Exercise 5: Epipolar geometry and triangulation

Machine perception

2018/2019

Create a folder `exercise5` that you will use during this exercise. Unpack the content of the `exercise5.zip` that you can download from the course webpage to the folder. Save the solutions for the assignments as the *Matlab/Octave* scripts to `exercise5` folder. In order to complete the exercise you have to present these files to the teaching assistant. Some assignments contain questions that require sketching, writing or manual calculation. Write these answers down and bring them to the presentation as well. The tasks that are marked with ★ are optional. Without completing them you can get at most 75 points for the exercise (the total number of points is 100 and results in grade 10). Each optional exercise has the amout of additional points written next to it.

Introduction

In this exercise we will review several parts of the epipolar geometry [1] (chapter 10.1) and robust estimation [1] (page 346).

Assignment 1: Disparity

In this assignment we 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) lay in the same plane (Image 1a).

In Image 1(b) we can observe that the following relations using similar triangles can be written:

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

(a) Using equations (1) derive the expression for *disparity*, that is defined as $d = x_1 - x_2$. What is the relation between the distance of the object (point $\mathbf{p}$ in Image 1) to 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) To answer the question with more confidence, write a *Matlab/Octave*script that computes a disparity for a range of values of p_z. Plot the values and set the axes to appropriate units. Use the following parameters of the system: focal length is $f = 2.5mm$ camera baseline is $T = 12cm$.

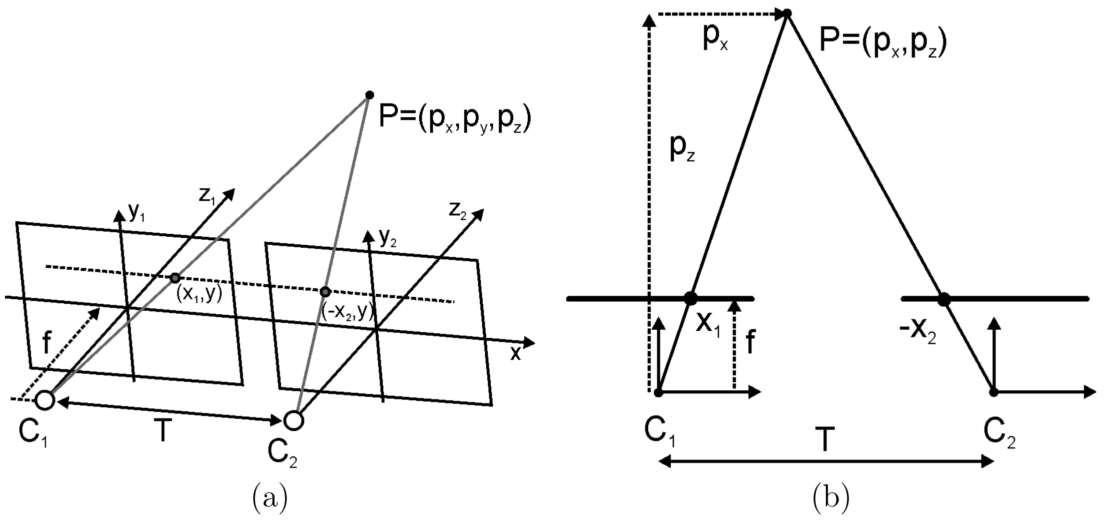


Figure 1: Left image shows the simplest stereo setup with two parallel cameras. Right image shows geometric relations when mapping of a point $\mathbf{p}$ in 3D space to axis x in image planes of the cameras.

(c) To get a better idea about how the distance to the object influences the disparity in terms of *pixels* of the sensor in a camera, lets look at a real case, where we will take parameters from a specification of a commercial stereo camera *Bumblebee2* manufactured by company PointGray: $f = 2.5mm$, $T = 12cm$, image sensor has a resolution of 648×488 pixels that are square and the width of an 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. Lets say that we use this system to observe an (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) ★ (5 points) To test the disparity in practice use the image pairs in the `disparity` and compute a dense depth field using the theory explained in the previous tasks. The idea is to search for the best match for a certain point from the left image in the right image. To do this we have to observe the neighborhood of a point and find the most similar neighborhood in the second image. For pre-processed images[1] we can limit the search of a match to the same row in the other image. A good way of matching a patch is to use normalized cross-correlation. Normalized cross-correlation for matrices X and Y of equal size is defined as

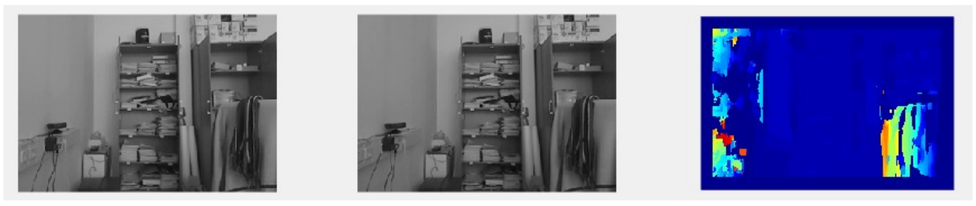
$$NCC(\mathbf{X}, \mathbf{Y}) = \frac{1}{N} \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(\sqrt{1/N \sum(x_i - \bar{x})^2})(\sqrt{1/N \sum(y_i - \bar{y})^2})}. \tag{2}$$

where x_i denotes a specific cell in matrix $\mathbf{X}$ and y_i a specific cell for matrix $\mathbf{Y}$. A match is considered as the patch in the corresponding row in the other image with the highest normalized cross-correlation value. The difference in x axis for the original point in the first image and the center of the match in the second image

[1]Raw captured images have to be first aligned and un-distorted if we want to use this assumption.

can be considered as a disparity estimate in terms of pixels[2]. For a single pixel a search can be therefore be done only in the corresponding row in the other image in a single direction since the disparity is usually positive. It is also common that the search is limited to some maximum value.

Write a function `disparity` that takes two matrices that represent a grayscale image pair, a size of a patch to be used and the maximum possible disparity to limit the search. The function returns a disparity field (a matrix that contains disparity values for each pixel). Write a script that calculates a disparity field for a given image pair and visualizes it using `imagesc`. Also experiment with the parameter values to set them optimally. Is the disparity estimated well for all pixels? If not, what are the reasons? Note: in this implementation explicit loops cannot be avoided (at least not in any straightforward way) so the implementation will not be very efficient. If the algorithm takes too much time, resize the input images to half size using `imresize(A, 0.5)`.



(e) ★ (10 points) Improve the noisy disparity estimation function form the previous task. This can be done by employing simple concepts that you have used in the previous exercises, for example implement a symmetric matching technique where the left image is compared to the right image and the right image is compared to the left image and the results are then merged in some clever way that you propose (note that the disparity estimates cannot be merged element-wise as they are acquired using different images as origin image). Smooth the disparity estimate using median filter and set the size of the filter window for optimal result.

More ambitious students can alternatively implement a state-of-the-art disparity refinement technique, described in [2] (special rewards may apply in this case). The paper is available in the supplementary material.

Assignment 2: Fundamental matrix, epipoles, epipolar lines

In the previous assignment we have considered a special case of a camera setup, where the image planes of two cameras are aligned. We observed that in this case a 3D point has the same y coordinate when projected on both sensors and that the difference in the projection along the x axis tells us the distance of the point to the camera. This kind of setup assures that the *epipolar lines in the cameras are parallel to the lines in the sensor*. This greatly simplifies the search for correspondences (the projections of the same 3D point to both sensors). In general these epipolar lines are not aligned with rows. In

[2]Converting this estimate to distance requires information about the camera, which is unavailable for the given image pairs.

order to establish a general connection between two image planes we have to compute a *fundamental matrix*.

In this assignment you will implement a simple version of a *eight-point algorithm* [1] that is used to estimate a fundamental matrix between two cameras and visualize the epipolar lines. For better understanding of the assignment we will first revisit the parts of the theory that you will need for the task. Lets assume that 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[3]. The fundamental matrix rule states that each correspondence be a valid solution for 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 that 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

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

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

$$\begin{array}{ccc} & \mathbf{Af} = \mathbf{0} & \\ \begin{bmatrix} u_1u_1' & u_1v_1' & u_1 & v_1u_1' & v_1v_1' & v_1 & u_1' & v_1' & 1 \\ u_2u_2' & u_2v_2' & u_2 & v_2u_2' & v_2v_2' & v_2 & u_2' & v_2' & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_Nu_N' & u_Nv_N' & u_N & v_Nu_N' & v_Nv_N' & v_N & u_N' & v_N' & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ \vdots \\ F_{33} \end{bmatrix} & = & \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \end{array} \tag{5}$$

We can solve the linear system above in a least-squares way by using a singular value decomposition method (SVD). Matrix $\mathbf{A}$ is decomposed to $\mathbf{A} = \mathbf{UDV}^T$[4]. The solution according to least-squares corresponds to the eigen-vector $\mathbf{v}_n$ with the lowers eigen-value, e.t. the last column of the matrix[5] $\mathbf{V}$.

Recall that the **epipole** of a camera is a point where all the epipolar (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

[3]See previous exercise for more information on homogeneous coordinates.

[4]Note that the SVD algorithm returns matrix V that is not transposed.

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

be done by performing a SVD decomposition to $\mathbf{F} = \mathbf{UDV}^T$, set the lowest eigen-value (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 limitation rank = 2 and can be used to compute two epipoles (one for each camera)

$$\mathbf{F}\mathbf{e}_1 = 0 \quad \text{in} \quad \mathbf{F}^T\mathbf{e}_2 = 0, \tag{6}$$

by decomposing the matrix $\mathbf{F}$ again and compute the *left* and *right*[6] eigen-vector of the matrix $\mathbf{F}$,

$$\mathbf{e}_1 = \begin{bmatrix} V_{13} & V_{23} & V_{33} \end{bmatrix} / V_{33} \text{ , } \mathbf{e}_1 = \begin{bmatrix} U_{13} & U_{23} & U_{33} \end{bmatrix} / U_{33}, \tag{7}$$

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 a matrix $\mathbf{A}$ as in equation (5), decompose the matrix using SVD $\mathbf{A} = \mathbf{UDV}^T$, and transform the last eigen-vector
  $\mathbf{v}_9 = \mathbf{V}(:, 9)$ in a 3×3 fundametal matrix $\mathbf{F}_t$[7].
- Decompose $\mathbf{F}_t = \mathbf{UDV}^T$ and set the lowest eigen-value to 0, reconstruct $\mathbf{F} = \mathbf{UDV}^T$.
- Decompose $\mathbf{F}_t = \mathbf{UDV}^T$ again, compute both epipoles as `e1 = V(:, 3) / V(3, 3)` and `e2 = U(:,3)/U(3,3)`.

Once we have a fundamental matrix, we can take any point $\mathbf{x}_1$ in the first image plane and determine an epipolar line for that point in the second image plane $\mathbf{l}_2 = \mathbf{F}\mathbf{x}_1$. Likewise, we can take point $\mathbf{x}_2$ in the second image plane and find an epipolar line in the first image plane $\mathbf{l}_1 = \mathbf{F}^T\mathbf{x}_2$. Recall that a line in a projective geometry is defined as a single 3D vector $\mathbf{l}_1 = [a, b, c]$. A line can be written using a classical Euclidean formula as

$$ax + by + c = 0, \tag{8}$$

and used to insert the coordinates $\mathbf{x}_1 = [X, Y, W]$. This gives us

$$\begin{aligned} aX + bY + cW &= 0 \quad (9) \\ \mathbf{l}_1^T\mathbf{x}_1 = \mathbf{x}_1^T\mathbf{l}_1 &= 0. \quad (10) \end{aligned}$$

The interpretation of the parameters is as follows: $-a/b$ is the angle of the line, $-c/a$ and $-c/b$ are the intersections with axes x and y.

(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. Use the reference fundamental matrix `house_fundamental.txt`.

Question: Compute the Euclidean equation of the epipolar line in the right camera that corresponds to the point at row = 120 and column = 300 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 row = 170 and col = 300 in the left camera.

Question: Compute the intersection of these two lines. What is the name of the point of intersection?

[6]Attention: the terms left and right eigen-vector are mathematical terms and do not hold any relation to the left and right camera in our system.

[7]Using `reshape(v9, 3, 3)'` – note the transpose operation, as the function `reshape` orders the elements by columns rather than rows.

Estimating a fundamental matrix

In this sub-assignment you will implement and test the basic fundamental matrix estimation.

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

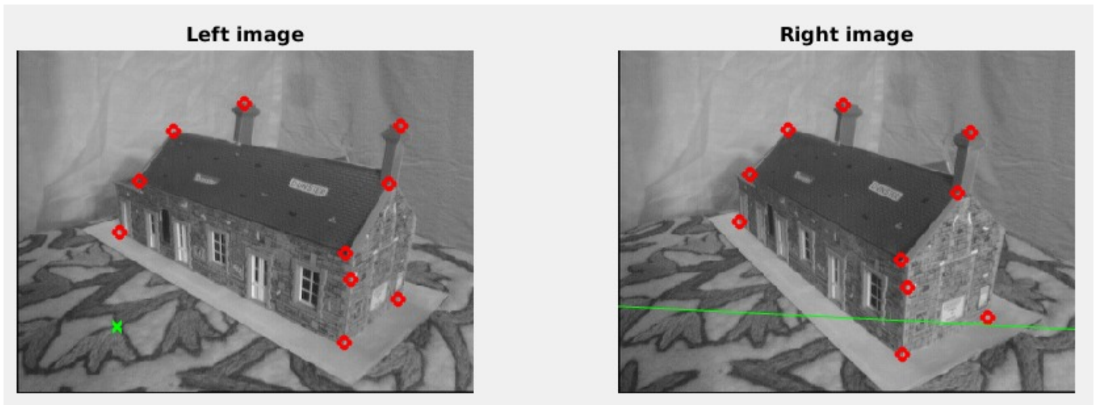
```
function [F, e1, e2] = fundamental_matrix(x1, x2)
% Input: x1, x2 : 3xN N homogeneous points in 2D
% Output:
% F : 3x3 fundamental matrix: x2' * F * x1 = 0
% e1 : epipole in the first image: F * e1 = 0
% e2 : epipole in the second image: F' * e2 = 0
...
```

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 $\sqrt{2}$. For a set of points $\mathbf{x}_1$ we achieve this by multiply them with a transformation matrix $\mathbf{T}_1$. Study the function below that can be used to perform point-set normalization.

```
function [NP, T] = normalize_points(P)
P = P(1:2,:);
mu = mean(P,2);
scale = sqrt(2) / mean(sqrt(sum((P - repmat(mu,1,size(P,2))) .^ 2, 1)));
T = [scale, 0, -mu(1) * scale; 0, scale, -mu(2) * scale; 0, 0, 1] ;
NP = zeros(3,size(P, 2));
for i = 1 : size(P, 2)
  NP(:,i) = T * [P(:,i);1];
end
```

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 fundamental matrix $\hat{\mathbf{F}}$, then transforms it into the original space using both transformation matrices $\mathbf{F} = \mathbf{T}_2^T \hat{\mathbf{F}} \mathbf{T}_1$ and finally computes the epipoles using the equation (7).

(b) Test your function for fundamental matrix estimation using the ten correspondence pairs that you load from the file `house_points.txt`. Compute the fundamental matrix $\mathbf{F}$ and for point $p = [85, 233]^T$ in the left image-plane compute the epipolar line in the right image-plane. A line can be plotted using the `draw_line` function that you can find in the supplementary material. According to epipolar geometry the corresponding point of the given point should lay on its epipolar line. The correspondence point for the point p is located at coordinates $[67, 219]^T$. As a testing reference a correct fundamental matrix is attached in the supplementary material in file `house_fundamental.txt`.



(c) We use the *re-projection error* as a quantitative measure of the quality of the estimated fundamental matrix. Use the function `reprojection_error` from the supplementary material that accepts three parameters, the first two being the correspondence points in the first and the second camera and the last being the fundamental matrix. The function returns the *symmetric* re-projection error, i.e. it computes the re-projection error for re-projection of x_1 according to x_2 as well as a re-projection error of x_2 according to x_1 and return an average of both distances.

Write a script that should perform two tests: (i) compute the re-projection error for points $p_1 = [85, 233]^T$ in the left image-plane and $p_2 = [67, 219]^T$ in right image-plane using the fundamental matrix (the error should be approximately 0.15 pixels). (ii) Load the points from the file `house_points.txt` and compute an average of symmetric re-projection errors for all pairs of points. If your code works properly, the average error should be approximately 0.33 pixels.

RANSAC algorithm

In practice automatic correspondence detection algorithms never find perfect matches. In most cases the locations of these points contain some noise, however, in some cases several correspondences are also completely wrong. These correspondences are called *outliers*[8]. As you have heard at lectures, a good meta-algorithm for such cases is RANdom SAmple Consensus (RANSAC). The algorithm can be applied to a wide variety of problems. A version that can be used for robust estimation of fundamental matrix is structured as follows:

- Randomly select a minimal set of point matches (correspondences) that are required to estimate a model (in case of fundamental matrix this number is 8).
- Estimate a fundamental matrix using the selected sub-set.
- Determine the *inliers* for the estimated fundamental matrix (i.e. matched pairs that have a re-projection error lower than a given threshold).
- If the percentage of inliers is big enough ($\geq m$) use the entire inlier subset to estimate a new fundamental matrix (using mean squares method and SVD).
- Otherwise repeat the entire procedure for k iterations.

[8]You have probably noticed some outliers in the previous exercise when computing a homography.

The value of parameter k is defined by the properties of our data that are set by knowing the estimation problem that we are trying to solve. E.g., suppose that we constantly encounter at least w percent of inliers (correctly matched point pairs). The number of pairs required to estimate a fundamental matrix $\mathbf{F}$ is at least $n = 8$. We can now compute the probability of successful estimation of $\mathbf{F}$, i.e. the probability that all n selected points will be inliers as w^n. The probability that this will not be true is $1 - w^n$. The probability that we do not manage to select a clean set of inliers in k repetitions is $p_{\text{fail}} = (1 - w^n)^k$. In practice we therefore select k high enough to reduce the probability of failure p_{fail} to an acceptable level.

You will now implement all the required steps of the RANSAC algorithm.

(d) Implement function `get_inliers`, that accepts an estimation of a fundamental matrix, a complete set of correspondences and a threshold ε as an input and returns a sub-set of correspondences that are considered inliers for the given fundamental matrix. A point pair, $\mathbf{x}$ and $\mathbf{x}'$, are considered inliers if the distance between $\mathbf{x}$ and the epipolar line $\mathbf{F}^T\mathbf{x}'$ (as well as the other way around) is lower than ε.

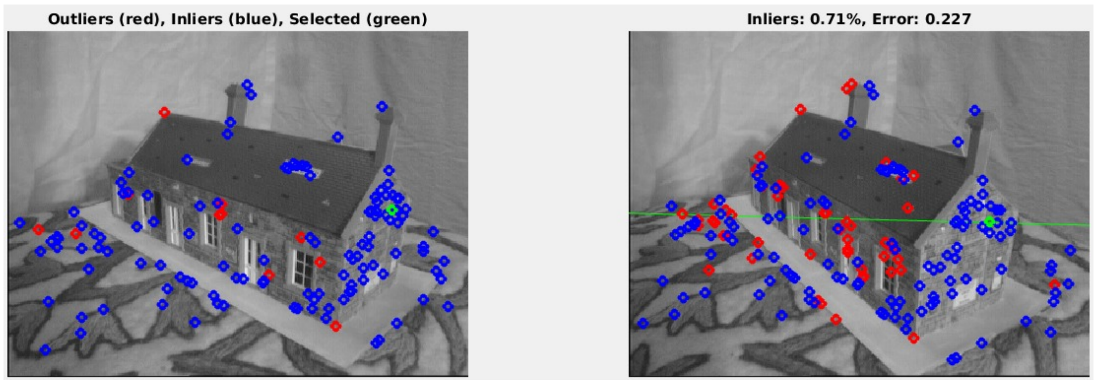
```
function [x1in x2in] = get_inliers(F, x1, x2, eps)
 ...
```

(e) Implement function `ransac_fundamental`, that executes the variation of the the RANSAC algorithm that robustly estimates a fundamental matrix using a normalized eight-point algorithm. To get a random set of points you can use function `randperm` that returns a random permutation of indices[9]. Implement a simple version of RANSAC algorithm that repeats k iterations and then returns the solution that is supported by the largest number of inliers.

```
function [F, e1, e2, x1, x2] = ransac_fundamental(x1, x2, eps, k)
 % Input:
 % x1, x2 : 3xN matrix of N homogeneous points v 2D space
 % eps : threshold for inliers
 % k : number of iterations
 % Output:
 % F : 3x3 fundamental matrix: x2' * F * x1 = 0
 % e1 : epipole in image 1: F * e1 = 0
 % e2 : epipole in image 2: F' * e2 = 0
 % x1, x2 : 3xNi matrix of Ni homogeneous inlier points
 ...
```

Apply your version of RANSAC algorithm with parameters $\varepsilon = 5$ and $k = 100$ to a set of correspondences that you read from the file `house_matches.txt`. In order to visually inspect the success of the estimation of $\mathbf{F}$ choose a point in the left image-plane and display its epipolar line in the right image-plane (using `draw_line`). Make sure that the line passes the correspondence point in the right image-plane. You can also do that for multiple points. Set the parameters ε and k for optimal results. In a single figure plot all potential correspondence pairs from `house_matches.txt` as well as correspondence pairs that were chosen by the RANSAC algorithm as inliers (using a difference color). What is the difference between the sets? Note that the examples below may be a bit different to your own results as the RANSAC algorithm is stochastic and therefore produces different result each time.

[9]If you want to get a random ten indices between 1 and 100, then use `a = randperm(100, 10)`.


Outliers (red), Inliers (blue), Selected (green)
Inliers: 0.71%, Error: 0.227

(f) ★ (10 points) Now we can fully automate the process of fundamental matrix estimation. Detect the correspondence points using the function `find_matches` that you have implemented for the previous exercise. Using the matches that you obtain run the RANSAC-based estimation of fundamental matrix. As a result of this task display the left image with all the detected points and the same image with the points that are kept by the RANSAC algorithm. What is the percent of the correspondence pairs that were used by RANSAC and what is the re-projection error? Set the parameters of all the parts of the system for best results.

(g) ★ (10 points) Implement another version of RANSAC algorithm, this time for estimation of homography that you have studied in the previous exercise. Apply the algorithm to the image alignment tasks, especially the panorama creation task at the end of the exercise. Compare the result with the mean-square-error minimization approach.

Assignment 3: Triangulation

For the last assignment in this exercise we will reconstruct the correspondences back to 3D space. This if of course impossible without prior calibration of the system. You can find the calibration matrices for both cameras stored in files `house1_camera.txt` and `house2_camera.txt`. A calibration matrix has the size 3×4.

We will use a linear algebraic approach to do triangulation. Assuming that 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 is given by relations

$$\begin{aligned} \lambda_1 \mathbf{x}_1 &= \mathbf{P}_1 \mathbf{X} && (11) \\ \lambda_2 \mathbf{x}_2 &= \mathbf{P}_2 \mathbf{X}. && (12) \end{aligned}$$

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:

$$\mathbf{x}_1 \times \mathbf{P}_1 \mathbf{X} = [\mathbf{x}_{1\times}] \mathbf{P}_1 \mathbf{X} = 0 \quad (13)$$
$$\mathbf{x}_2 \times \mathbf{P}_2 \mathbf{X} = [\mathbf{x}_{2\times}] \mathbf{P}_2 \mathbf{X} = 0, \quad (14)$$

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

$$\mathbf{a} \times \mathbf{b} = [\mathbf{x}_{1\times}]\mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \mathbf{b}. \quad (15)$$

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 $\mathbf{AX} = 0$. As you already know by now, such a solution can be obtained by computing the eigen-vector of matrix $\mathbf{A}$ that has the lowest eigen-value. Note again that the solution of the system is a point $\mathbf{X}$ in homogeneous coordinates (4D space), therefore you have to first normalize the values in a way that the last coordinate becomes 1. To change a vector to a shear-symmetric form you can use a helper function `cross_form` below.

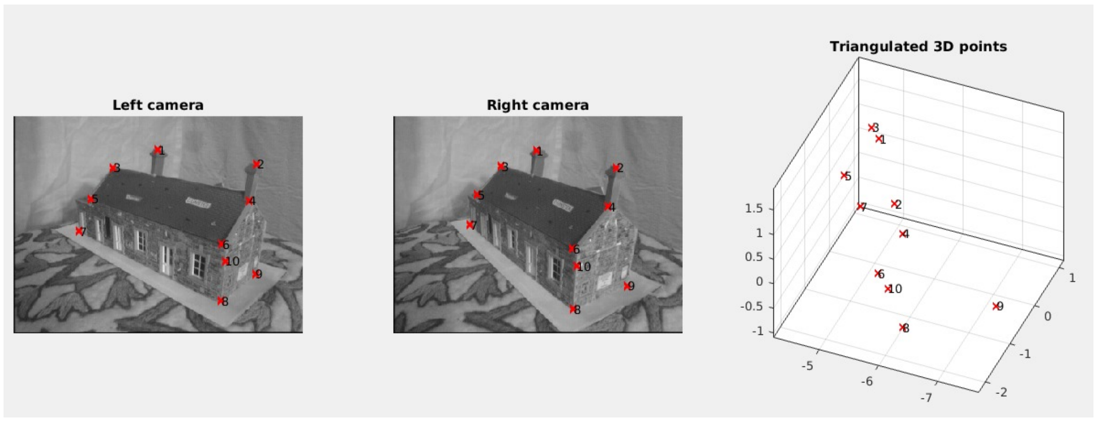
```
function Cx = cross_form(a)
Cx = [ 0, -a(3), a(2); a(3), 0, -a(1); -a(2), a(1), 0];
```

(a) Implement a function that accepts a set of correspondence points and a pair of calibration matrices as an input and returns the triangulated 3D points. You can use the stub-function `triangulate` below as a starting point. Visualize the result using function `show_triangulation` that you can find in the supplementary material. Test the triangulation on the ten points from the file `house_points.txt`. Plot these points over the left and right image and then use `plot3` function to also plot their triangulated position on a separate plot. Due to calibration matrix setup, you should change the order of dimensions to get a better representation of the 3D information (the plot on the example below is adjusted this way).

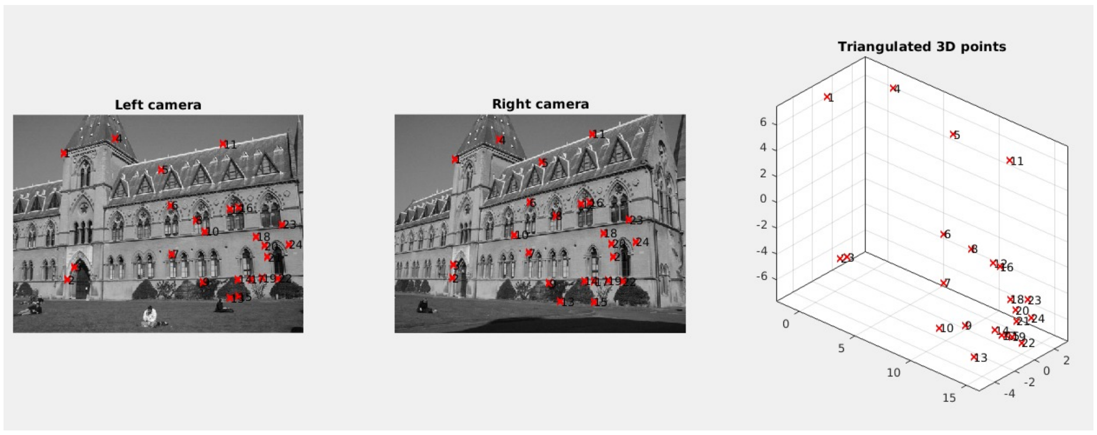
```
function X = triangulate(pts_1, pts_2, P_1, P_2)
% Input:
% pts_1 ... 3xN left camera points in homogeneous coordinates
% pts_2 ... 3xN right camera points in homogeneous coordinates
% P_1 ... 3x4 calibration matrix of the left camera
% P_2 ... 3x4 calibration matrix of the right camera
% Output:
% X ... 4XN vector of 3D points in homogeneous coordinates
 X = [] ;
 for i = 1 : size(pts_1,2)
    a1 = cross_form(pts_1(:,i)) ;
    a2 = cross_form(pts_2(:,i)) ;
    c1 = a1*P_1 ;
    c2 = a2*P_2 ;
    A = [ c1(1:2,:); c2(1:2,:) ] ;

    % TODO: perform SVD decomposition of matrix A

    % store the solution in vector X
    X = [X, x] ;
end
```



(b) ★ (15 points) Now you have everything ready for an automatic sparse detection of 3D points at an object in the observed scene. Detect all the matching points and use RANSAC to remove outliers. Triangulate the remaining points by taking account the calibration matrices and display the result as a 3D plot. Set the parameters to get as good results as possible. Test your algorithm also on the `library1.jpg` and `library2.jpg` pair of images from the supplementary material.



References

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

[2] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. In *IEEE International Conference on Computer Vision*, pages 49–56, 2013.