Assignment 5: Epipolar geometry and triangulation

General instructions: The assignment is composed of the compulsory tasks and optional tasks (denoted by ★). To approach the assignment defense, you are required to complete at least the compulsory parts. Successfully defending the compulsory part will give you a maximum 75 points out of 100. At the defense, the obligatory tasks **must** be implemented correctly and completely. If your implementation is incomplete or has major errors, you will not be able to successfully defend the assignment. You can choose arbitrarily among the optional tasks to reach the 100 points. The number of points for an optional task is given in the task description. There might be more than 25 points available for an assignment. However, the maximum possible number of points for a single assignment is 100. The maximum overall number of points obtained from the 6 assignments is 600.

Required formating and submission: Create a folder `assignment5` that you will use during this assignment. Unpack the content of the `assignment5.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as Python scripts to `assignment5` folder. In order to complete the assignment you have to present your solutions to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the assignment defense as well. The code must be submitted on the e-classroom **before** the defense. Submit the code as .py source files (not Jupyter notebooks). If you have more that one source file, submit a zip of .py files.

COMMON ERRORS AND DEBUG IDEAS

- check your x and y coordinates
- check your data type: float, uint8
- check your data range: [0,255], [0,1]
- perform simple checks (synthetic data examples)

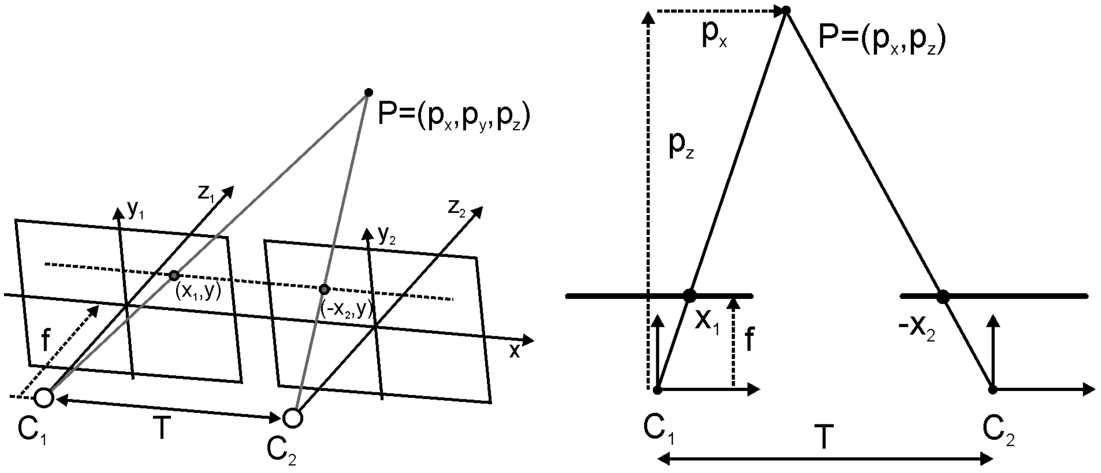
Introduction

In this assignment, we will review several parts of the epipolar geometry [2] (chapter 10.1) and robust estimation [2] (page 346).

Note: You can use feature point detection and matching that you implemented in Assignment 4. If you are unsure of your implementation or if you want to experiment with different detection and matching methods, you are allowed to use algorithms implemented in OpenCV, such as SIFT or ORB. Using functions such as `cv2.findFundamentalMat()` or `cv2.triangulatePoins()` is, of course, not allowed.

Exercise 1: Disparity

This exercise will focus on calculating disparity from a two-camera system. Our analysis will be based on a simplified stereo system, where two identical cameras are aligned with parallel optical axes and their image planes (CCD sensors) lie on the same plane (Image 1a).



(a) (b)

Figure 1: The left image shows the simplest stereo setup with two parallel cameras. The right image shows geometric relations when mapping of a point $\mathbf{p}$ in 3D space to axis x in image planes of the cameras.

In Figure 1b we can observe that we can write the following relations using similar triangles:

$$\frac{x_1}{f} = \frac{p_x}{p_z} \quad , \quad \frac{-x_2}{f} = \frac{T - p_x}{p_z}. \tag{1}$$

(a) Using the equations (1) derive the expression for *disparity* which is defined as $d = x_1 - x_2$. What is the relation between the distance of the object (point $\mathbf{p}$ in Figure 1) to the camera and the disparity d? What happens to disparity when the object is close to the cameras and what when it is far away?

(b) Write a script that computes the disparity for a range of values of p_z. Plot the values to a figure and set the appropriate units to axes. Use the following parameters of the system: focal length is $f = 2.5mm$ and stereo system baseline is $T = 12cm$.

(c) In order to get a better grasp on the idea of distance and disparity, you will calculate the numbers for a specific case. We will take the parameters from a specification of a commercial stereo camera *Bumblebee2* manufactured by the company PointGray: $f = 2.5mm$, $T = 12cm$, whose image sensor has a resolution of 648×488 pixels that are square and the width of a pixel is $7.4\mu m$. We assume that there is no empty space between pixels and that both cameras are completely parallel and equal. Let's say that we use this system to observe a (point) object that is detected at pixel 550 in x axis in the left camera and at the pixel 300 in the right camera. How far is the object (in meters) in this case? How far is the object if the object is detected at pixel 540 in the right camera? Solve this task analytically and bring your solution to the presentation of the exercise.

(d) ★ (15 points) Write a script that calculates the disparity for an image pair. Use the images in the directory `disparity`. Since the images were pre-processed, we can limit the search for the most similar pixel to the same row in the other image. Since just the image intensity carries too little information, we will instead compare small image patches. A simple way of finding a matching patch is to use normalized cross correlation. NCC for matrices $\mathbf{X}$ and $\mathbf{Y}$ of equal size is defined as

$$NCC(\mathbf{X}, \mathbf{Y}) = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}. \tag{2}$$

where x_i denotes a specific cell in matrix $\mathbf{X}$ and y_i a specific cell in matrix $\mathbf{Y}$. A patch from the second image is considered a match if it has the highest NCC value. The difference in x axis between the point from the first image and the matching point from the second image is the disparity estimate in terms of pixels[1]. The disparity search is performed in a single direction. You can expect this to be quite slow, so save the results for the defense upfront.

Note: If you wish, you can also use different images, e.g. any from Middlebury stereo datasets.

Question: Is the disparity estimated well for all pixels? If not, what are the characteristics of pixels with more consistent disparity estimation?

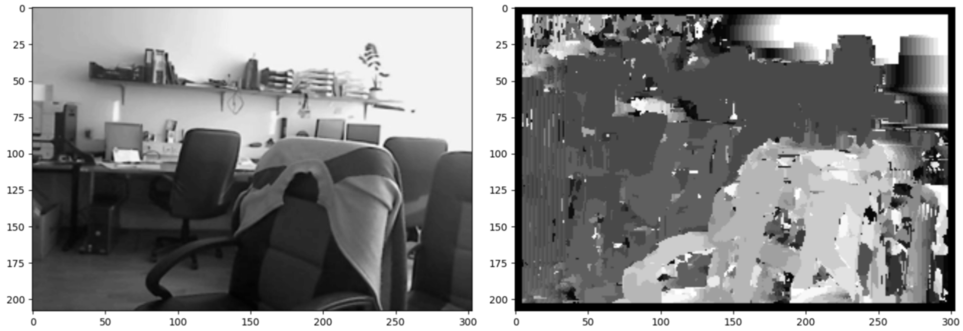


Figure 2: Disparity calculated with patch size 10. The image was reduced to 50% of its original size.

(e) ★ (5 points) Improve the noisy disparity estimation. You can implement a symmetric matching technique where the left image is compared to the right image and vice versa. The result can then be merged in some way that you choose. Smooth the results using median filter and choose an appropriate search window to obtain good results. Check your solution by using the disparity values to shift the pixel values from one image to another, then compare the result with the target image. Very noisy results will not be graded.

[1] Converting this estimate to distance requires information about the camera, which we do not have for the given image pairs.

Note: You cannot merge the results element-wise. The images were taken from different viewpoints and the results would be incorrect.

Exercise 2: Fundamental matrix, epipoles, epipolar lines

In the previous exercise, we were dealing with a special stereo setup where image planes of two cameras were aligned. In that case, the projection of a 3D point has the same y coordinate in both cameras and a different x coordinate. Such a setup ensures that the *epipolar lines in the cameras are parallel to the lines in the sensor*. This simplifies the search for correspondences (i.e. the projection of the same 3D point to both cameras). Generally the epipolar lines are not aligned with rows and in order to establish the relation between two image planes, the *fundamental matrix* needs to be computed.

In this exercise, you will implement a simple version of an *eight-point algorithm* [2] that can be used to estimate a fundamental matrix between two cameras. We will first revisit the theory that will be used for the task.

We are given a list of perfect correspondence pairs of points in left $\mathbf{x} = [u, v, 1]^T$ and right $\mathbf{x}' = [u', v', 1]^T$ image in homogeneous coordinates (u corresponds to column, v to row). The fundamental matrix rule states that each correspondence is a valid solution of the following equation:

$$\mathbf{x}'^T\mathbf{F}\mathbf{x} = 0 \quad ; \quad \mathbf{F} = \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix}, \tag{3}$$

where $\mathbf{F}$ denotes a fundamental matrix. Similarly to what we have done for the estimation of homography in the previous exercise, we can write a relation between a single pair of correspondence points as

$$\begin{bmatrix} uu' & u'v & u' & uv' & vv' & v' & u & v & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = 0. \tag{4}$$

If we combine $N \geq 8$ of these equations in a matrix $\mathbf{A}$ we get a system of linear equations:

$$\begin{array}{ccc} & \mathbf{Af} = \mathbf{0} & \\ \begin{bmatrix} u_1u_1' & u_1'v_1 & u_1' & u_1v_1' & v_1v_1' & v_1' & u_1 & v_1 & 1 \\ u_2u_2' & u_2'v_2 & u_2' & u_2v_2' & v_2v_2' & v_2' & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_Nu_N' & u_N'v_N & u_N' & u_Nv_N' & v_Nv_N' & v_N' & u_N & v_N & 1 \end{bmatrix} & \begin{bmatrix} F_{11} \\ F_{12} \\ \vdots \\ F_{33} \end{bmatrix} = & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \end{array} \tag{5}$$

We can solve the linear system above in a least-squares way by using the singular value decomposition method (SVD). Matrix $\mathbf{A}$ is decomposed to $\mathbf{A} = \mathbf{UDV}^T$. The solution according to least squares corresponds to the eigenvector $\mathbf{v}_n$ with the lowest eigenvalue, e.g. the last column of the matrix[2] $\mathbf{V}$.

Recall that the *epipole* of a camera is a point where all the epipolar lines (for that camera) intersect. This requires the rank of the matrix $\mathbf{F}$ to be 2, however, this is usually not true when dealing with noisy data – the fundamental matrix estimated using the approach above will not have a rank 2. In practice this means that the epipolar lines will not intersect in a single point but rather in a small neighborhood of such a point. To stabilize the system, we have to subsequently limit the rank of the fundamental matrix. This can be done by performing SVD decomposition to $\mathbf{F} = \mathbf{UDV}^T$, set the lowest eigenvalue (D_{33} in matrix $\mathbf{D}$) to 0, and then reconstruct back the corrected matrix $\mathbf{F}$ by multiplying $\mathbf{UDV}^T$. A newly created fundamental matrix will satisfy the rank condition rank $= 2$ and can be used to compute two epipoles (one for each camera)

$$\mathbf{Fe} = 0 \quad \text{in} \quad \mathbf{F}^T\mathbf{e}' = 0, \tag{6}$$

by decomposing the matrix $\mathbf{F}$ again and computing the *left* and *right*[3] eigenvector of the matrix $\mathbf{F}$,

$$\mathbf{e} = \begin{bmatrix} V_{13} & V_{23} & V_{33} \end{bmatrix} / V_{33}\ , \mathbf{e}' = \begin{bmatrix} U_{13} & U_{23} & U_{33} \end{bmatrix} / U_{33}, \tag{7}$$

which are then used to obtain both epipoles in homogeneous coordinates.

A simple eight-point algorithm for estimation of a fundamental matrix and the epipoles can be summarized as:

- Construct the matrix $\mathbf{A}$ as in equation (5)
- Decompose the matrix using SVD $\mathbf{A} = \mathbf{UDV}^T$, and transform the last eigenvector $\mathbf{v}_9$ in a 3×3 fundamental matrix $\mathbf{F}$
- Decompose $\mathbf{F} = \mathbf{UDV}^T$ and set the lowest eigenvalue to 0, reconstruct $\mathbf{F} = \mathbf{UDV}^T$.
- Decompose $\mathbf{F} = \mathbf{UDV}^T$ again, compute both epipoles following the equation (7).

Once we have the fundamental matrix, we can take any point $\mathbf{x}$ in the first image plane and determine an epipolar line for that point in the second image plane $\mathbf{l}' = \mathbf{Fx}$. Likewise, we can take the point $\mathbf{x}'$ in the second image plane and find the epipolar line in the first image plane $\mathbf{l} = \mathbf{F}^T\mathbf{x}'$. The line in the second image is then represented by all solutions of u' and v' in the equation $[u', v', 1]^T\mathbf{l}' = 0$.

(a) Solve the following task analytically. We are given a system of two cameras and a fundamental matrix that connects the left camera to the right one

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{8}$$

Compute the equation of the epipolar line in the right camera that corresponds to the point at column $= 0$ and row $= 2$ in the left camera. Take into account that the point has to be first written in homogeneous coordinates, i.e. $\mathbf{x} = [\text{column}, \text{row}, 1]^T$. Also compute the epipolar line for another point at column $= 1$ and row $= 0$ in the left camera, and the epipole.

[2] This is a solution in column form as seen in equation (4) that has to be reshaped to the form in equation (3).

[3] Attention: the terms left, and right eigenvector are mathematical terms and do not hold any relation to the left and right camera in our system.

Estimating a fundamental matrix

(b) Implement the function `fundamental_matrix()` that is given a set of (at least) eight pairs of points from two images and computes the fundamental matrix using the eight-point algorithm.

As the eight-point algorithm can be numerically unstable, it is usually not executed directly on given pairs of points. Instead, the input is first normalized by centering them to their centroid and scaling their positions so that the average distance to the centroid is $\sqrt{2}$. To achieve this, you can use the function `normalize_points()` from the supplementary material.

Extend the function `fundamental_matrix()` so that it first *normalizes* the input point-set of the left camera (we get transformed points and the transformation matrix $\mathbf{T}_1$) and then transform the input point set of the right camera (we get the transformed points and the transformation matrix $\mathbf{T}_2$). Using the transformed points, the algorithm computes the fundamental matrix $\hat{\mathbf{F}}$, then transforms it into the original space using both transformation matrices $\mathbf{F} = \mathbf{T}_2^T \hat{\mathbf{F}} \mathbf{T}_1$.

You can test your function using the data in the `house` directory. The directory contains both images and keypoints, as well as the perfect matches between the keypoints. The keypoints are stored in files with the extension `.corners` (in subdirectory 2D) and you can load them using `np.loadtxt()`. The matches are also stored in subdirectory 2D, in the file `house.nview-corners`. The file contains a line for each unique keypoint in the sequence, with the columns indicating the corresponding keypoint index for each of the images. If a keypoint is not visible in an image, this is marked with an asterisk (as described in the file `README-2D`). You can use the function `read_matches()` from the supplementary material to load the matches for a pair of images. Display the loaded points and matches to verify that you are using them correctly.

Compute the fundamental matrix $\mathbf{F}$ from the matching points, and for each point in each image calculate the corresponding epipolar line in the other image. You can draw the epipolar lines using `draw_epiline()` from the supplementary material. According to epipolar geometry, the corresponding epipolar line should pass through the point. As a testing reference, the correct fundamental matrix for images `house.007.png` and `house.008.png` is included in the supplementary material in file `F_7_8.txt`.

(c) We can use the *reprojection error* as a quantitative measure of the quality of the estimated fundamental matrix. Write a function `reprojection_error()` that calculates the reprojection error of a fundamental matrix F given two matching points. For each point, the function should calculate the corresponding epipolar line from each point's match in the other image, then calculate the perpendicular distance between the point and the corresponding line using the equation:

$$distance(ax + by + c = 0, (x_0, y_0)) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}, \tag{9}$$

where a, b and c are the parameters of the epipolar line. Finally, the function should return the average of the two distances.

Write a script that performs two tests:

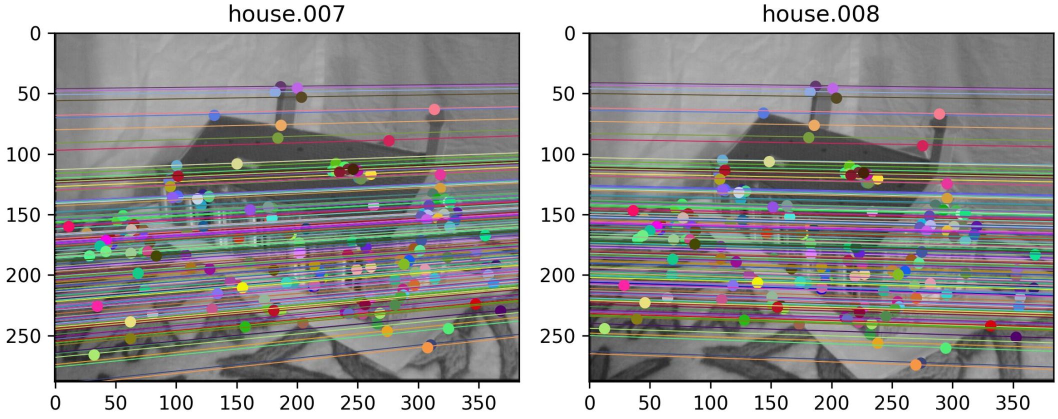


Figure 3: Correspondences and epipolar lines for both images, calculated with the fundamental matrix in `F_7_8.txt`.

(a) compute the reprojection error for points $p_1 = [160, 463]^T$ in the left image-plane and $p_2 = [128, 437]^T$ in the right image-plane using the fundamental matrix (the error should be approximately 0.52 pixels).

(b) Load the points for images `house.007.png` and `house.008.png`, calculate F and compute the average of symmetric reprojection errors for all pairs of points. If your calculation is correct, the average error should be approximately 0.41 pixels.

(d) ★ (15 points) Perform fully automatic fundamental matrix estimation on a pair of images from the directory `desk`[4]. Detect the correspondence points using your preferred method. As some matches might be incorrect, you will have to modify the RANSAC algorithm so that it works for fundamental matrix estimation. You can measure the quality of a solution by using the point-to-line reprojection error. Display the correspondences to check whether all of them are correct. Calculate the fundamental matrix on the final set of inliers and display the keypoints with the corresponding epipolar lines. If the epipolar lines converge inside the image, your fundamental matrix estimation is not correct. This can happen in a degenerate case when all the inliers lie on the same plane, thus the fundamental matrix cannot be correctly estimated (even if the point-line distance is sufficiently small).

(e) ★ (15 points) Improve the search for a good model by implementing PROSAC [1]. You can find the full paper here and a simplified explanation here. The main idea is that some matches are more likely to be inliers and should be sampled more often than others. This includes calculating a match score (e.g. Lowe's ratio test or something similar), sorting the matches, then focusing the random sampling on the higher-scored matches. The range of sampling is then increased over time, eventually working identically to standard RANSAC. You should show that on the same data, PROSAC generally returns models with a larger percent of inliers compared to

[4] Credit: http://nghiaho.com/

RANSAC in the same number of iterations (store and load the correspondences using `np.save()` and `np.load()`).

Note: The images and data for the dataset `house` are taken from the Multi-view datasets created by Visual Geometry Group at Oxford. You can download and use other sequences for developing your solutions, if you wish.

Exercise 3: Triangulation

If we know the intrinsic parameters of the cameras as well as their relative poses, we can calculate the 3D position of the points observed in both cameras. You can find the projection matrices for both cameras stored in files `house1_camera.txt` and `house2_camera.txt`. Note that these are projection matrices and thus include both intrinsic and extrinsic parameters and are of size 3 × 4.

We will use an algebraic approach to perform the triangulation. Assuming we have a 2D correspondence between $\mathbf{x}_1$ in the first image plane and $\mathbf{x}_2$ in the second image plane (in homogeneous coordinates), a location of the common point $\mathbf{X}$ in 3D space (4D in homogenous coordinates) is given by relations

$$\begin{aligned} \lambda_1 \mathbf{x}_1 &= \mathbf{P}_1 \mathbf{X} && (10) \\ \lambda_2 \mathbf{x}_2 &= \mathbf{P}_2 \mathbf{X}. && (11) \end{aligned}$$

where $\lambda_1, \lambda_2 \in \mathbb{R}$. We know that a vector product between parallel vectors is 0 so we use $\mathbf{x}_1 \times \lambda_1 \mathbf{x}_1 = 0$ and get:

$$\begin{aligned} \mathbf{x}_1 \times \mathbf{P}_1 \mathbf{X} = [\mathbf{x}_{1\times}] \mathbf{P}_1 \mathbf{X} = 0 && (12) \\ \mathbf{x}_2 \times \mathbf{P}_2 \mathbf{X} = [\mathbf{x}_{2\times}] \mathbf{P}_2 \mathbf{X} = 0, && (13) \end{aligned}$$

where we have used the following form (shear-symmetric form) to get rid of a vector product:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{x}_{1\times}] \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \mathbf{b}. \qquad (14)$$

For each pair of 2D points we get two independent linear equations for three unknown variables. If we combine in a matrix $\mathbf{A}$ the first two lines of the product $[\mathbf{x}_{1\times}]\mathbf{P}_1$ and first two lines of the product $[\mathbf{x}_{2\times}]\mathbf{P}_2$, we can compute the mean quadratic estimate of $\mathbf{X}$ by solving a linear equation system $\mathbf{A}\mathbf{X} = 0$. As you already know by now, such a solution can be obtained by computing the eigenvector of matrix $\mathbf{A}$ that has the lowest eigenvalue. Note that the solution of the system is a point $\mathbf{X}$ in homogeneous coordinates (4D space), therefore you have to first normalize the values so the last coordinate becomes 1.

(a) Implement the function `triangulate()` that accepts a set of correspondence points and a pair of projection matrices as an input and returns the triangulated 3D points. Test the triangulation on the matching keypoints from the directory `house`, and the corresponding projection matrices P located in subdirectory `3D` (again, load them using `np.loadtxt()`). Visualize the result using `plt.plot()` or `plt.scatter()` and draw the corresponding points with the same color. Also plot the index of the point in 3D space (use `plt.text()`) so the results will be easier to interpret. Plot the points interactively, so that you can rotate the visualization.

Note: The coordinate system used for plotting in 3D space is usually not the same as the camera coordinate system. In order to make the results easier to interpret, the ordering of the axes can be modified by using a transformation matrix. For example, you may want to use matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{15}$$

to transform 3D points with $\mathbf{Tx}$.

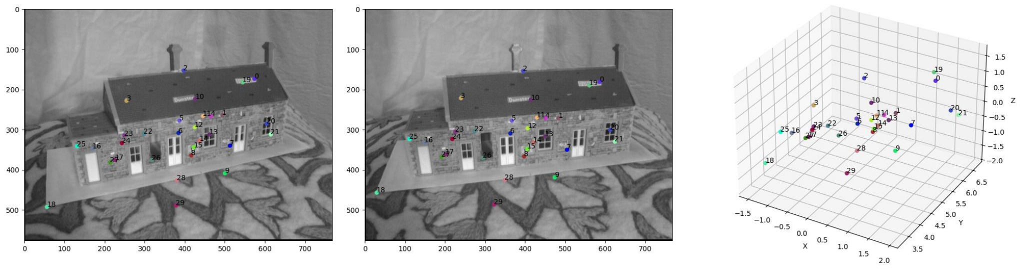


Figure 4: Points on the first and second image, and their 3D positions visualized in a plot. To get a feeling of their placement, rotate the plot.

(b) ★ (10 points) Detect and match your own points on two of the images in the `house` dataset. Estimate the fundamental matrix F using RANSAC, and perform the triangulation using the included projection matrices. Try to get as good results as possible. Under what circumstances does the triangulation perform poorly?

References

[1] Ondrej Chum and Jiri Matas. Matching with prosac-progressive sample consensus. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 220–226. IEEE, 2005.

[2] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall, 2002.