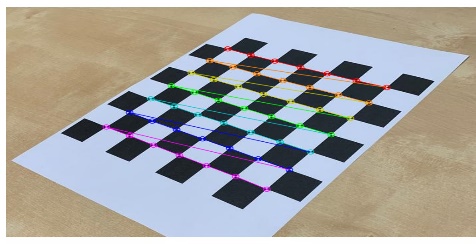
4) Calibrate the phone camera and find K. And then find E using K and F.

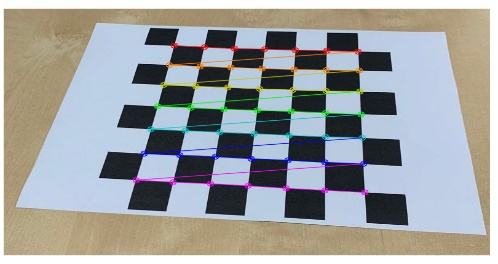
In order to find the intrinsic camera matrix K:

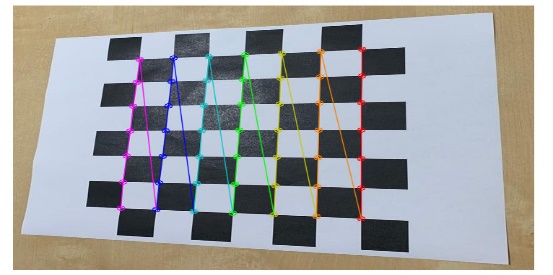
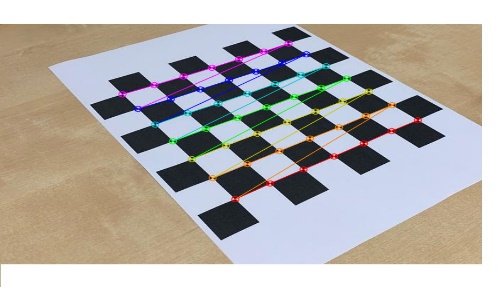


We calibrate the camera using pictures of a chessboard pattern:









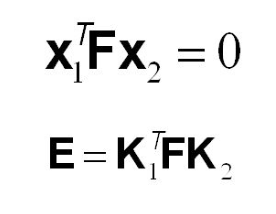
Resulting K matrix:

K =

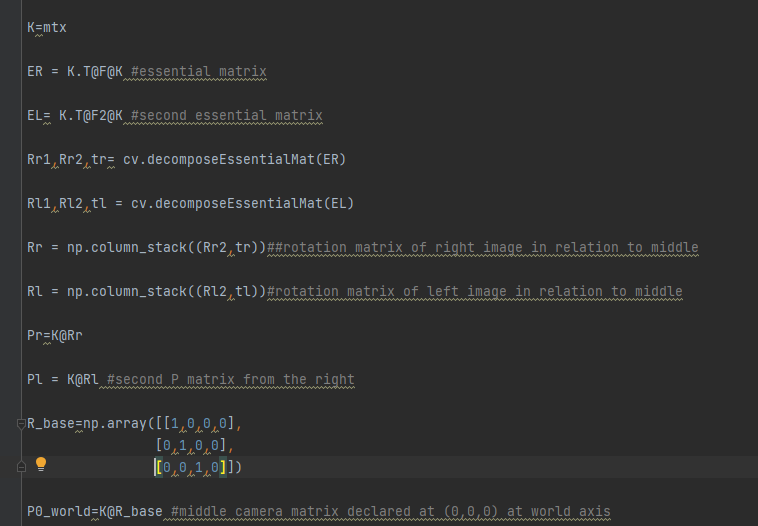
5) Find the rotation matrix R and the translation t from left image to right image via using SVD on E, and then finding the camera matrices

Camera calibration allowed us to retrieve the intrinsic parameters of the camera.

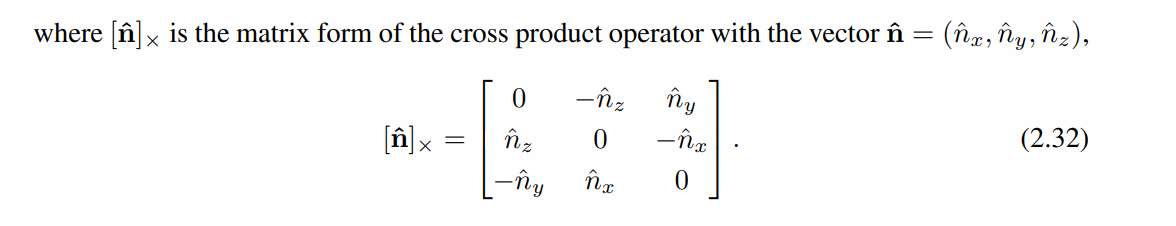
Now we able to construct the essential matrix using the relation with the fundamental matrix:



Later we proceeded to construct the camera matrices:



Since the essential matrix E decomposes into the 3x3 rotation matrix R and 3x1 translation matrix t, we get that:



And after decomposition with our python code, we receive two rotation matrices R1,R2.

We chose the rotation matrix that gave us the best 3D reconstruction results.

Now that we have R and t, we have our camera matrix P:

Since the middle image is our point of reference, we set:

Where the rotation matrix is the 3x3 unit matrix and the translation vector is zero.

This means that the reference camera matrix has zero rotation angle and zero translation.

=> Reference camera is “zero”.

The left camera matrix has camera matrix:

The right camera matrix has camera matrix:

Both left and right camera matrices have rotations and translations with respect to the middle image.

6) Triangulation, finding the right scale for 3d reconstruction

The motivation (visual representation ):

Left image :

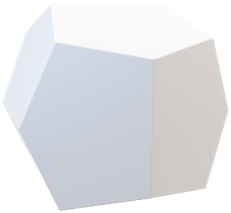
P\_left=K [R\_left|t\_left]

Middle image :

P\_mid=K [I|0]

Right image :

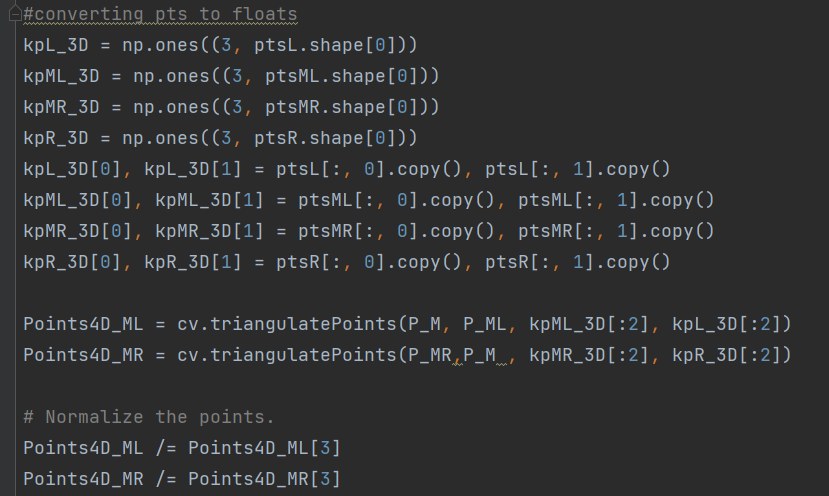
P\_right=K [R\_right|t\_right]



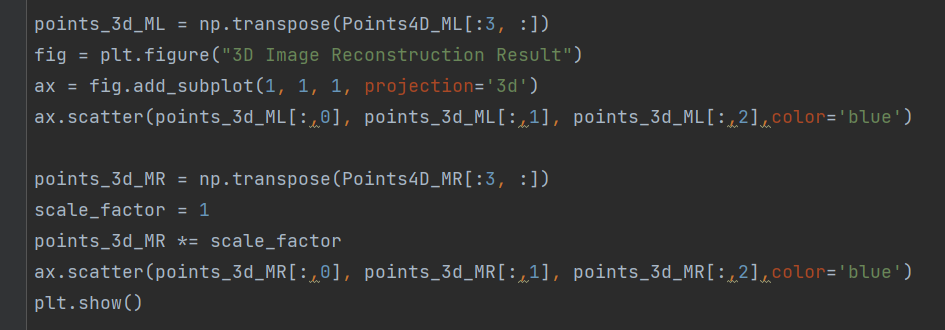
3D object(scatter) :

(X,Y,Z,1)

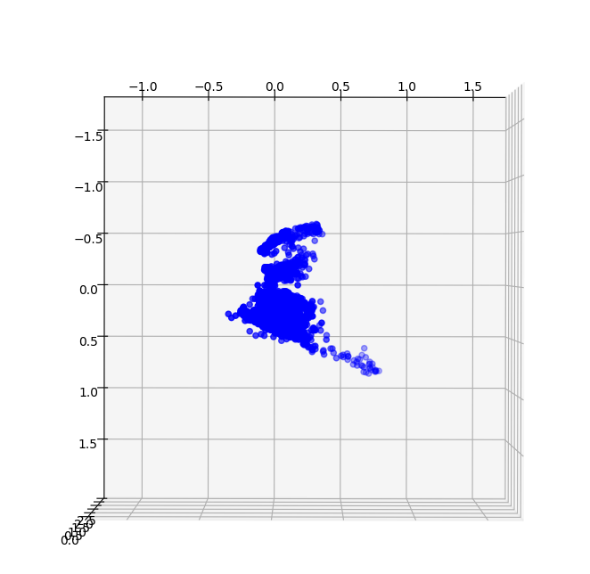
Now we have the 3 camera matrices P\_left, P\_mid and P\_right , with the corresponding points of each pair we use triangulation by solving a linear equation, thus getting the desired 3D points.

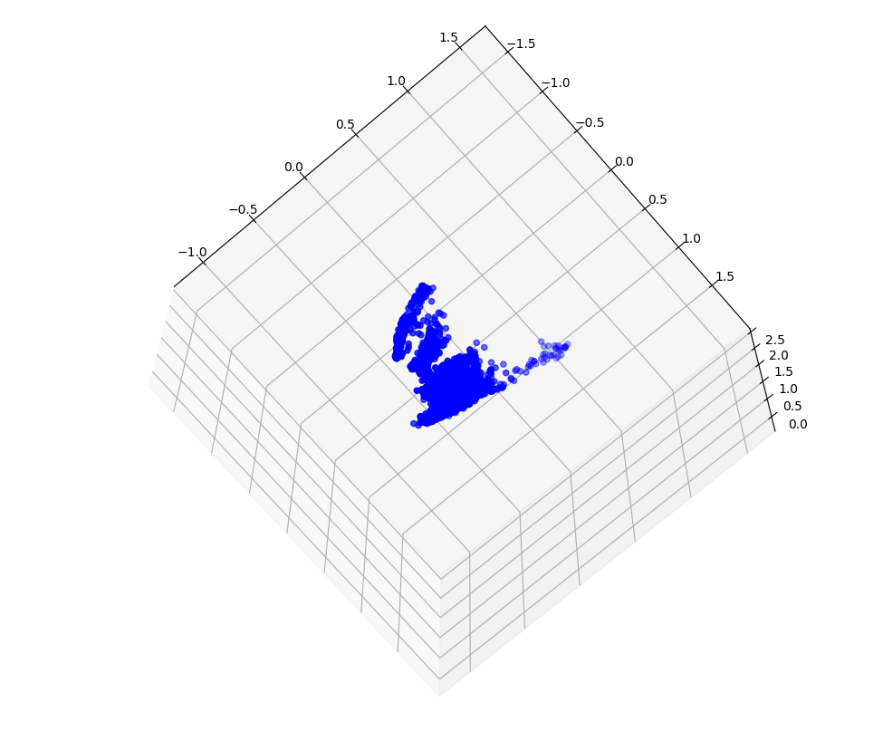


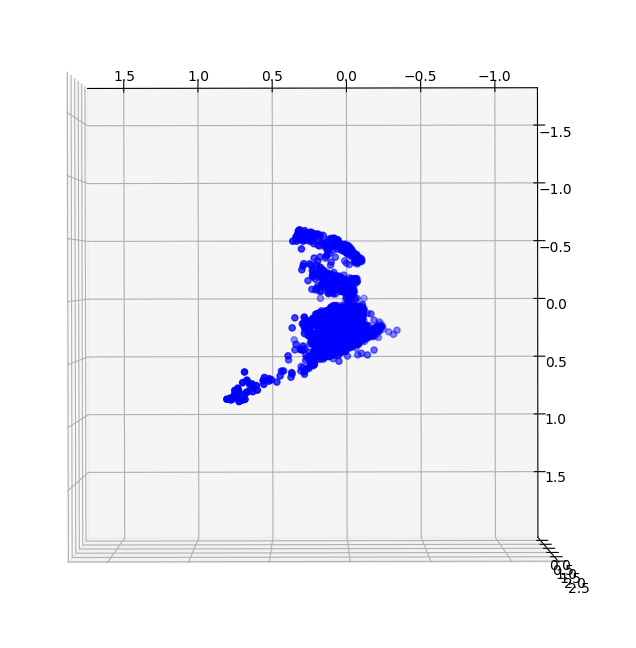
After normalizing the points, we can cut the fourth dimension to be left with 3D points (X, Y, Z) that we can show on a 3D plot as points of the reconstructed object:

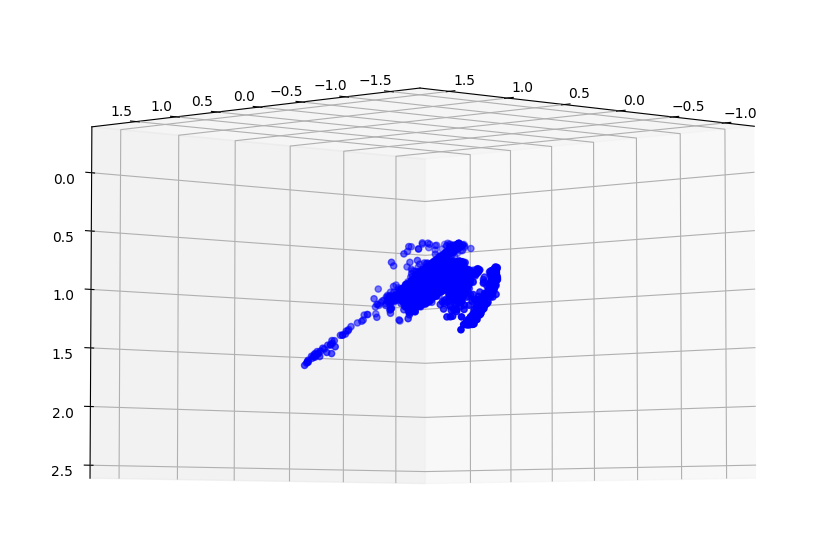


And finally, we get a 3D plot of the reconstructed object from 3 images:







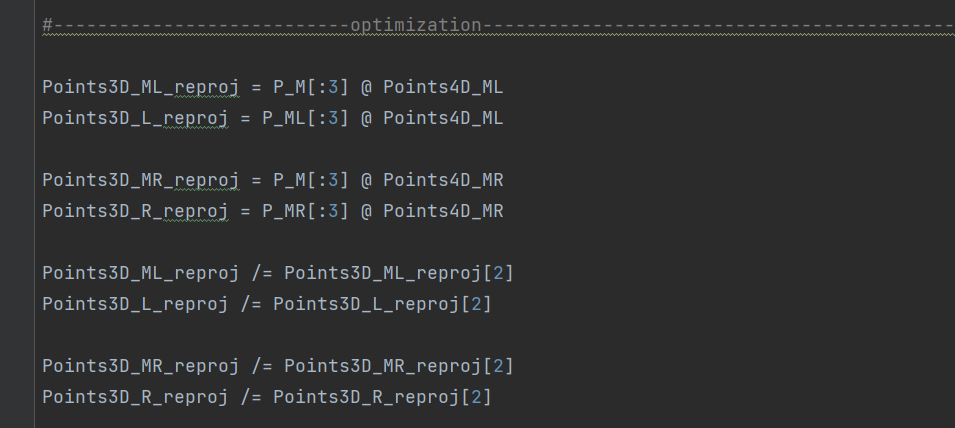


7) optimization

Since a point x on the image plane can be found via the reprojection equation:

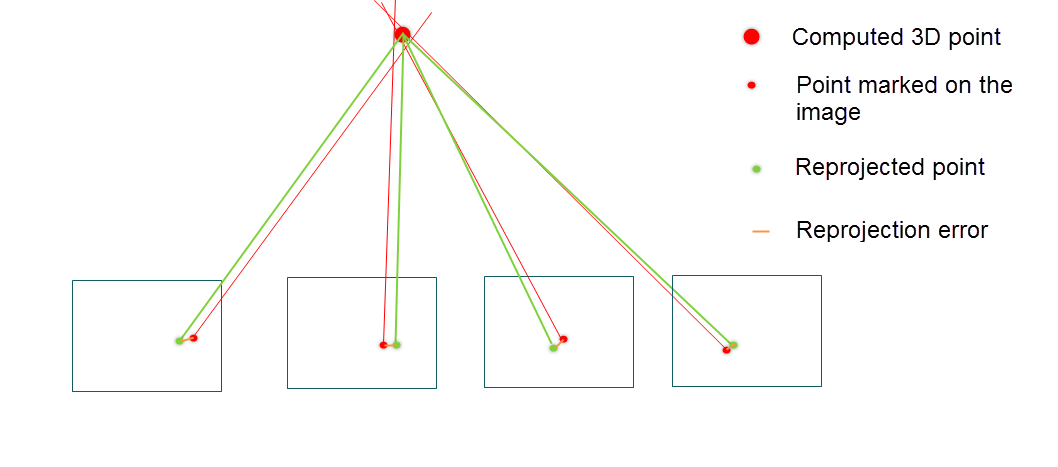
where X is a point in the 3D world and x is the **corresponding point on the image plane**, and P is the camera matrix that maps the 3D point onto the 2D image plane.

We realize this reprojection in the following code:



In order to optimize our reconstruction, we need to calculate the reprojection error for each image and for each reprojected point in each image.

Reprojection Error illustration:



We find the sum of errors of reprojections of all points via the formula:

Where are coordinates of the found keypoints in the image i and are the projection of the 3D points onto the 2D plane of image i.

The reprojection error is the sum of all the lengths of the orange lines in the above image.

Using the reprojection error as a function of the camera matrices , which themselves are functions of rotations and translations , we can optimize the translations and rotations using the reprojection error as a loss function.

When we insert the loss function as a parameter inside the optimizing function:

**scipy.optimize.minimize(*fun*, *x0*, *args=()*, *method=None*, *jac=None*, *hess=None*, *hessp=None*, *bounds=None*, *constraints=()*, *tol=None*, *callback=None*, *options=None*)**

Then the optimizing function tries to minimize the reprojection error and thus it may find better values for each camera matrix which in return can give better 3D reconstruction results.

Constructing the function with camera parameters:  R,t,f,k1,k2 . that should return the residuals.

We tried to use that approach instead of calculating the Jacobian because the jacobian got too big.

With the help of scipy library that iterates until the residuals reduce to minimum, we tried to achieve optimized camera params.

Unfortunately, we haven’t succeed in applying the optimization.

**References:**

Using SIFT, computing matches and epipolar lines: <https://docs.opencv.org/4.x/da/de9/tutorial_py_epipolar_geometry.html>

Camera Calibration: <https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>

Optimization: <https://scipy-cookbook.readthedocs.io/items/bundle_adjustment.html>

Cv Library functions: <https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html>

3D Reconstruction: <https://github.com/alyssaq/3Dreconstruction>