MAKE SURE THAT YOUR REPORT IS ANONYMOUS

Exercise 1: Virtual rotation of a camera

We consider an application to measure the size of a foot. The foot is placed on a flat piece of A4 paper, the size of which is 210 x 297 mm. See Figure 1. The set-up of the system is shown in Figure 2. Here the camera coordinate system and the image plane are presented together with the A4 paper.

Figure 1 Image of a foot

If a flat object is positioned on the A4 paper, its size can be measured by a camera image, provided that the optical axis is orthogonal to the paper. In that case, the image would be a similarity transform[1] of the paper in which the geometry is fully preserved apart from a scaling factor. This scaling factor can be easily retrieved from the known size of the A4 paper.

A foot, which is placed on the paper, is not fully flat. Nonetheless, its occluding boundary in the image is supposed to provide sufficient information to find the size of it, albeit with some uncertainty. Actually, the most disturbing factor is the (almost) impossibility to have the optical axis pointing exactly orthogonal to the A4-paper. As a result, the similarity transform becomes a projective transform, which does not preserve the geometry: the A4 paper in the image is not a rectangle anymore. The main question that is addressed in this exercise is as follows:

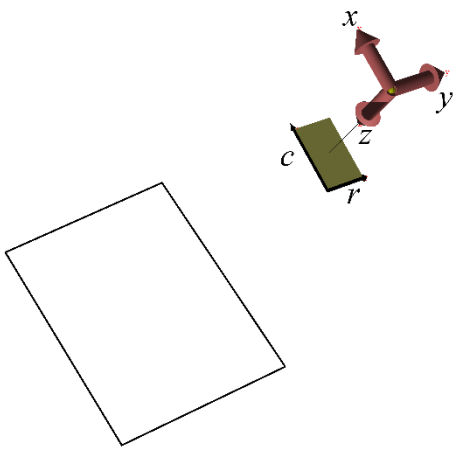


Figure 2 Camera pose relative to A4 paper

> *How to process the image in such a way that the image becomes a similarity transform of the floor on which that A4 paper is lying?*

Approach:

To achieve the similarity transform we are going to virtually rotate the camera such that its optical axis becomes orthogonal[2]. The challenge of this exercise is to accomplish this. The virtually rotation of a camera is described in Section 4.3 of the syllabus "Projective geometry and camera models". We have to know the camera calibration matrix, and a suitable selected rotation matrix. Apart from that, we may have to compensate the nonlinear lens distortion.

[1] A similarity transform is a rigid geometrical transform which only consists of isotropic scaling, rotation, and shifting.

[2] Other methods exist, but for educational reasons virtual rotation is chosen.

We only focus on the virtual rotation. Other image processing needed to make the system a full automated application, e.g., finding the boundary contours of the A4 paper and the foot, and performing the actual measurement will not be addressed.

Available data and software:

The data that will be used for the exercise are:

- An image of a foot as shown in Figure 1.
- Calibration images of a calibration chessboard.
- A matlab function `ut_plot_lens_distortion` to visualize the results of camera calibration.

Assignments

1. Camera calibration

1.1. Copy the chessboard images to your local folder. Use the camera calibration app from MATLAB. Load the images. The size of the checkers on the board is 40 mm. Inspect whether in all images the corners are well detected. If not discard the bad images. Apply the calibration. Inspect the reproduction errors. If one image shows much larger errors relative to others, discard the image and calibrate again.

The calibration can be done with 2 or 3 radial distortion parameters; with or without a tangential distortion parameter, and with or without a skew (shear) parameter. Investigate which combinations of these adjustments provides the best results. Finally, export the cameraParameters object to the work space, and save it there for later use. If you name this object `camp`, then saving is accomplished with: `save camp camp`.

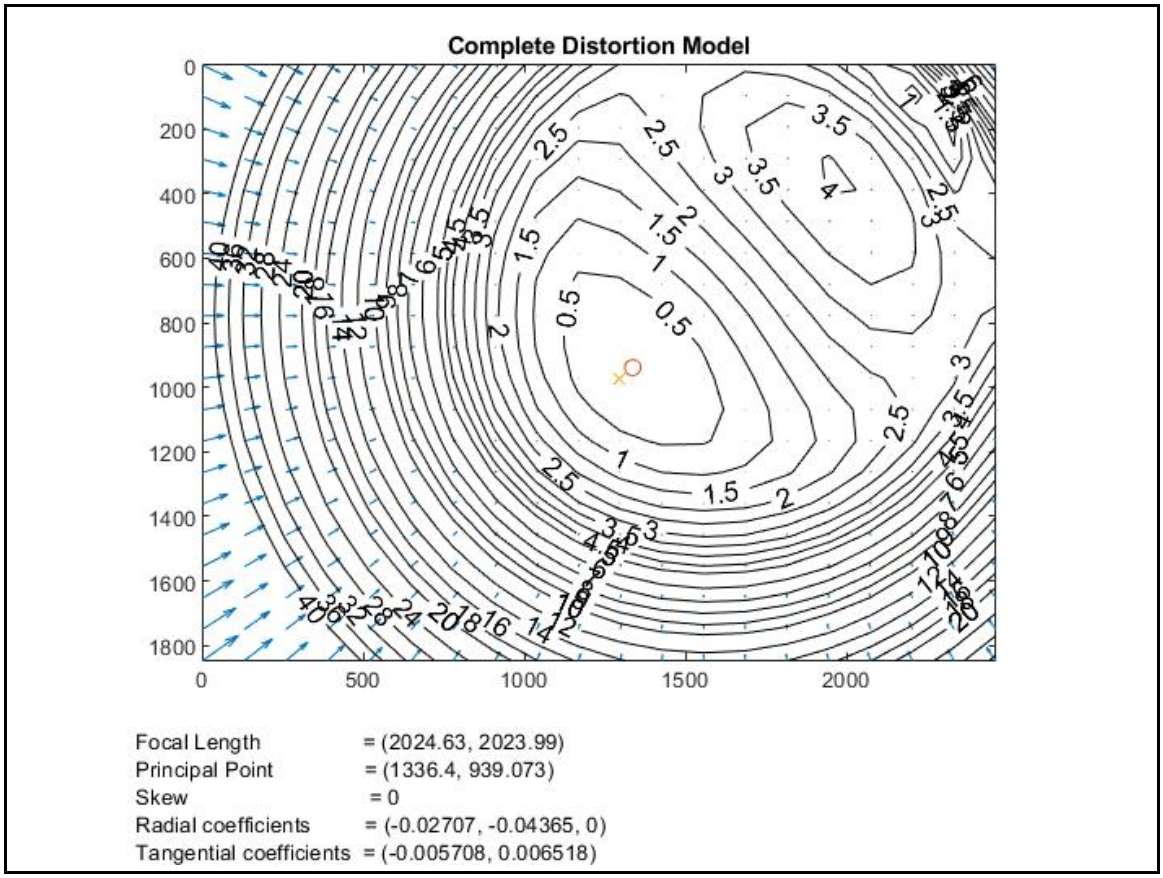
Answer the question:

Nr. of images discarded:	7
Mean pixel reprojection error	0.17
Nr of radial distortion parameters:	2
Did you include tangential distortion parameters?	Yes
Did you include the skew parameter:	No

Motivate your choice:

We choose the combination of tangential distortion and without skew parameters as it has the smallest mean pixel reprojection error.
Images that has mean error too high comparing with mean error are discarded.
We choose 2 radial distortion parameters as there is no diffferent with 3 parameters.
In the case of radial distortion, the coefficient of 2 showed a better output than 3.

1.2. The image size of the video and the calibration images is 1944x2592 (HxW). Use the function `ut_plot_lens_distortion` to create an overview of the results of calibration. Print the plot to a png file, and insert the result here:



Answer the following questions:

Give the calibration matrix: Note Matlab usually sticks to transposed versions of matrices, e.g. calibration matrices, rotation matrices. We will not do that. So, you have to correct for that.	1.0e+03 * 2.0246 0 1.3364 0 2.0240 0.9391 0 0 0.0010
The radial distortion parameters:	[-0.027074899316257,-0.043649496871693]
The tangential distortion parameters (if any)	[-0.005707927907212,0.006517939707450]
The skew parameter (if any)	0

2. Nonlinear lens deformation

2.1. To see whether compensation of nonlinear lens deformation is needed, we examine the four corner points in the foot image. Read the image from file (`imread`), show the image in a figure window (`imshow`), and then manually pinpoint the four corners in the order as indicated in Figure 2. This should be done very accurately. The functions `getpts` and `ginput` both allow this manual pinpointing, but these functions inhibit the 'Zoom in' and 'Zoom out' tools in the figure toolbar. Since zooming in to the corners is really needed for the pixel accuracy, alternatives, like `roi=rawpolygon`, must be used. To output the pinpointed coordinates, use

the `roi.Position` method of this object. Store the 4 points in an array `P`. Use the function `undistortPoints` to compensate for the lens distortion. Results in the array `Pcomp`. Finally, calculate for each point the pixel shift distance between `P` and `Pcomp`. Use the graph in Section 1.2 to check whether these distances are consistent with the graph.

	corner 1	corner 2	corner 3	corner 4
pixel shift:	6.8437	2.1804	3.2158	2.1140

Based on these results, do you think that compensation is useful? Yes

Motivate:

As we can see, the distortion of the points are quite consistence between the undistortPoints function and the graph 1.2. There is a significant in pixel shift with these corner of the points. To have straight lines connecting between the corners and used for later correct rotation, the radial distortion and tangential distortion are better to be compensated.

2.2. Apply the function `undistortImage` to the foot image to correct for the lens deformation. Use this image in the further processing. Also, in the further processing, use the compensated points `Pcomp`.

3. Perspective distortion

The A4 paper in the image is perspectively distorted. To check how serious this is, we can measure the angles at the corners in the image, which in case of a similarity transform should all be 90 degrees. Also, parallel sides of the A4 should have the same length. The ratio between the length of the shortest side and the longest side should be 210/297.

3.1. Use the array `Pcomp` to determine[3] the angles of corner 2 and corner 3.

	corner 2	corner 3
angle in degrees:	100.53	82.94

3.2. Use the function `imtool` to interactively determine the length of the sides between the corners. Use the tool 'Measure distance' in the toolbar.

	side 1-2	side 2-3	side 3-4	side 4-1
length:	940.188	651.345	1032.725	702.934

The ratio between the lengths of the shortest side and the longest side: 0.6307

[3] The angle between two 2D or 3D vectors $\mathbf{u}$ and $\mathbf{v}$ can be calculated using the relation $(\mathbf{u},\mathbf{v}) = \|\mathbf{u}\|\|\mathbf{v}\|\cos\varphi$. See Section A1.2 in the syllabus "Math for Computer Vision and Navigation".

4. The effect of camera rotations on the perspective distortion

To get a similarity transform, we will virtually rotate the camera. The needed rotation will be represented by Euler angles. The function `eul2rotm` converts the Euler angles $\{\varphi,\theta,\psi\}$ to a rotation matrix $\mathbf{R}$ according to the ZYX convention[4]. The Euler angles around these axes are referred to as 'roll', 'pitch', and 'yaw', respectively. In camera systems, the 'roll' is around the optical axis, the 'pitch' is around the horizontal x-axis, and the 'yaw' is around the vertical y-axis. This corresponds well with the common definition used in craft, except that there the vertical axis is usually defined as the z-axis, so that then 'yaw' refers to a rotation around the z-axis. See also the syllabus "Math for Computer Vision and Navigation", Section D.3.

4.1. To rotate the camera, such that its optical axis becomes orthogonal to the floor, one of the three Euler angles can be kept unaltered. Which one? Motivate.

The roll angle in Z direction is unecesary for rotation. In the case when the camera is already perpendicular to the floor, it is clear that the rotation of camara in Z axis do not have an effect in changing ratio/size of the paper and only allow us to see the paper in straight way. Therefore, with perspective distorted picture as we have, rotating camera in X axis and the Y axis are necessary to undistort the picture, while Z axis does not help us to get the correct size of the paper.

4.2. For finding the two best Euler angles, we need an error criterion[5] that quantifies how much the resulting image deviate from a similarity transform. For that purpose, we geometrically transform the corner points that are stored in `Pcomp`, that is $\underline{\mathbf{p}}_{new} = \mathbf{H}\underline{\mathbf{p}}_{comp}$ where $\mathbf{H}$ is a homography that depends on $\mathbf{R}$. See the syllabus "Camera models". Next, we define an error measure based on the transformed corner points. Possible ingredients in this error measure are:

- Angles should be 90 degrees.
- Opposite sides of the rectangle should have the same length.
- Ratio of the lengths of shortest and longest sides must be 210/297.

Describes in words (not in MATLAB notation) which error measure you choose:

From the code, I use two methods for this error function - same length vs ratio. However, I use the same length method for this exercise.
From the given Euler angles we can calculate rotation matrix R and homography matrix R. From these matrices, we can carry the transformation for the 4 corners we found from the last exercise.
The new 4 corners are calculated after transformation and the paper sizes are found from these corner. Least square error of the equal sizes are used for the error function J

4.3. To find suitable Euler angles, the error measure must be implemented in a MATLAB function. Create an m-file `error_measure.m`, the header of which is:

```
function J = error_measure(Eangles, K, Pcomp)
% ERROR_MEASURE calculates the error measure that quantifies the
% deviation from a similarity transform using the corners of the A4
%
% Eangles:      two Euler angles stored in an array
```

[4] Remember that the matrix is then given in transposed form.

[5] An error measure is non-negative number that quantifies how much a certain state deviates from a desired state. Often used measures are: 'sum of squared differences' and 'sum of absolute differences'.

```
% K:            calibration matrix
% Pcomp:        coordinates of the four corner points
% J:            error measure after applying the virtual rotation
```

Implement your error measure. That is, the function should first transform the four corner points using the calibration matrix and the two Euler angles. Next, the error measure of your choice must be implemented, returning its value in the variable **J**.

You can test your function by inserting a breakpoint and then single step through the code while inspecting the intermediate results.

5. Finding the rotation

Suppose that the error measure is mathematically represented by $J(\mathbf{a})$ in which $\mathbf{a}$ is a 2D vector containing the two Euler angles, then we are looking for $\mathbf{a}$ such that $J(\mathbf{a})$ is minimal. Mathematically, the solution is given by:

$$\mathbf{a}_{optimum} = \arg\min_{\mathbf{a}} J(\mathbf{a})$$

5.1. In MATLAB, this is accomplished with the function `fminsearch`. To implement this, our function `error_measure` must be passed as an argument to `fminsearch`. Since the function `error_measure` has more input arguments then `Eangles` alone, we need an auxiliary so-called anonymous function to enable multiple arguments. The full code is as follows:

```
options = optimset('Display','iter',...
    'MaxFunEvals',10000,'MaxIter',5000);    % set options
EAstart = [0;0];                            % start values
foo = @(Eangles)error_measure(Eangles,K,P); % create anonymous function
[EA,J] = fminsearch(foo,EAstart,options);   % minimize the error measure
```

Implement and run this code.

EA(1) in degrees: -12.8860 EA(2) in degrees: 14.5291 J: 1.83125e-04

6. Application to image

6.1. The optimal Euler angles define the homography $\mathbf{H}$. Calculate this matrix.

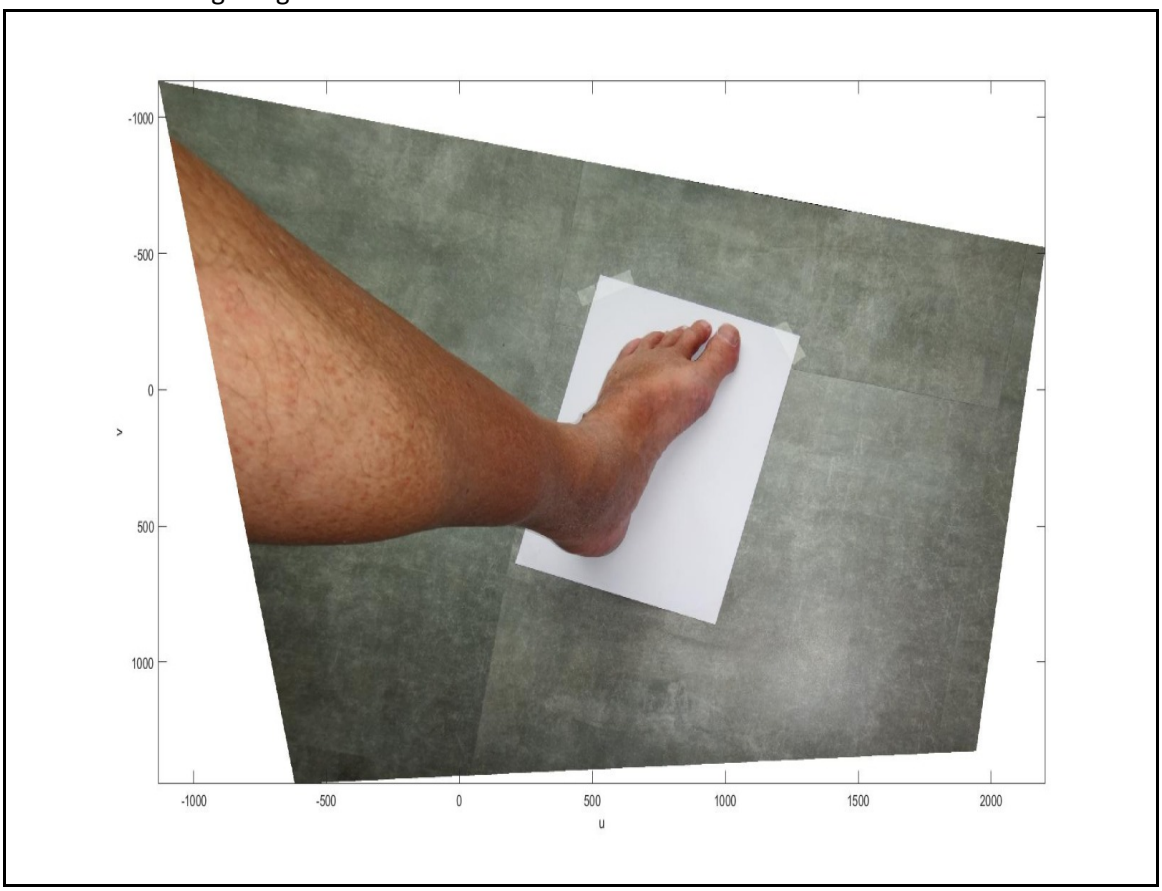
Give the homography in non-transposed form. That is, such that $\underline{\mathbf{p}}_{new} = \mathbf{H}\underline{\mathbf{p}}$ is valid, and not $\underline{\mathbf{p}}_{new} = \underline{\mathbf{p}}\mathbf{H}$:

```
   1.1220    0.1055 -774.5386
   0.1034    1.0815 -775.4413
   0.0001    0.0001    0.6830
```

6.2. Apply the found homography to the foot image. The code to do this is as follows:

```
Tform = projective2d(H'); % Note: Matlab uses the transposed form
[im_sim,Rout]=imwarp(imun,Tform,'FillValues',[255;255;255]);
figure;
imagesc(Rout.XWorldLimits,Rout.YWorldLimits,im_sim);
xlabel('u');
ylabel('v');
```

Show the resulting image:



7. Evaluating the result

To evaluate the result, we apply the same method as in question 0. Pinpoint the corners of the A4, and use this the following question.

7.1. Use the array P to determine the angles of corner 2 and corner 3.

	corner 2	corner 3
angle in degrees:	90.1120	89.9049

7.2. Use the function `imtool` to interactively determine the length of the sides between the corners. There is a tool 'Measure distance' in the toolbar.

	side 1-2	side 2-3	side 3-4	side 4-1
length:	1.105e+03	784.0	1.10475e+03	784.36

7.3. The ratio between the lengths of the shortest side and the longest side: 0.7095

7.4. Based on the results in question 7.1, 7.2, and Figure 1, with what uncertainty (units in mm) do you think that points on the floor can be located from the image (a rough assessment of this uncertainty suffices):

0.5 mm

8. Take home messages = preparation for the oral exam

Prepare yourself for the oral exam by making a list of new concepts that you have learned in this exercise. Make sure that you know these concepts, and that you understand them. Examples of questions that may be asked during the oral are:

		level
1.	What is a principal point?	knowledge
2.	What is a calibration matrix?	knowledge
3.	What is the essence of 2D projective geometry?	understanding
4.	In which way is the pinhole model of a camera embedded in 2D projective geometry?	understanding
5.	Describe the principle-of-operation (not the mathematics behind it), of virtual rotation of a camera.	understanding
6.	Virtual rotation can be used for rectification of a perspective distortion of rectangular shapes. Describe the principle-of-operation of an alternative method.	understanding

MATLAB code of your function `error_measure`: (see next page for your MATLAB script)

```
function J = error_measure(Eangles, K, Pcomp)
% ERROR_MEASURE calculates the error measure that quantifies the
% deviation from a similarity transform using the corners of the A4
%
% Eangles: two Euler angles stored in an array
% K: calibration matrix
% Pcomp: coordinates of the four corner points
% J: error measure after applying the virtual rotation

% Step A: Calculate homography from Euler angles
R=eul2rotm([0 Eangles']); % Create matrix R
H=K'*R*inv(K'); % Making homography

% Step B: Calculate the new corner with homography
Pcomph=cart2hom(Pcomp); % change into homogeneous coordinate
Pcomp_persp_undistort = H*Pcomph'; % Undistort perspective
Pcomp_persp_undistort = hom2cart(Pcomp_persp_undistort'); % change back cart

% Step C: Calculate the errors - we used the equal lengths properties
side_12=norm(Pcomp_persp_undistort(1,:)-Pcomp_persp_undistort(2,:));
side_23=norm(Pcomp_persp_undistort(2,:)-Pcomp_persp_undistort(3,:));
side_34=norm(Pcomp_persp_undistort(3,:)-Pcomp_persp_undistort(4,:));
side_41=norm(Pcomp_persp_undistort(4,:)-Pcomp_persp_undistort(1,:));
J=norm([side_41-side_23,side_34-side_12]); % least square

% % Step C: Calculate the errors - we used the ratio properties
% side_12=norm(Pcomp_persp_undistort(1,:)-Pcomp_persp_undistort(2,:));
% side_23=norm(Pcomp_persp_undistort(2,:)-Pcomp_persp_undistort(3,:));
% side_34=norm(Pcomp_persp_undistort(3,:)-Pcomp_persp_undistort(4,:));
```

MATLAB code of your script:

```
%%%%% 3DCV Exercise 1: VIRTUAL ROTATION OF THE CAMERA %%%%%%%%%%%%%

%% Summary
% Calibration the measurement of foot images using:
% - image of foot given
% - Chessboard for camera calbiration
% - Matlab fnc ut_plot_lens_distortion for visualization of the image
clear all; close all

%% Part 1: Camera calibration
% 1.1 Using camera calibration app in Matlab to select chessboard images
% The size of chessboard is 40mm
% save cameraParams cameraParams
load cameraParams;

% 1.2 Image size: 1944x2592, visualize image
imsize=[1944,2592];
ut_plot_lens_distortion(cameraParams,imsize);

%% Part 2: Nonlinear lens distortion
% 2.1 Calculate corners without distortion
% imshow(imread('voet.jpg'));
% roi=drawpolygon;
% p=roi.Position;
% save p p
p=[7.562119377104335e+02,1.176222905623030e+03;...
    1.059961465736607e+03,2.910014809128537e+02;...
    1.704208250594654e+03,3.858322927626756e+02;...
    1.424121676781341e+03,1.380464510787172e+03];
Pcomp=undistortPoints(p,cameraParams);
corner_shift=vecnorm((Pcomp-p)');

% 2.2 Un-distort image
non_lensdistort_image=undistortImage(imread('voet.jpg'),cameraParams);
imshow(non_lensdistort_image);

%% Part 3: Perspective distortion
%3.1 Find angles 2 and 3
side_12=Pcomp(1,:)-Pcomp(2,:);
side_23=Pcomp(3,:)-Pcomp(2,:);
corner2=rad2deg(acos(dot(side_12,side_23)/(norm(side_12)*norm(side_23))));
side_23=Pcomp(2,:)-Pcomp(3,:);
side_34=Pcomp(4,:)-Pcomp(3,:);
corner3=rad2deg(acos(dot(side_23,side_34)/(norm(side_23)*norm(side_34))));

%3.2 Measure the side
%imtool(non_lensdistort_image);
%messide=[messide_12; messide_23; messide_34; messide_41];
%save messide messide
messide=[9.401886806832057e+02;6.513452812406317e+02;...
    1.032725758671771e+03;7.029338211827289e+02];
ratio=min(messide)/max(messide);
```