Assignment 2: Pinhole camera model, camera calibration, and spatial resection

In this assignment, we will work with the pinhole camera model, camera calibration and undistortion and finally spatial resection using the P3P algorithm.

1 Intrinsic and extrinsic parameters

In this task, you should work with a projection of a 3D box into an image using the pinhole camera model. The intrinsics of the camera is given as:

$$f_x = f_y = 1000\ [px], \qquad p_x = 300\ [px], \qquad p_y = 200\ [px]$$

Use the above parameters to form the intrinsic matrix, K.

The location of the camera center has the the following world coordinates, $\tilde{\mathbf{C}} = [-5 \quad 0 \quad 3.5]^T$ and the rotation with respect to world coordinate system is given as $\mathbf{R} = \mathbf{R_z(0)} \cdot \mathbf{R_y(90)} \cdot \mathbf{R_x(0)}$, where the arguments to the rotation matrices are in degrees.

You should project a 3D box onto an image using the pinhole camera model, which has the general form.

$$\mathbf{x} = \mathbf{PX} = \mathbf{K}[\mathbf{R} \quad \mathbf{t}]\mathbf{X}$$

where $\mathbf{x}$ is the homogeneous coordinates in pixels.

You can generate the 3D points for the box using `X=Box3D_v2` (from the supplied matlab function). Show the generated 2D image and also make a 3D plot where both the box is shown together with the camera position and orientation. You can use the code below for inspiration of the 3D plot.

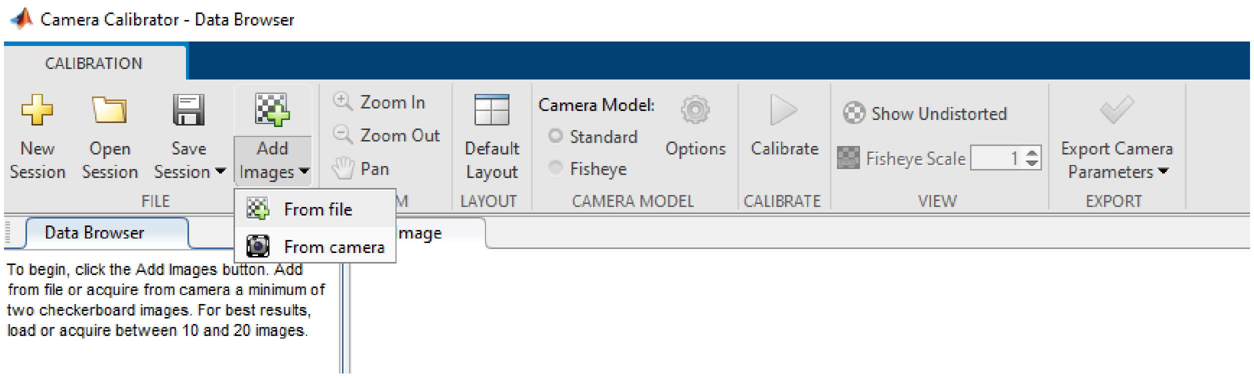
```
figure, plot3(X(1,:),X(2,:),X(3,:),'.');
hold on
cam1 = plotCamera('Location',Ctilde,'Orientation',R,...
    'Opacity',0.0,'Color',[1 0 0],'Size',0.5,'Label','Camera', 'AxesVisible',1);
hold off
axis equal;
axis([-6 6 -1 1 -1 5]);
xlabel('x');
ylabel('y');
zlabel('z');
```

2 Camera Calibration and Undistortion of Images

The first step in this task is to obtain a model of a camera and later use that model to compensate for radial and tangential distortion. We will use the 'cameraCalibrator' toolbox in MATLAB to obtain a model of the camera. This model can be generated by sourcing multiple images of a checkerboard taken from various viewpoints to the program. Once the coefficients for the radial- and tangential distortion polynomials are calculated, they will be used to implement an algorithm where we compensate for the distortion.

2.1 Estimate Radial and Tangential Distortion coefficients

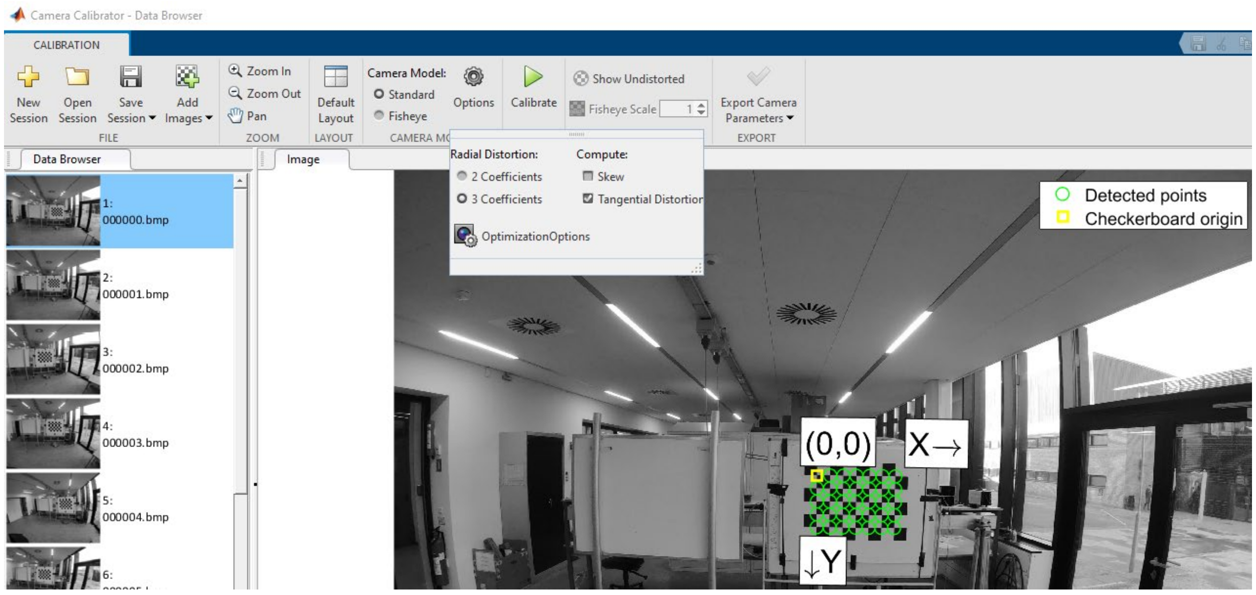
Write 'cameraCalibrator' in the command window of MATLAB. Press 'Add Images' and select all images in the calibration_images folder (images of checkerboard from various viewpoints).



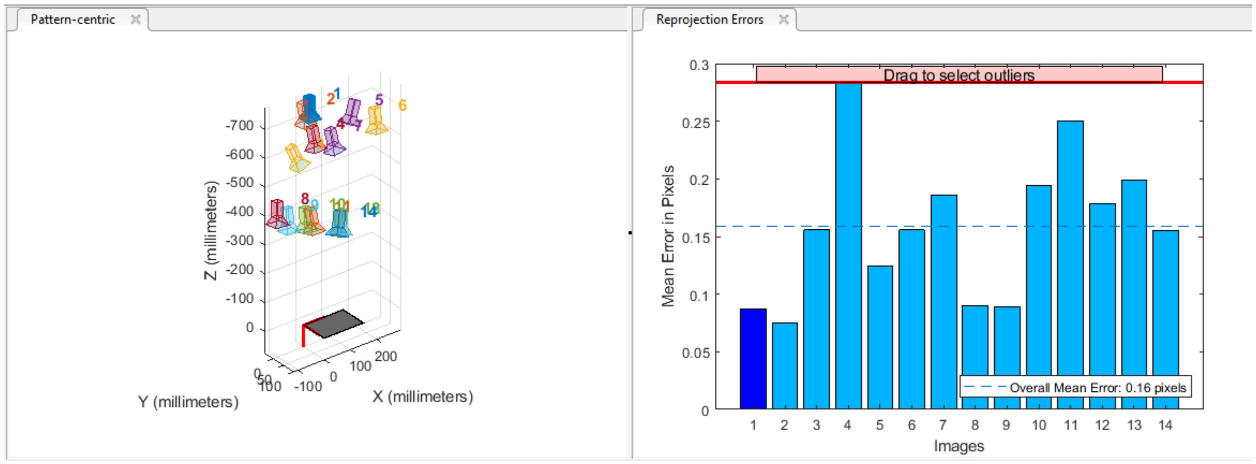
To begin, click the Add Images button. Add from file or acquire from camera a minimum of two checkerboard images. For best results, load or acquire between 10 and 20 images.

At this point you are asked about the size of the squares in the checkerboard – use the default size (25mm) and leave other settings as they are – press **OK**.

The program will now automatically detect the checkerboard in all the images. Click on 'Options' and select 3 Coefficients for Radial Distortion and mark Compute Tangential Distortion. Hereafter press '**Calibrate**'.



After the calibration, a histogram should appear that shows the mean error in pixels for the various calibration images and an estimate of the camera position relative to the checkerboard.



Hereafter you should press the button, 'Export Camera Parameters' and select to workspace. It is highly recommended that you save the workspace into a .mat-file, so you easily can load the estimated parameters later on.

2.2 Undistort image with radial and tangential distortion

The parameters you need to extract from the camera calibration are:

- `K = cameraParams.IntrinsicMatrix';`
- `radial = cameraParams.RadialDistortion;`

- `tangential = cameraParams.TangentialDistortion;`

The image, ‘image_with_radial_distortion.bmp’, should be undistorted according to the radial- and tangential distortion models which were introduced during the lecturer for convenience the algorithm is summarized below:

- Load the distorted image and the camera calibration parameters.
- Initialize a blank image with same size as the distorted input image
- Sweep over all pixels (make a for loop for each dimension) and perform the following calculations

Normalize pixel coordinates, i.e. $[x, y, 1] = K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$,

Calculate the squared radial distance to the image center, $r^2 = x^2 + y^2$

Calculate the distorted coordinates as:

$$x_d = x(1 + a_1 r^2 + a_2 r^4 + a_3 r^6) + (2p_1 xy + p_2(r^2 + 2x^2)$$

$$y_d = y(1 + a_1 r^2 + a_2 r^4 + a_3 r^6) + (p_1(r^2 + 2y^2) + 2p_2 xy)$$

Transform the coordinate back to image coordinates:

$$\begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}$$

The undistorted image is created by copying pixel values from the distorted image

$$I_{undistort}(u, v) = I(u_d, v_d)$$

Remember to round u_d, v_d to integer values for proper indexing.

Plot the undistorted and original image side-by-side and comment on the results. You can use imshow() to show the image in MATLAB.

In the figure below you can see the input image on the left and on the right how the undistorted image should look like – if done correctly.

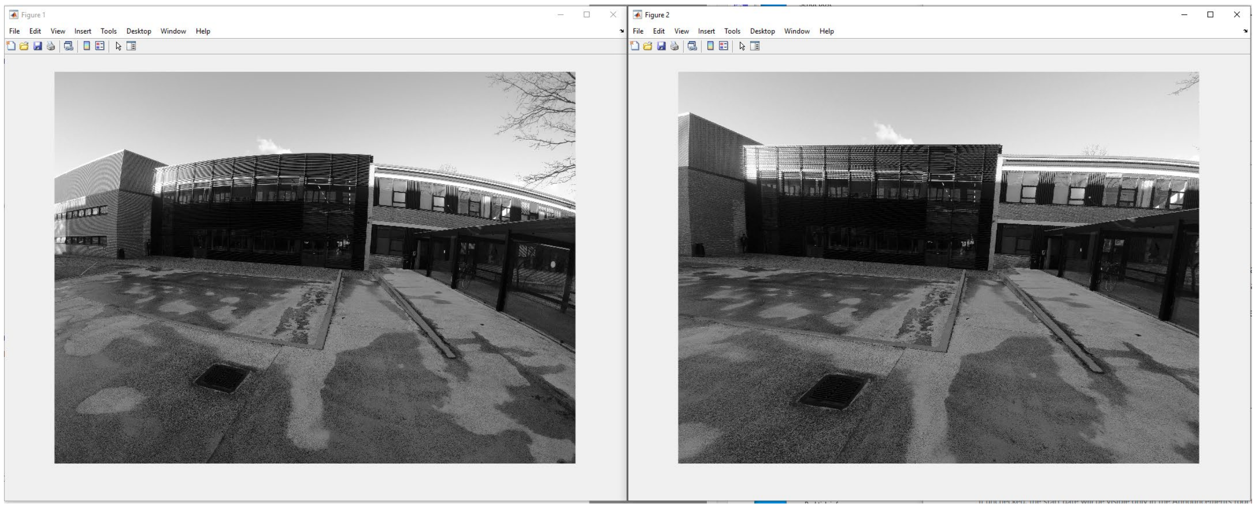


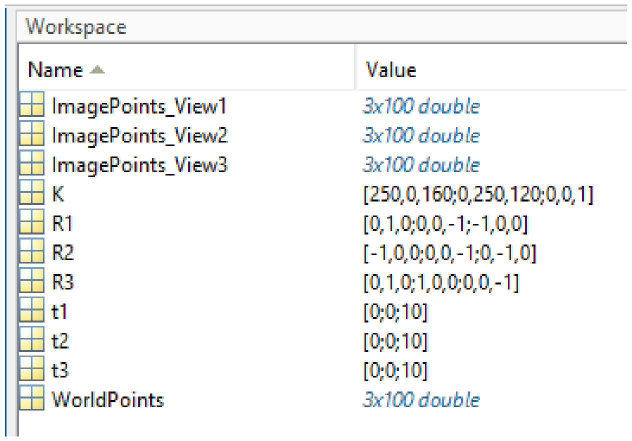
Figure 1
File Edit View Insert Tools Desktop Window Help
Figure 2
File Edit View Insert Tools Desktop Window Help

3 Spatial Resection

In this part of the assignment you should solve a simulated spatial resection problem using the P3P algorithm. This usually involves solving the side lengths of a tetrahedron as shown in the lecture slides using a 4th order polynomial and afterwards finding the position and attitude of the camera centre.

In DTU Learn you are provided with a matlab function '**p3p_sidelengths.m**', which can estimate the side lengths of the tetrahedron formed between the known world-points and the unknown camera centre. The input to this function is **<u>normalized image coordinates</u>** in homogeneous representation and the corresponding inhomogeneous 3D world points.

You are also provided with a data file, '**p3p.mat**', which contains the following parameters:



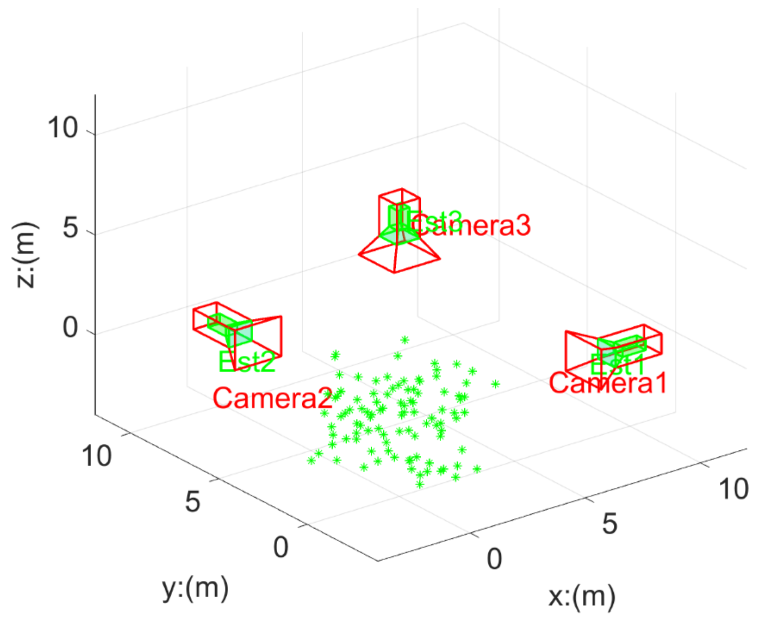
Workspace

Name	Value
ImagePoints_View1	*3x100 double*
ImagePoints_View2	*3x100 double*
ImagePoints_View3	*3x100 double*
K	[250,0,160;0,250,120;0,0,1]
R1	[0,1,0;0,0,-1;-1,0,0]
R2	[-1,0,0;0,0,-1;0,-1,0]
R3	[0,1,0;1,0,0;0,0,-1]
t1	[0;0;10]
t2	[0;0;10]
t3	[0;0;10]
WorldPoints	*3x100 double*

Where 'K' contains the intrinsic parameters of the cameras. 'WorldPoints' represent 100 different 3D points in inhomogeneous representation. These points have been captured from a simulated camera in 3 different view-points. The image coordinates in homogeneous representation is stored in {'ImagePoints_View1', 'ImagePoints_View2', 'ImagePoints_View3'}. The parameters {'R1','t1'} denotes the true position and orientation for the first view and {'R2','t2'}, {'R3','t3'} is for the second and third views respectively.

The first step is that you find the 3D camera coordinates of minimum 3 points. You can use the supplied functions to initially calculate the side lengths of the tetrahedron, but be aware that you would obtain 4 solutions. In order to find the correct answer you could use a 4th point for validation. Use the image points to form unit vectors, which then can be scaled by the side lengths in order to get the coordinates expressed in the camera coordinate system.

Hereafter you should find the location (rotation and translation) of the camera view by implementing the Kabsch algorithm (see the lecture slides and the uploaded paper on DTU Learn).


z:(m)
Camera3
Est3
Est2
Camera2
Est1
Camera1
y:(m)
x:(m)