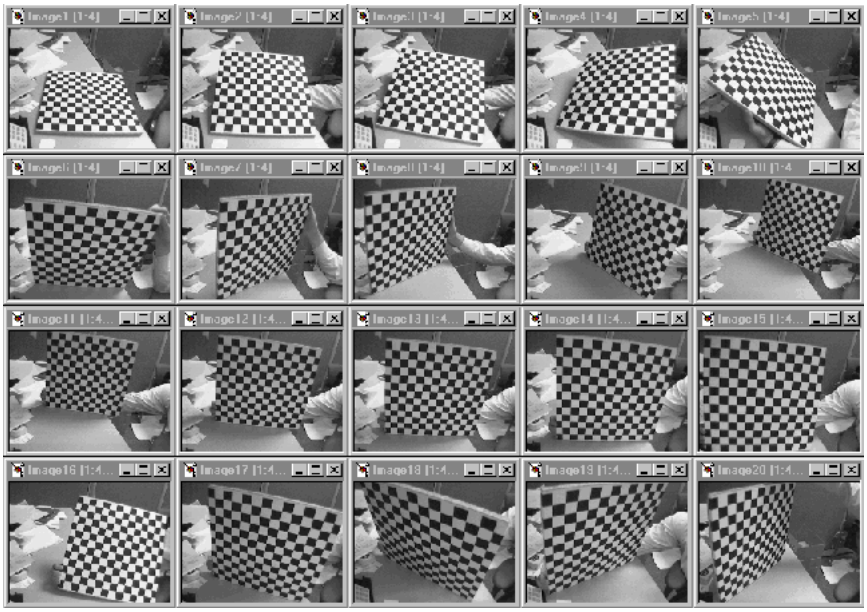
Camera Calibration Toolbox for Matlab

First calibration example - Corner extraction, calibration, additional tools

This section takes you through a complete calibration example based on a total of 20 (and 25) images of a planar checkerboard.
This example lets you learn how to use all the features of the toolbox: loading calibration images, extracting image corners, running the main calibration engine, displaying the results, controlling accuracies, adding and suppressing images, undistorting images, exporting calibration data to different formats... This example is highly recommended for someone who is just starting using the toolbox.

- Download the calibration images all at once calib_example.zip (4461Kb zipped) or one by one, and store the 20 images into a seperate folder named **calib_example**.



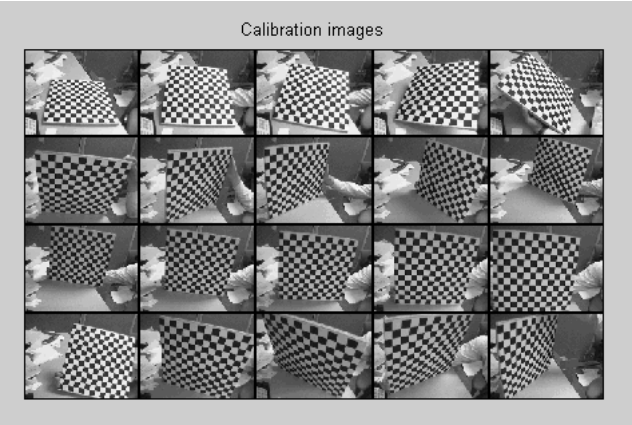
- From within matlab, go to the example folder **calib_example** containing the images.

- **Reading the images:**

  Click on the **Image names** button in the **Camera calibration tool** window. Enter the basename of the calibration images (**Image**) and the image format (**tif**).
  All the images (the 20 of them) are then loaded in memory (through the command **Read images** that is automatically executed) in the variables **I_1**, **I_2** ,...., **I_20**.
  The number of images is stored in the variable **n_ima (=20 here)**.
  The matlab window should look like this:

```
.           Image11.tif Image15.tif Image19.tif Image4.tif  Image8.tif
..          Image12.tif Image16.tif Image2.tif  Image5.tif  Image9.tif
Image1.tif  Image13.tif Image17.tif Image20.tif Image6.tif
Image10.tif Image14.tif Image18.tif Image3.tif  Image7.tif


Basename camera calibration images (without number nor suffix): Image
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm') t
Loading image 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...
done
```

  The complete set of images is also shown in thumbnail format (this images can always be regenerated by running **mosaic**):



If the **OUT OF MEMORY** error message occurred during image reading, that means that your computer does not have enough RAM to hold the entire set of images in local memory. This can easily happen of you are running the toolbox on a 128MB or less laptop for example. In this case, you can directly switch to the memory efficient version of the toolbox by running **calib_gui** and selecting the memory efficient mode of operation. The remaining steps of calibration (grid corner extraction and calibration) are exactly the same. Note that in memory efficient mode, the thumbnail image is not displayed since the calibration images are not loaded all at once.

- **Extract the grid corners:**

Click on the **Extract grid corners** button in the **Camera calibration tool** window.

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) =
```

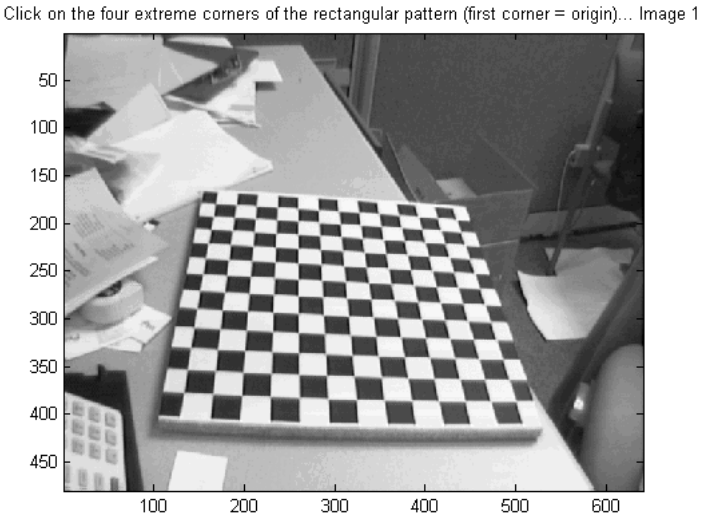
Press "enter" (with an empty argument) to select all the images (otherwise, you would enter a list of image indices like **[2 5 8 10 12]** to extract corners of a subset of images). Then, select the default window size of the corner finder: **wintx=winty=5** by pressing "enter" with empty arguments to the **wintx** and **winty** question. This leads to a effective window of size 11x11 pixels.

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) =
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)?
```

The corner extraction engine includes an automatic mechanism for counting the number of squares in the grid. This tool is specially convenient when working with a large number of images since the user does not have to manually enter the number of squares in both x and y directions of the pattern. On some very rare occasions however, this code may not predict the right number of squares. This would typically happen when calibrating lenses with extreme distortions. At this point in the corner extraction procedure, the program gives the option to the user to disable the automatic square counting code. In that special mode, the user would be prompted for the square count for every image. In this present example, it is perfectly appropriate to keep working in the default mode (i.e. with automatic square counting activated), and therefore, simply press "enter" with an empty argument. (**NOTE:** it is generally recommended to first use the corner extraction code in this default mode, and then, if need be, re-process the few images with "problems")

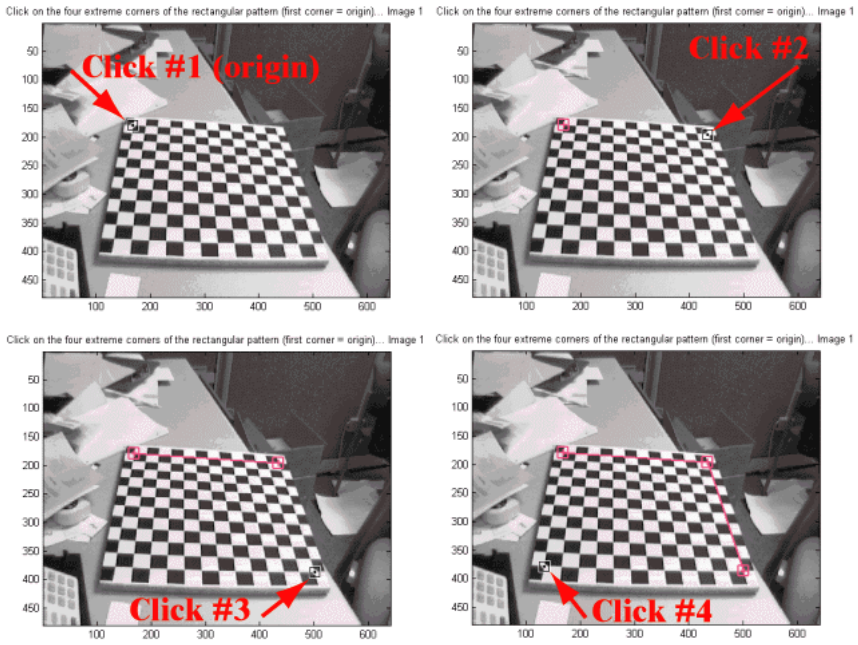
```
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)?

Processing image 1...
Using (wintx,winty)=(5,5) - Window size = 11x11      (Note: To reset the window size, run script clearwin)
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
```

The first calibration image is then shown on Figure 2:

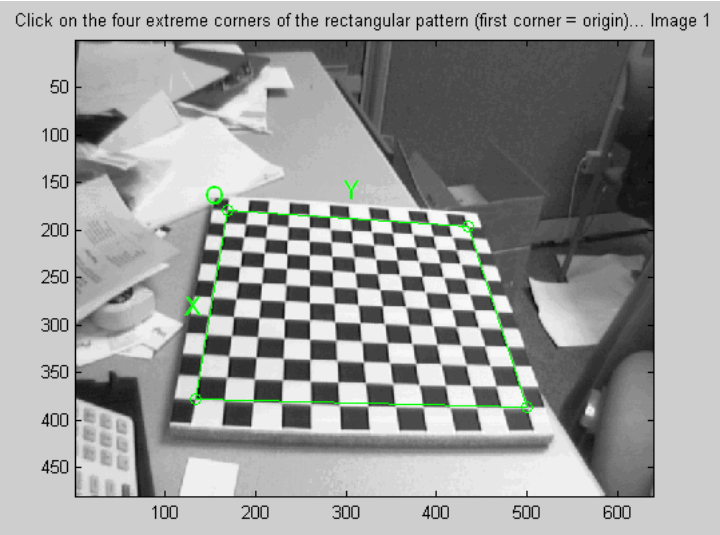


Click on the four extreme corners on the rectangular checkerboard pattern. The clicking locations are shown on the four following figures (**WARNING:** try to click accurately on the four corners, at most 5 pixels away from the corners. Otherwise some of the corners might be missed by the detector).

Ordering rule for clicking: The first clicked point is selected to be associated to the origin point of the reference frame attached to the grid. The other three points of the rectangular grid can be clicked in any order. This first-click rule is especially important if you need to calibrate externally multiple cameras (i.e. compute the relative positions of several cameras in space). When dealing with multiple cameras, the same grid pattern reference frame needs to be consistently selected for the different camera images (i.e. grid points need to correspond across the different camera views). For example, it is a requirement to run the stereo calibration toolbox **stereo_gui.m** (try **help stereo_gui** and visit the fifth calibration example page for more information).



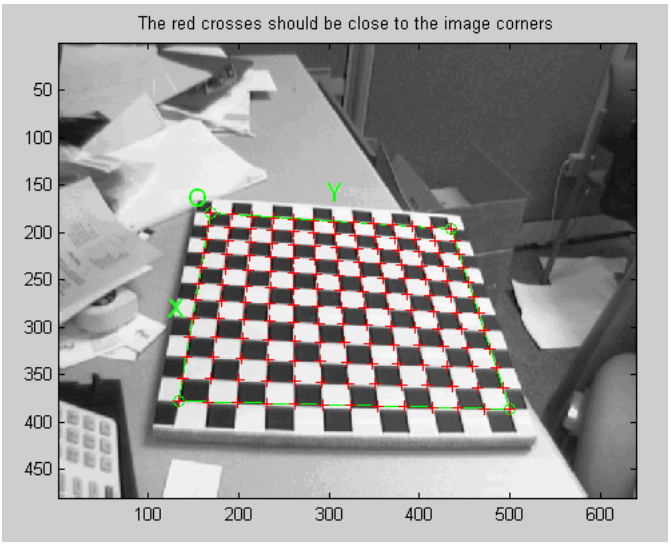
The boundary of the calibration grid is then shown on Figure 2:



Enter the sizes **dX** and **dY** in X and Y of each square in the grid (in this case, **dX=dY=30mm**=default values):

```
Size dX of each square along the X direction ([]=30mm) = 30
Size dY of each square along the Y direction ([]=30mm) = 30
```

Note that you could have just pressed "enter" with an empty argument to select the default values. The program automatically counts the number of squares in both dimensions, and shows the predicted grid corners in absence of distortion:

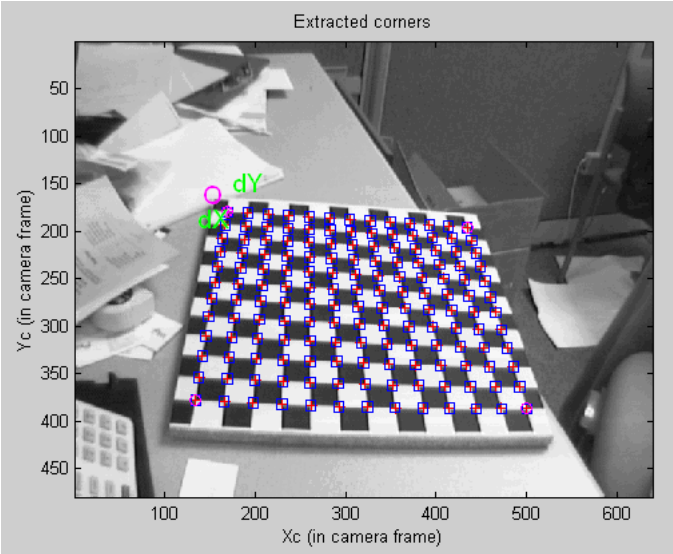



```
If the guessed grid corners (red crosses on the image) are not close to the actual corners,
it is necessary to enter an initial guess for the radial distortion factor kc (useful for subpixel detection)
Need of an initial guess for distortion? ([]=no, other=yes)
```

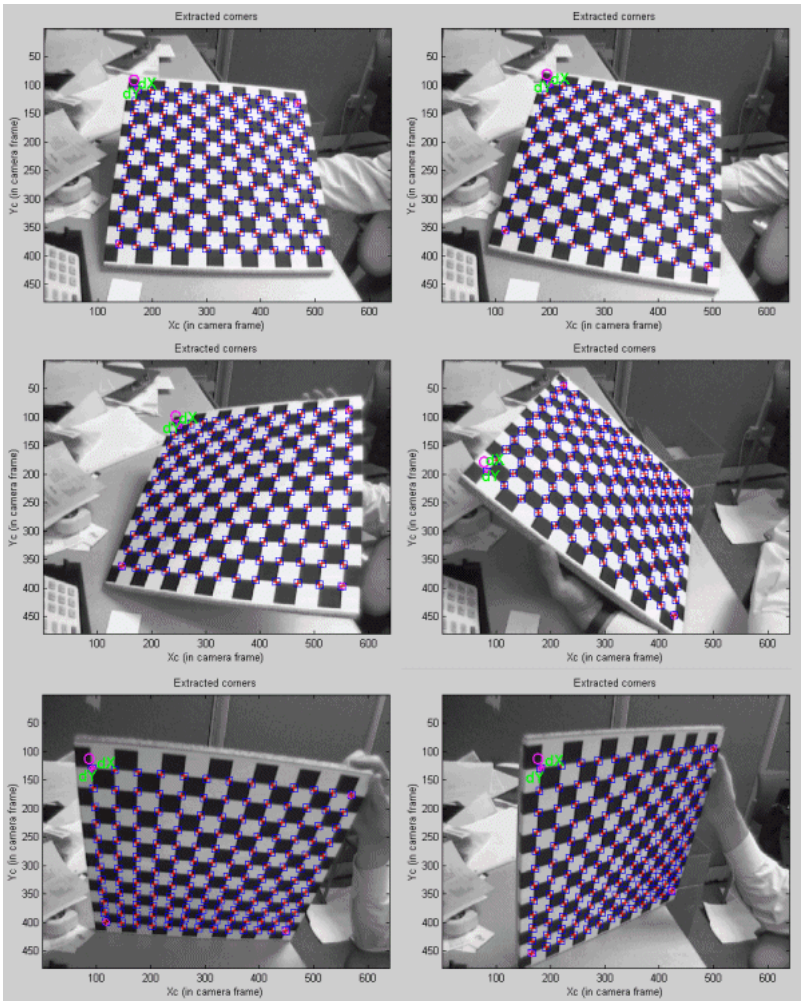
If the predicted corners are close to the real image corners, then the following step may be skipped (if there is not much image distortion). This is the case in that present image: the predicted corners are close enough to the real image corners. Therefore, it is not necessary to "help" the software to detect the image corners by entering a guess for radial distortion coefficient. Press "enter", and the corners are automatically extracted using those positions as initial guess.

```
Corner extraction...
```

The image corners are then automatically extracted, and displayed on figure 3 (the blue squares around the corner points show the limits of the corner finder window):



The corners are extracted to an accuracy of about 0.1 pixel.
Follow the same procedure for the 2nd, 3rd, ... , 14th images. For example, here are the detected corners of image 2, 3, 4, 5, 6 and 7:

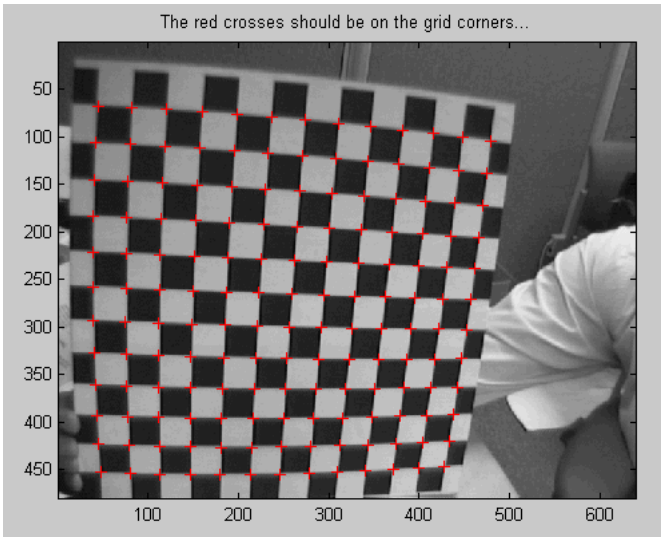


Observe the square dimensions **dX**, **dY** are always kept to their original values (30mm).
Sometimes, the predicted corners are not quite close enough to the real image corners to allow for an effective corner extraction. In that case, it is necessary to refine the predicted corners by entering a guess for lens distortion coefficient. This situation occurs at image 15. On that image, the predicted corners are:

Observe that some of the predicted corners within the grid are far enough from the real grid corners to result into wrong extractions. The cause: image distortion. In order to help the system make a better guess of the corner locations, the user is free to manually input a guess for the first order lens distortion coefficient **kc** (to be precise, it is the first entry of the full distortion coefficient vector **kc** described at this page). In order to input a guess for the lens distortion coefficient, enter a non-empty string to the question **Need of an initial guess for distortion?** (for example **1**). Enter then a distortion coefficient of **kc=-0.3** (in practice, this number is typically between -1 and 1).

```
If the guessed grid corners (red crosses on the image) are not close to the actual corners,
it is necessary to enter an initial guess for the radial distortion factor kc (useful for subpixel detection)
Need of an initial guess for distortion? ([]=no, other=yes) 1
Use number of iterations provided
Use focal provided
Estimated focal: 723.3095 pixels
Guess for distortion factor kc ([]=0): -.3
Satisfied with distortion? ([]=no, other=yes)
```

According to this distortion, the new predicted corner locations are:

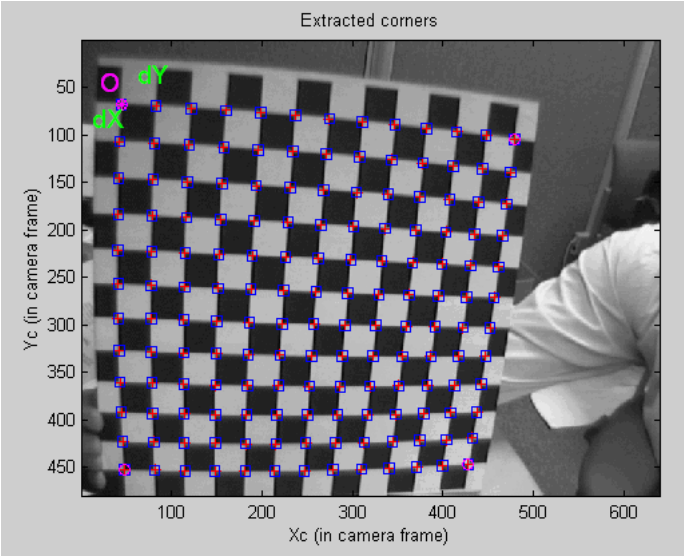


If the new predicted corners are close enough to the real image corners (this is the case here), input any non-empty string (such as **1**) to the question **Satisfied with distortion?**. The subpixel corner locations are then computed using the new predicted locations (with image distortion) as initial guesses:

```
Corner extraction...
```

If we had not been satisfied, we would have entered an empty-string to the question **Satisfied with distortion?** (by directly pressing "enter"), and then tried a new distortion coefficient **kc**. You may repeat this process as many times as you want until satisfied with the prediction (**side note:** the values of distortion used at that stage are only used to help corner extraction and will not affect at all the next main calibration step. In other words, these values are neither used as final distortion coefficients, nor used as initial guesses for the true distortion coefficients estimated through the calibration optimization stage).

The final detected corners are shown on Figure 3:



Repeat the same procedure on the remaining 5 images (16 to 20). On these images however, do not use the predicted distortion option, even if the extracted corners are not quite right. In the next steps, we will correct them (in this example, we could have not used this option for image 15, but that was quite useful for illustration).

After corner extraction, the matlab data file **calib_data.mat** is automatically generated. This file contains all the information gathered throughout the corner extraction stage (image coordinates, corresponding 3D grid coordinates, grid sizes, ...). This file is only created in case of emergency when for example matlab is abruptly terminated before saving. Loading this file would prevent you from having to click again on the images.

During your own calibrations, when there is a large amount of distortion in the image, the program may not be able to automatically count the number of squares in the grid. In that case, the number of squares in both X and Y directions have to be entered manually. This should not occur in this present example.

Another problem may arise when performing your own calibrations. If the lens distortions are really too severe (for fisheye lenses for example), the simple guiding tool based on a single distortion coefficient **kc** may not be sufficient to provide good enough initial guesses for the corner locations. For those few difficult cases, a script program is included in the toolbox that allows for a completely manual corner extraction (i.e. one click per corner). The script file is called **manual_corner_extraction.m** (in memory efficient mode, you should use **manual_corner_extraction_no_read.m** instead) and should be executed AFTER the traditional corner extaction code (the script relies on data that were computed by the traditional corner extraction code -square count, grid size, order of points, ...- even if the corners themselves were wrongly detected). Obviously, this method for corner extraction could be extremely time consuming when applied on a lot of images. It therefore recommended to use it as a last resort when everything else has failed. Most users should never have to worry about this, and it will not happen in this present calibration example.

- **Main Calibration step:**

After corner extraction, click on the button **Calibration** of the **Camera calibration tool** to run the main camera calibration procedure.
Calibration is done in two steps: first initialization, and then nonlinear optimization.
The initialization step computes a closed-form solution for the calibration parameters based not including any lens distortion (program name: **init_calib_param.m**). The non-linear optimization step minimizes the total reprojection error (in the least squares sense) over all the calibration parameters (9 DOF for intrinsic: focal, principal point, distortion coefficients, and 6*20 DOF extrinsic => 129 parameters). For a complete description of the calibration parameters, click on that link. The optimization is done by iterative gradient descent with an explicit (closed-form) computation of the Jacobian matrix (program name: **go_calib_optim.m**).

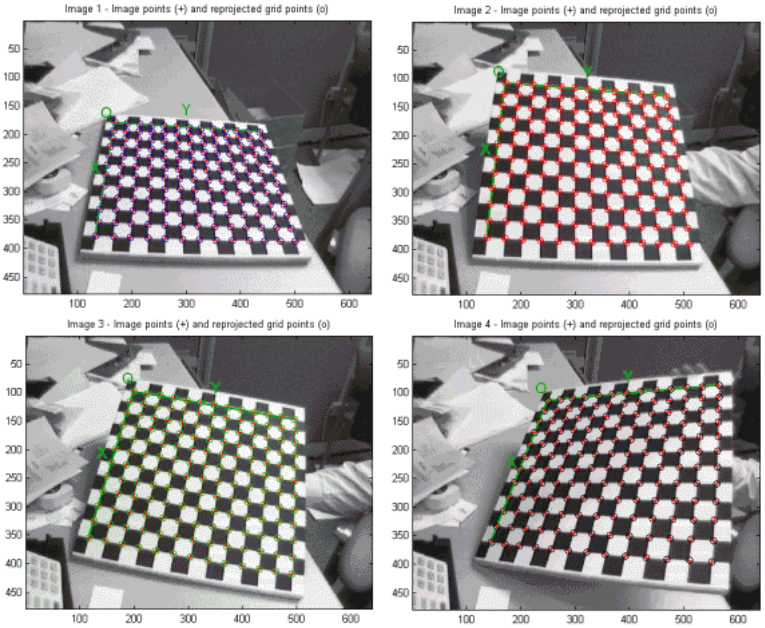
```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
Initialization of the principal point at the center of the image.
Initialization of the image distortion to zero.
Initialization of the intrinsic parameters using the vanishing points of planar patterns.

Initialization of the intrinsic parameters - Number of images: 20


Calibration parameters after initialization:

Focal Length:          fc = [ 671.13759   680.77186 ]
Principal point:       cc = [ 319.50000   239.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel = 90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 20
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 661.67001   662.82858 ] ± [ 1.17913   1.26567 ]
Principal point:       cc = [ 306.09590   240.78987 ] ± [ 2.38443   2.17481 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.26425   0.22645   0.00020   0.00023  0.00000 ] ± [ 0.00934   0.03826   0.00052   0.00053  0.00000 ]
Pixel error:          err = [ 0.45330   0.38916 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).


Recommendation: Some distortion coefficients are found equal to zero (within their uncertainties).
                To reject them from the optimization set est_dist=[1;1;0;0;0] and run Calibration
```

The Calibration parameters are stored in a number of variables. For a complete description of them, visit this page. Notice that the skew coefficient **alpha_c** and the 6th order radial distortion coefficient (the last entry of **kc**) have not been estimated (this is the default mode). Therefore, the angle between the x and y pixel axes is 90 degrees. In most practical situations, this is a very good assumption. However, later on, a way of introducing the skew coefficient **alpha_c** in the optimization will be presented.

Observe that only 11 gradient descent iterations are required in order to reach the minimum. This means only 11 evaluations of the reprojection function + Jacobian computation and inversion. The reason for that fast convergence is the quality of the initial guess for the parameters computed by the initialization procedure.
For now, ignore the recommendation of the system to reduce the distortion model. The reprojection error is still too large to make a judgement on the complexity of the model. This is mainly because some of the grid corners were not very precisely extracted for a number of images.
Click on **Reproject on images** in the **Camera calibration tool** to show the reprojections of the grids onto the original images. These projections are computed based on the current intrinsic and extrinsic parameters. Input an empty string (just press "enter") to the question **Number(s) of image(s) to show ([] = all images)** to indicate that you want to show all the images:

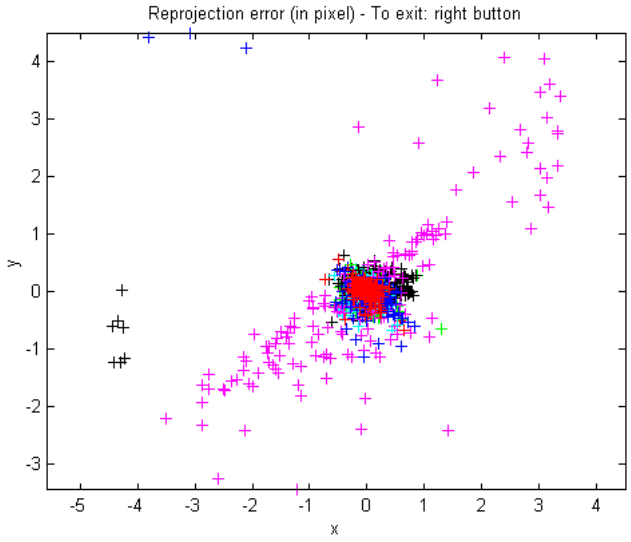
```
Number(s) of image(s) to show ([] = all images) =
```

The following figures shows the first four images with the detected corners (red crosses) and the reprojected grid corners (circles).

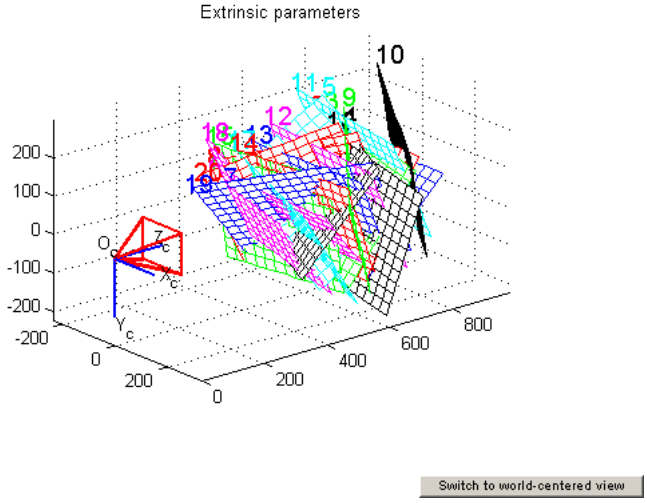



```
Number(s) of image(s) to show ([] = all images) =
Pixel error:      err = [0.45330   0.38917] (all active images)
```

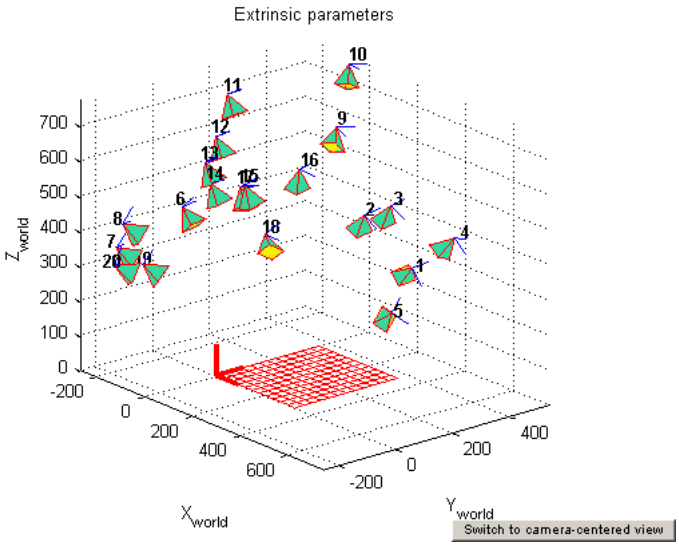
The reprojection error is also shown in the form of color-coded crosses:



In order to exit the error analysis tool, right-click on anywhere on the figure (you will understand later the use of this option).
Click on **Show Extrinsic** in the **Camera calibration tool**. The extrinsic parameters (relative positions of the grids with respect to the camera) are then shown in a form of a 3D plot:



On this figure, the frame **(O_c,X_c,Y_c,Z_c)** is the camera reference frame. The red pyramid corresponds to the effective field of view of the camera defined by the image plane. To switch from a "camera-centered" view to a "world-centered" view, just click on the **Switch to world-centered view** button located at the bottom-left corner of the figure.



On this new figure, every camera position and orientation is represented by a green pyramid. Another click on the **Switch to camera-centered view** button turns the figure back to the "camera-centered" plot.

Looking back at the error plot, notice that the reprojection error is very large across a large number of figures. The reason for that is that we have not done a very careful job at extracting the corners on some highly distorted images (a better job could have been done by using the predicted distortion option). Nevertheless, we can correct for that now by recomputing the image corners on all images automatically. Here is the way it is going to be done: press on the **Recomp. corners** button in the main **Camera calibration tool** and select once again a corner finder window size of **wintx = winty = 5** (the default values):

```
Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11
Number(s) of image(s) to process ([] = all images) =
```

To the question **Number(s) of image(s) to process ([] = all images)** press "enter" with an empty argument to recompute the corners on all the images. Enter then the mode of extraction: the automatic mode (auto) uses the re-projected grid as initial guess locations for the corner, the manual mode lets the user extract the corners manually (the traditional corner extraction method). In the present case, the reprojected grid points are very close to the actual image corners. Therefore, we select the automatic mode: press "enter" with an empty string. The corners on all images are then recomputed. your matlab window should look like:

```
Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11
Number(s) of image(s) to process ([] = all images) =
Use the projection of 3D grid or manual click ([]=auto, other=manual):
Processing image 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...
done
```

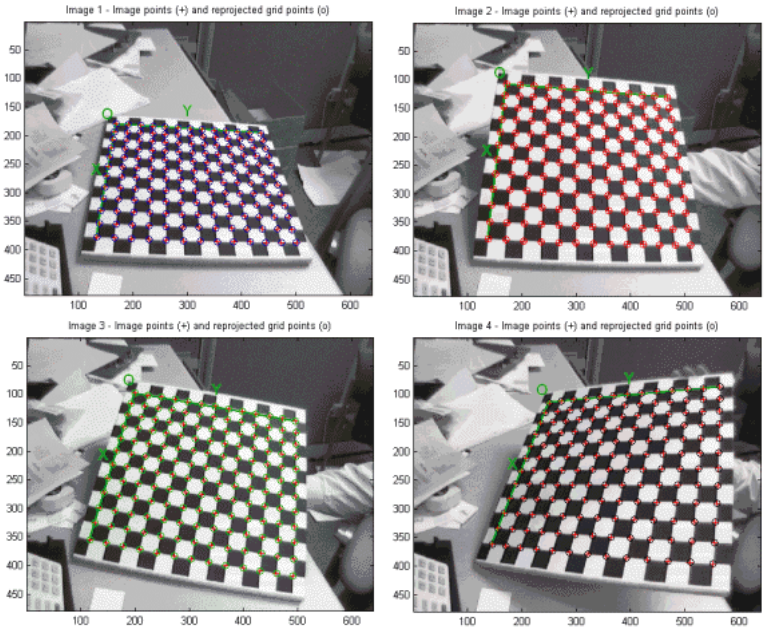
Run then another calibration optimization by clicking on **Calibration**:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 20
Gradient descent iterations: 1...2...3...4...5...6...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.39535   657.76309 ] ± [ 0.34691   0.37111 ]
Principal point:       cc = [ 302.98368   242.61630 ] ± [ 0.70546   0.64553 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25584   0.12758   -0.00021   0.00003  0.00000 ] ± [ 0.00271   0.01076   0.00015   0.00014  0.00000 ]
Pixel error:          err = [ 0.12668   0.12604 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

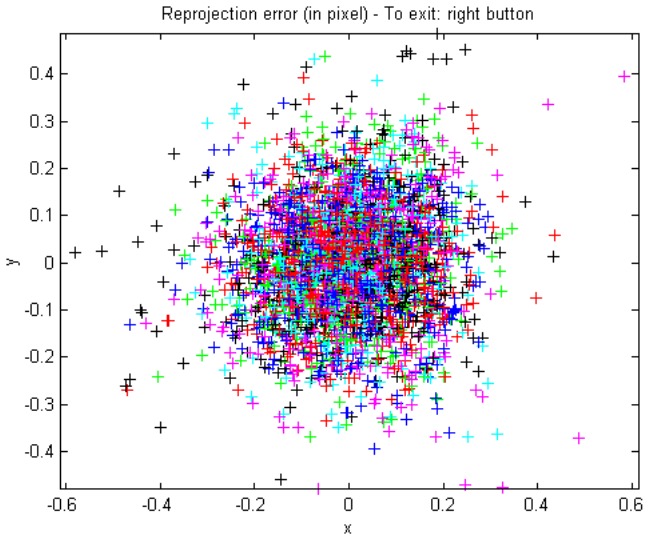
Observe that only six iterations were necessary for convergence, and no initialization step was performed (the optimization started from the previous calibration result). The two values **0.12668** and **0.12604** are the standard deviation of the reprojection error (in pixel) in both x and y directions respectively. Observe that the uncertainties on the calibration parameters are also estimated. The numerical values are approximately three times the standard deviations.
After optimization, click on **Save** to save the calibration results (intrinsic and extrinsic) in the matlab file **Calib_Results.mat**

```
Saving calibration results under Calib_Results.mat
done
```

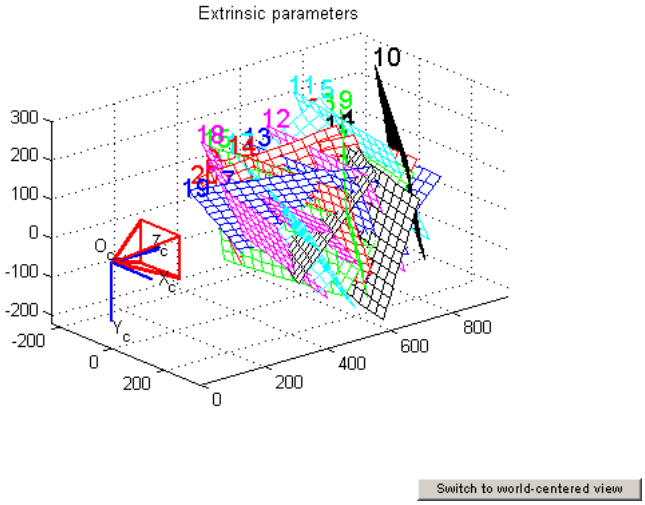
For a complete description of the calibration parameters, click on that link.
Once again, click on **Reproject on images** to reproject the grids onto the original calibration images. The four first images look like:



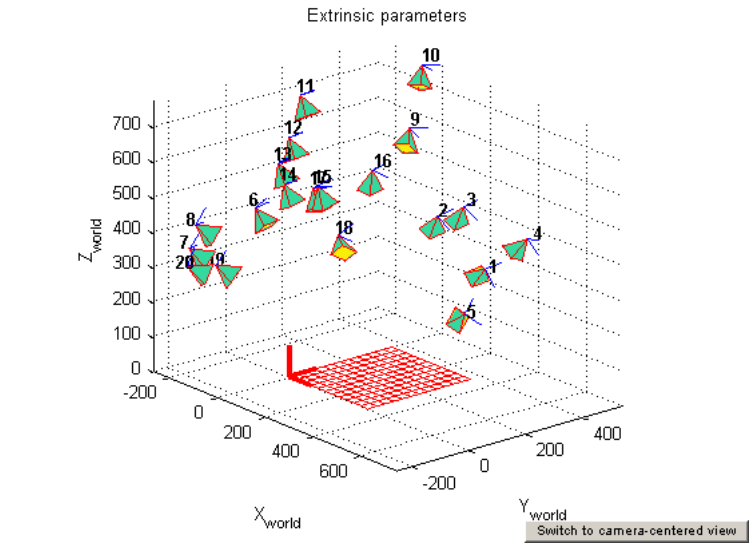
Click on **Analyse error** to view the new reprojection error (observe that the error is much smaller than before):



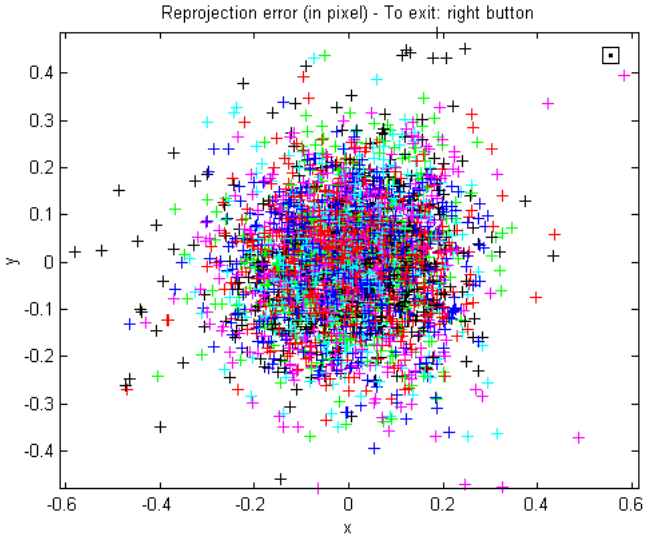
After right-clicking on the error figure (to exit the error-analysis tool), click on **Show Extrinsic** to show the new 3D positions of the grids with respect to the camera:



A simple click on the **Switch to world-centered view** button changes the figure to this:



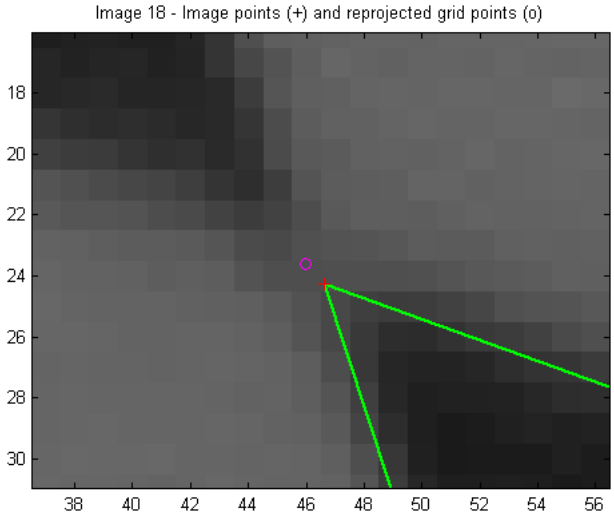
The tool **Analyse error** allows you to inspect which points correspond to large errors. Click on **Analyse error** and click on the figure region that is shown here (upper-right figure corner):



After clicking, the following information appears in the main Matlab window:

```
Selected image: 18
Selected point index: 145
Pattern coordinates (in units of (dX,dY)): (X,Y)=(0,0)
Image coordinates (in pixel): (45.63,23.27)
Pixel error = (0.58234,0.39466)
Window size: (wintx,winty) = (5,5)
```

This means that the corresponding point is on image 18, at the grid coordinate (0,0) in the calibration grid (at the origin of the pattern). The following image shows a close up of that point on the calibration image (before, exit the error inspection tool by clicking on the right mouse button anywhere within the figure):



The error inspection tool is very useful in cases where the corners have been badly extracted on one or several images. In such a case, the user can recompute the corners of the specific images using a different window size (larger or smaller).

For example, let us recompute the image corners using a window size (**wintx=winty=9**) for all 20 images except for images 20 (use **wintx=winty=5**), images 5, 7, 8, 19 (use **wintx=winty=7**), and images 18 (use **wintx=winty=8**). The extraction of the corners should be performed with three calls of **Recomp. corners**. At the first call of **Recomp. corners**, select **wintx=winty=9**, choose to process images 1, 2, 3, 4, 6, 9, 10, 11, 12, 13, 14, 15, 16 and 17, and select the automatic mode (the reprojections are already very close to the actual image corners):

```
Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([] = 5) = 9
winty ([] = 5) = 9
Window size = 19x19
Number(s) of image(s) to process ([] = all images) = [1:4 6 9:17]
Use the projection of 3D grid or manual click ([]=auto, other=manual):
Processing image 1...2...3...4...6...9...10...11...12...13...14...15...16...17...
done
```

At the second call of **Recomp. corners**, select **wintx=winty=8**, choose to process image 18 and select once again the automatic mode:

```
Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([] = 5) = 8
winty ([] = 5) = 8
Window size = 17x17
Number(s) of image(s) to process ([] = all images) = 18
Use the projection of 3D grid or manual click ([]=auto, other=manual):
Processing image 18...
done
```

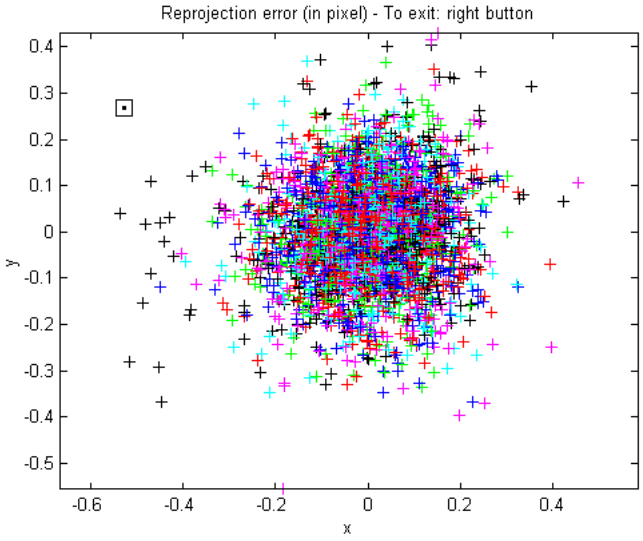
At the third call of **Recomp. corners**, select **wintx=winty=7**, choose to process images 5, 7, 8 and 19 and select once again the automatic mode:

```
Re-extraction of the grid corners on the images (after first calibration)
Window size for corner finder (wintx and winty):
wintx ([] = 5) = 7
winty ([] = 5) = 7
Window size = 15x15
Number(s) of image(s) to process ([] = all images) = [5 7 8 19]
Use the projection of 3D grid or manual click ([]=auto, other=manual):
Processing image 5...7...8...19...
done
```

Re-calibrate by clicking on **Calibration**:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 20
Gradient descent iterations: 1...2...3...4...5...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.46290   657.94673 ] ± [ 0.31819   0.34046 ]
Principal point:       cc = [ 303.13665   242.56935 ] ± [ 0.64682   0.59218 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25403   0.12143   -0.00021   0.00002  0.00000 ] ± [ 0.00248   0.00986   0.00013   0.00013  0.00000 ]
Pixel error:          err = [ 0.11689   0.11500 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

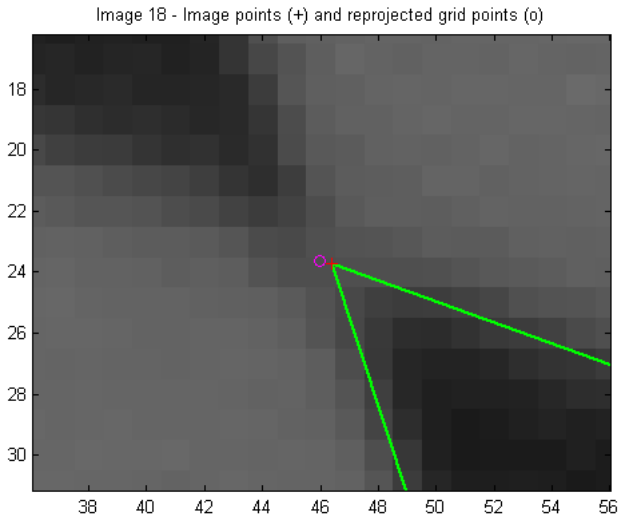
Observe that the reprojection error (**0.11689**,**0.11500**) is slightly smaller than the previous one. In addition, observe that the uncertainties on the calibration parameters are also smaller. Inspect the error by clicking on **Analyse error**:



Let us look at the previous point of interest on image 18, at the grid coordinate (0,0) in the calibration grid. For that, click on **Reproject on images** and select to show image 18 only (of course, before that, you must exit the error inspection tool by right-cliking within the window):

```
Number(s) of image(s) to show ([] = all images) = 18
Pixel error:      err = [0.11672   0.11510] (all active images)
```

A close view at the point of interest (on image 18) shows a smaller reprojection error:



Click once again on **Save** to save the calibration results (intrinsic and extrinsic) in the matlab file **Calib_Results.mat**

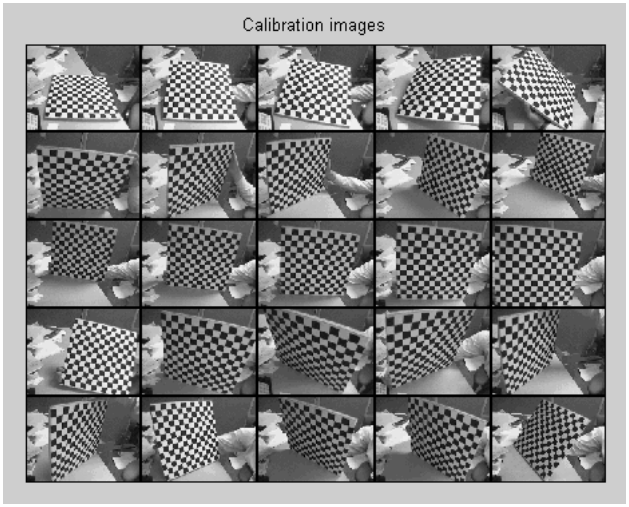
```
WARNING: File Calib_Results.mat already exists
Copying the current Calib_Results.mat file to Calib_Results_old0.mat

Saving calibration results under Calib_Results.mat
done
```

Observe that the previous calibration result file was copied under **Calib_Results_old0.mat** (just in case you want to use it later on).

Download now the five additional images Image21.tif, Image22.tif, Image23.tif, Image24.tif and Image25.tif and re-calibrate the camera using the complete set of 25 images without recomputing everything from scratch.
After saving the five additional images in the current directory, click on **Read images** to read the complete new set of images:

```
Loading image 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...21...22...23...24...25...
done
```

To show a thumbnail image of all calibration images, run **mosaic** (if you are running in memory efficient mode, run **mosaic_no_read** instead).



Click on **Extract grid corners** to extract the corners on the five new images, with default window sizes **wintx=winty=5**:

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) = 21:25
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11

Processing image 21...
Using (wintx,winty)=(5,5) - Window size = 11x11      (Note: To reset the window size, run script clearwin)
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
```

And go on with the traditional corner extraction on the five images. Afterwards, run another optimization by clicking on **Calibration**:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...5...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.25377   657.68169 ] ± [ 0.28772   0.29230 ]
Principal point:       cc = [ 302.97978   242.50018 ] ± [ 0.58444   0.55880 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25362   0.11974   -0.00026   0.00003  0.00000 ] ± [ 0.00233   0.00951   0.00012   0.00012  0.00000 ]
Pixel error:          err = [ 0.11911   0.11652 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Next, recompute the image corners of the four last images using different window sizes. Use **wintx=winty=9** for images 22 and 24, use **wintx=winty=8** for image 23, and use **wintx=winty=6** for image 25. Follow the same procedure as previously presented (three calls of **Recomp. corners** should be enough). After recomputation, run **Calibration** once again:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.29599   657.74627 ] ± [ 0.28562   0.29017 ]
Principal point:       cc = [ 302.98382   242.47752 ] ± [ 0.58016   0.55480 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25340   0.11884   -0.00027   0.00002  0.00000 ] ± [ 0.00232   0.00945   0.00012   0.00012  0.00000 ]
Pixel error:          err = [ 0.11801   0.11592 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Click once again on **Save** to save the calibration results (intrinsic and extrinsic) in the matlab file **Calib_Results.mat**

```
WARNING: File Calib_Results.mat already exists
Copying the current Calib_Results.mat file to Calib_Results_old1.mat

Saving calibration results under Calib_Results.mat
done
```

As an exercise, recalibrate based on all images, except images 16, 18, 19, 24 and 25 (i.e. calibrate on a new set of 20 images).
Click on **Add/Suppress images**.

```
This function is useful to select a subset of images to calibrate

There are currently 25 active images selected for calibration (out of 25):
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24 and 25.

You probably want to suppress images
Number(s) of image(s) to suppress ([] = no image) =
```

Enter the list of images to suppress (**[16 18 19 24 25]**):

```
Number(s) of image(s) to suppress ([] = no image) = [16 18 19 24 25]

There is now a total of 20 active images for calibration:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 20, 21, 22 and 23.

You may now run 'Calibration' to recalibrate based on this new set of images.
```

Click on **Calibration** to recalibrate:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 20
Gradient descent iterations: 1...2...3...4...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.58756   658.13078 ] ± [ 0.32586   0.34986 ]
Principal point:       cc = [ 303.27138   242.56161 ] ± [ 0.64229   0.60094 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25500   0.12903   -0.00029   -0.00007  0.00000 ] ± [ 0.00269   0.01195   0.00014   0.00013  0.00000
Pixel error:          err = [ 0.10988   0.11036 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

It is up to user to use the function **Add/Suppress images** to activate or de-activate images. In effect, this function simply updates the binary vector **active_images** setting zeros to inactive images, and ones to active images.
Next, load the old calibration results previously saved in **Calib_Results.mat** by clicking on **Load**:

```
Loading calibration results from Calib_Results.mat
done
```

The setup is now back to what it was before supressing 5 images 16, 18, 19, 24 and 25. Let us now run a calibration by including the skew factor **alpha_c** describing the angle between the x and y pixel axes. For that, set the variable **est_alpha** to one (at the matlab prompt). As an exercise, let us fit the radial distortion model up to the 6th order (up to now, it was up to the 4th order, with tangential distortion). For that, set the last entry of the vector **est_dist** to one:

```
» est_dist = [1;1;1;1;1];
» est_alpha = 1;
```

Then, run a new calibration by clicking on **Calibration**:

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew optimized (est_alpha=1). To disable skew estimation, set est_alpha=0.

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...5...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.35518   657.80170 ] ± [ 0.29740   0.30426 ]
Principal point:       cc = [ 302.72777   242.34976 ] ± [ 0.59104   0.55731 ]
Skew:             alpha_c = [ 0.00042 ] ± [ 0.00019  ]   => angle of pixel axes = 89.97622 ± 0.01092 degrees
Distortion:            kc = [ -0.25636   0.14595   -0.00028   0.00005  -0.06886 ] ± [ 0.00528   0.04596   0.00012   0.00012  0.11354
Pixel error:          err = [ 0.11733   0.11591 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Observe that after optimization, the skew coefficient is very close to zero (**alpha_c = 0.00042**). This leads to an angle between x and y pixel axes very close to 90 degrees (89.976 degrees). This justifies the previous assumption of rectangular pixels (**alpha_c = 0**). In addition, notice that the uncertainty on the 6th order radial distortion coefficient is very large (the uncertainty is much larger than the absolute value of the coefficient). In this case, it is preferable to disable its estimation. In this case, set the last entry of est_dist to zero:

```
» est_dist(5) = 0;
```

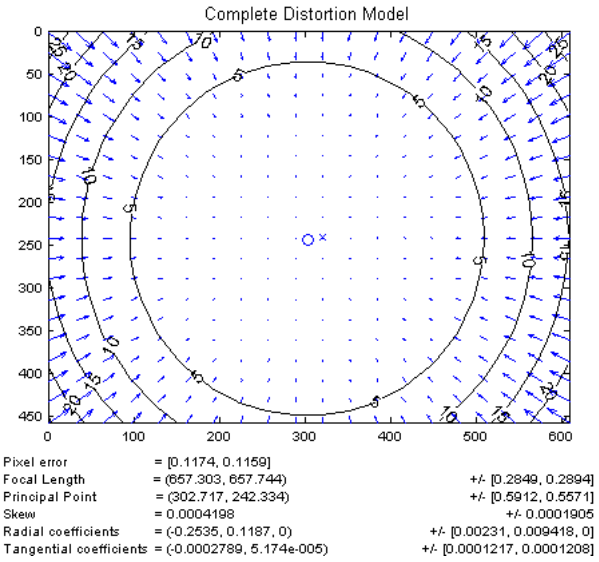
Then, run calibration once again by clicking on **Calibration**:

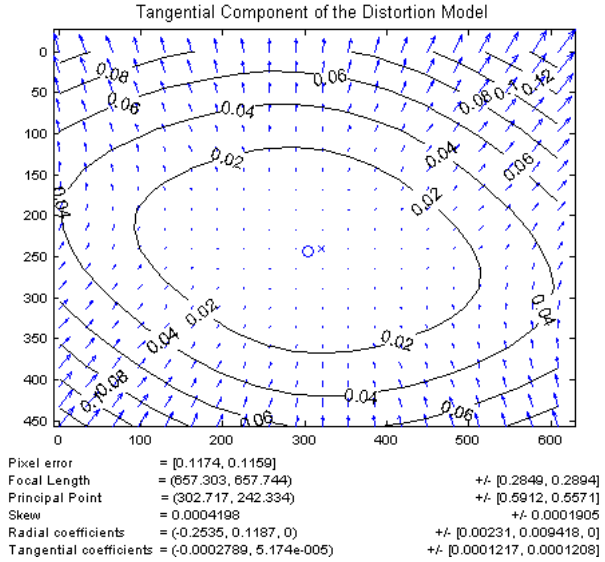
```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew optimized (est_alpha=1). To disable skew estimation, set est_alpha=0.
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...5...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 657.30254   657.74391 ] ± [ 0.28487   0.28937 ]
Principal point:       cc = [ 302.71656   242.33386 ] ± [ 0.59115   0.55710 ]
Skew:             alpha_c = [ 0.00042 ] ± [ 0.00019  ]   => angle of pixel axes = 89.97595 ± 0.01092 degrees
Distortion:            kc = [ -0.25349   0.11868   -0.00028   0.00005  0.00000 ] ± [ 0.00231   0.00942   0.00012   0.00012  0.00000 ]
Pixel error:          err = [ 0.11743   0.11585 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

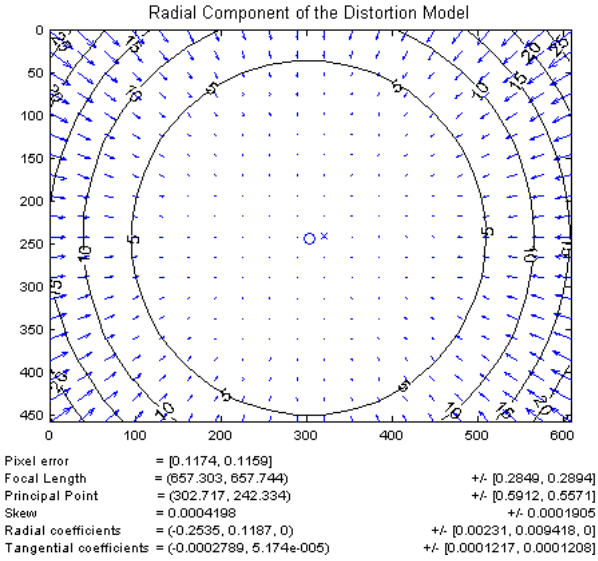
Judging the result of calibration satisfactory, let us save the current calibration parameters by clicking on **Save**:

```
WARNING: File Calib_Results.mat already exists
Copying the current Calib_Results.mat file to Calib_Results_old2.mat

Saving calibration results under Calib_Results.mat
done
```

In order to make a decision on the appropriate distortion model to use, it is sometimes very useful to visualize the effect of distortions on the pixel image, and the importance of the radial component versus the tangential component of distortion. For this purpose, run the script **visualize_distortions** at the matlab prompt (this function is not yet linked to any button in the GUI window). The three following images are then produced:







The first figure shows the impact of the complete distortion model (radial + tangential) on each pixel of the image. Each arrow represents the effective displacement of a pixel induced by the lens distortion. Observe that points at the corners of the image are displaced by as much as 25 pixels. The second figure shows the impact of the tangential component of distortion. On this plot, the maximum induced displacement is 0.14 pixel (at the upper left corner of the image). Finally, the third figure shows the impact of the radial component of distortion. This plot is very similar to the full distortion plot, showing the tangential component could very well be discarded in the complete distortion model. On the three figures, the cross indicates the center of the image, and the circle the location of the principal point.

Now, just as an exercise (not really recommended in practice), let us run an optimization without the lens distortion model (by enforcing **kc = [0;0;0;0;0]**) and without aspect ratio (by enforcing both components of **fc** to be equal). For that, set the binary variables **est_dist** to **[0;0;0;0;0]** and **est_aspect_ratio** to **0** at the matlab prompt:

```
» est_dist = [0;0;0;0;0];
» est_aspect_ratio = 0;
```

Then, run a new optimization by clicking on **Calibration**:

```
Aspect ratio not optimized (est_aspect_ratio = 0) -> fc(1)=fc(2). Set est_aspect_ratio to 1 for estimating aspect ratio.
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew optimized (est_alpha=1). To disable skew estimation, set est_alpha=0.
Distortion not fully estimated (defined by the variable est_dist):
     Second order distortion not estimated (est_dist(1)=0).
     Fourth order distortion not estimated (est_dist(2)=0).
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
     Tangential distortion not estimated (est_dist(3:4)~=[1;1]).

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...5...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 666.59533   666.59533 ] ± [ 2.44825   2.44825 ]
Principal point:       cc = [ 311.17378   244.99519 ] ± [ 1.82387   1.91165 ]
Skew:             alpha_c = [ -0.00216 ] ± [ 0.00172  ]   => angle of pixel axes = 90.12404 ± 0.09843 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000  0.00000 ] ± [ 0.00000   0.00000   0.00000   0.00000  0.00000 ]
Pixel error:          err = [ 1.07759   0.96414 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

As expected, the distortion coefficient vector **kc** is now zero, and both components of the focal vector are equal (**fc(1)=fc(2)**). In practice, this model for calibration is not recommended: for one thing, it makes little sense to estimate skew without aspect ratio. In general, unless required by a specific targeted application, it is recommended to always estimate the aspect ratio in the model (it is the 'easy part'). Regarding the distortion model, people often run optimization over a subset of the distortion coefficients. For example, setting **est_dist** to **[1;0;0;0]** keeps estimating the first distortion coefficient **kc(1)** while enforcing the three others to zero. This model is also known as the second order symmetric radial distortion model. It is a very viable model, especially when using low distortion optical systems (expensive lenses), or when only a few images are used for calibration. Another very common distortion model is the 4th order symmetric radial distortion with no tangential component (**est_kc = [1;1;0;0]**). This model, used by Zhang, is justified by the fact that most lenses currently manufactured do not have imperfection in centering (for more information, visit this page). This model could have very well been used in this present example, recalling from the previous three figures that the tangential component of the distortion model is significantly smaller that the radial component.

Finally, let us run a calibration rejecting the aspect ratio **fc(2)/fc(1)**, the principal point **cc**, the distortion coefficients **kc**, and the skew coefficient **alpha_c** from the optimization estimation. For that purpose, set the four binary variables , **center_optim**, **est_dist** and **est_alpha** to the following values:

```
» est_aspect_ratio = 0;
» center_optim = 0;
» est_dist = [0;0;0;0];
» est_alpha = 0;
```

Generally, if the principal point is not estimated, the best guess for its location is the center of the image:

```
» cc = [(nx-1)/2;(ny-1)/2];
```

Then, run a new optimization by clicking on **Calibration**:

```
Aspect ratio not optimized (est_aspect_ratio = 0) -> fc(1)=fc(2). Set est_aspect_ratio to 1 for estimating aspect ratio.
Principal point not optimized (center_optim=0). Note: to set it in the middle of the image, clear variable cc, and run calibration ag
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Second order distortion not estimated (est_dist(1)=0).
     Fourth order distortion not estimated (est_dist(2)=0).
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
     Tangential distortion not estimated (est_dist(3:4)~=[1;1]).

Main calibration optimization procedure - Number of images: 25
Gradient descent iterations: 1...2...3...4...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 665.77415   665.77415 ] ± [ 2.42637   2.42637 ]
Principal point:       cc = [ 319.50000   239.50000 ] ± [ 0.00000   0.00000 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000  ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000  0.00000 ] ± [ 0.00000   0.00000   0.00000   0.00000  0.00000 ]
Pixel error:          err = [ 1.09184   0.99454 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

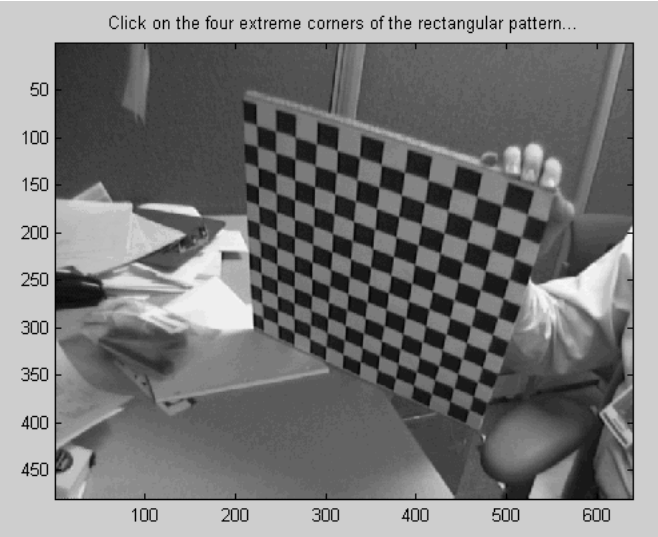
Observe that the principal point **cc** is still at the center of the image after optimization (since **center_optim=0**).

Next, load the old calibration results previously saved in **Calib_Results.mat** by clicking on **Load**:

```
Loading calibration results from Calib_Results.mat
done
```

Additional functions included in the calibration toolbox:

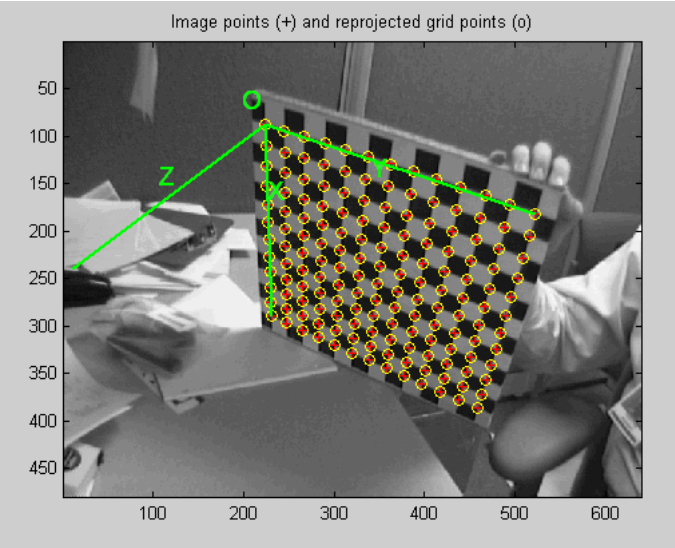
- **Computation of extrinsic parameters only:** Download an additional image of the same calibration grid: Image_ext.tif.



Notice that this image was not used in the main calibration procedure. The goal of this exercise is to compute the extrinsic parameters attached to this image given the intrinsic camera parameters previously computed.
Click on **Comp. Extrinsic** in the **Camera calibration tool**, and successively enter the image name without extension (**Image_ext**), the image type (**tif**), and extract the grid corners (following the same procedure as previously presented - remember: the first clicked point is the origin of the pattern reference frame). The extrinsic parameters (3D location of the grid in the camera reference frame) is then computed. The main matlab window should look like:

```
.                     Image15.tif           Image3.tif
..                    Image16.tif           Image4.tif
Calib_Results.mat     Image17.tif           Image5.tif
Calib_Results_old0.mat Image18.tif          Image6.tif
Calib_Results_old1.mat Image19.tif          Image7.tif
Calib_Results_old2.mat Image2.tif           Image8.tif
Image1.tif            Image20.tif           Image9.tif
Image10.tif           Image21.tif           Image_ext.tif
Image11.tif           Image22.tif           calib_data.mat
Image12.tif           Image23.tif
Image13.tif           Image24.tif
Image14.tif           Image25.tif


Computation of the extrinsic parameters from an image of a pattern
The intrinsic camera parameters are assumed to be known (previously computed)

Image name (full name without extension): Image_ext
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm') t

Extraction of the grid corners on the image
Window size for corner finder (wintx and winty):
wintx ([] = 5) = 6
winty ([] = 5) = 6
Window size = 13x13
Click on the four extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
Size dX of each square along the X direction ([]=30mm) = 30
Size dY of each square along the Y direction ([]=30mm) = 30
Corner extraction...


Extrinsic parameters:

Translation vector: Tc_ext = [ -94.617156        -184.010867     766.209711 ]
Rotation vector:   omc_ext = [ -1.451113         -1.827059       -0.179105 ]
Rotation matrix:    Rc_ext = [ -0.043583         0.875946        -0.480436
                               0.765974          0.338032        0.546825
                               0.641392          -0.344170       -0.685684 ]
Pixel error:           err = [ 0.10156   0.08703 ]
```

The extrinsic parameters are encoded in the form of a rotation matrix (**Rc_ext**) and a translation vector (**Tc_ext**). The rotation vector **omc_ext** is related to the rotation matrix (**Rc_ext**) through the Rodrigues formula: **Rc_ext = rodrigues(omc_ext)**.
Let us give the exact definition of the extrinsic parameters:
Let **P** be a point space of coordinate vector **XX = [X;Y;Z]** in the grid reference frame **(O,X,Y,Z)** shown on the following figure:



Let $\mathbf{XX_c = [X_c;Y_c;Z_c]}$ be the coordinate vector of **P** in the camera reference frame $\mathbf{(O_c,X_c,Y_c,Z_c)}$.
Then **XX** and $\mathbf{XX_c}$ are related to each other through the following rigid motion equation:

$$\mathbf{XX_c = Rc\_ext * XX + Tc\_ext}$$

In addition to the rigid motion transformation parameters, the coordinates of the grid points in the grid reference frame are also stored in the matrix **X_ext**. Observe that the variables **Rc_ext**, **Tc_ext**, **omc_ext** and **X_ext** are not automatically saved into any matlab file.

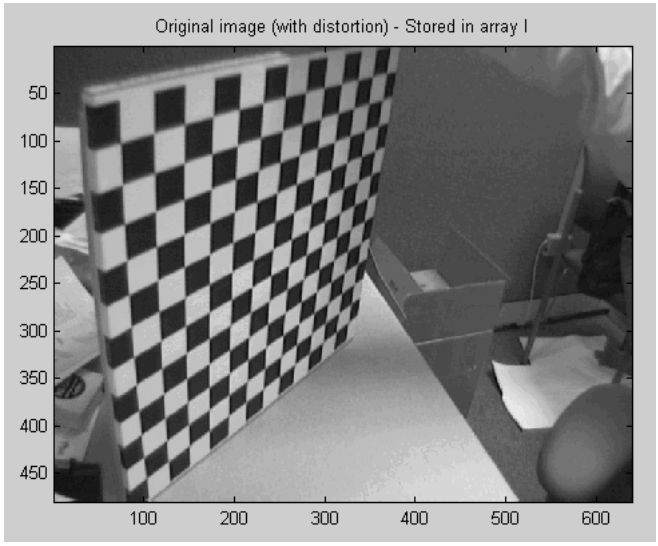
- **Undistort images:** This function helps you generate the undistorted version of one or multiple images given pre-computed intrinsic camera parameters.
As an exercise, let us undistort **Image20.tif**.
Click on **Undistort image** in the **Camera calibration tool**.

```
Program that undistorts images
The intrinsic camera parameters are assumed to be known (previously computed)

Do you want to undistort all the calibration images ([],0) or a new image (1)?
```

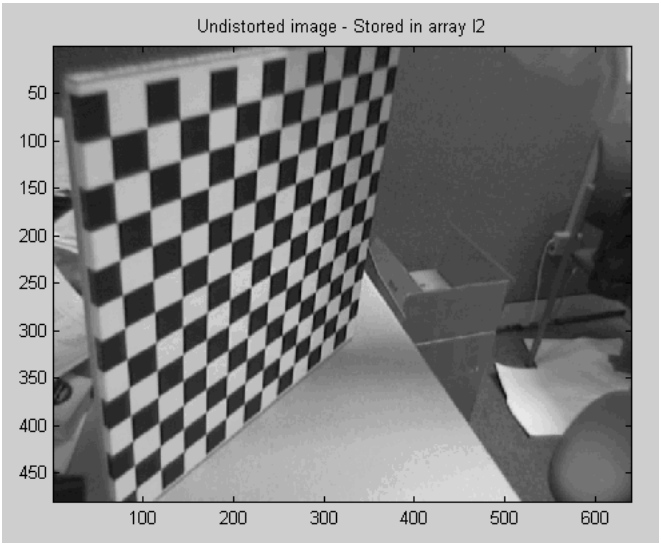
Enter 1 to select an individual image, and successively enter the image name without extension (**Image20**), the image type (**tif**). The main matlab window should look like this:

```
Program that undistorts images
The intrinsic camera parameters are assumed to be known (previously computed)

Do you want to undistort all the calibration images ([],0) or a new image (1)? 1

.                     Image15.tif           Image3.tif
..                    Image16.tif           Image4.tif
Calib_Results.mat     Image17.tif           Image5.tif
Calib_Results_old0.mat Image18.tif          Image6.tif
Calib_Results_old1.mat Image19.tif          Image7.tif
Calib_Results_old2.mat Image2.tif           Image8.tif
Image1.tif            Image20.tif           Image9.tif
Image10.tif           Image21.tif           Image_ext.tif
Image11.tif           Image22.tif           calib_data.mat
Image12.tif           Image23.tif
Image13.tif           Image24.tif
Image14.tif           Image25.tif


Image name (full name without extension): Image20
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'p'='pgm', 'j'='jpg', 'm'='ppm') t
Computing the undistorted image...done
Saving undistorted image under Image20_rect.tif...done
```

The initial image is stored in the matrix **I**, and displayed in figure 2:



The undistorted image is stored in the matrix **I2**, and displayed in figure 3:



The new undistorted image (**I2**) is also saved on disk under **Image20_rect.tif**.

Let us now undistort the complete set of calibration images. Click on **Undistort image**, and enter an empty argument to the first question. All the calibration images are then undistorted and saved onto disk under **Image_rect1.tif**, **Image_rect2.tif**, ..., **Image_rect25.tif**:

```
Program that undistorts images
The intrinsic camera parameters are assumed to be known (previously computed)

Do you want to undistort all the calibration images ([],0) or a new image (1)? 0
Saving undistorted image under Image_rect1.tif...
Saving undistorted image under Image_rect2.tif...
Saving undistorted image under Image_rect3.tif...
Saving undistorted image under Image_rect4.tif...
Saving undistorted image under Image_rect5.tif...
Saving undistorted image under Image_rect6.tif...
Saving undistorted image under Image_rect7.tif...
Saving undistorted image under Image_rect8.tif...
Saving undistorted image under Image_rect9.tif...
Saving undistorted image under Image_rect10.tif...
Saving undistorted image under Image_rect11.tif...
Saving undistorted image under Image_rect12.tif...
Saving undistorted image under Image_rect13.tif...
Saving undistorted image under Image_rect14.tif...
Saving undistorted image under Image_rect15.tif...
Saving undistorted image under Image_rect16.tif...
Saving undistorted image under Image_rect17.tif...
Saving undistorted image under Image_rect18.tif...
Saving undistorted image under Image_rect19.tif...
Saving undistorted image under Image_rect20.tif...
Saving undistorted image under Image_rect21.tif...
Saving undistorted image under Image_rect22.tif...
Saving undistorted image under Image_rect23.tif...
Saving undistorted image under Image_rect24.tif...
Saving undistorted image under Image_rect25.tif...
done
```

- **Export calibration data to other formats (Willson-Heikkil&#65533; and Zhang):** This function lets you export the calibration data (extracted image corners + associated 3D world coordinates) to Willson-Heikkil&#65533; or Zhang formats. This may be useful for comparison purposes (if you want to run other people calibration engines on the same data). This function may be used just after the corner extraction phase. Click on **Export calib data** in the main toolbox window.

```
Tool that exports calibration data to Willson-Heikkila or Zhang formats
Two possible formats of export: 0=Willson and Heikkila, 1=Zhang
Format of export (enter 0 or 1):
```

Enter **0** to select the data format used by Willson and Heikkil&#65533;, and enter the basename of the data files (**shot**). The calibration data of each image is then saved to individual files **shot1**, **shot2**,...,**shot25**:

```
Tool that exports calibration data to Willson-Heikkila or Zhang formats
Two possible formats of export: 0=Willson and Heikkila, 1=Zhang
Format of export (enter 0 or 1): 0

Export of calibration data to text files (Willson and Heikkila's format)
File basename: shot
Exporting calibration data (3D world + 2D image coordinates) of image 1 to file shot1...
Exporting calibration data (3D world + 2D image coordinates) of image 2 to file shot2...
Exporting calibration data (3D world + 2D image coordinates) of image 3 to file shot3...
Exporting calibration data (3D world + 2D image coordinates) of image 4 to file shot4...
Exporting calibration data (3D world + 2D image coordinates) of image 5 to file shot5...
Exporting calibration data (3D world + 2D image coordinates) of image 6 to file shot6...
Exporting calibration data (3D world + 2D image coordinates) of image 7 to file shot7...
Exporting calibration data (3D world + 2D image coordinates) of image 8 to file shot8...
Exporting calibration data (3D world + 2D image coordinates) of image 9 to file shot9...
Exporting calibration data (3D world + 2D image coordinates) of image 10 to file shot10...
Exporting calibration data (3D world + 2D image coordinates) of image 11 to file shot11...
Exporting calibration data (3D world + 2D image coordinates) of image 12 to file shot12...
Exporting calibration data (3D world + 2D image coordinates) of image 13 to file shot13...
Exporting calibration data (3D world + 2D image coordinates) of image 14 to file shot14...
Exporting calibration data (3D world + 2D image coordinates) of image 15 to file shot15...
Exporting calibration data (3D world + 2D image coordinates) of image 16 to file shot16...
Exporting calibration data (3D world + 2D image coordinates) of image 17 to file shot17...
Exporting calibration data (3D world + 2D image coordinates) of image 18 to file shot18...
Exporting calibration data (3D world + 2D image coordinates) of image 19 to file shot19...
Exporting calibration data (3D world + 2D image coordinates) of image 20 to file shot20...
Exporting calibration data (3D world + 2D image coordinates) of image 21 to file shot21...
Exporting calibration data (3D world + 2D image coordinates) of image 22 to file shot22...
Exporting calibration data (3D world + 2D image coordinates) of image 23 to file shot23...
Exporting calibration data (3D world + 2D image coordinates) of image 24 to file shot24...
Exporting calibration data (3D world + 2D image coordinates) of image 25 to file shot25...
done
```

Let us now export the data under Zhang's format. Click on **Export calib data**, and enter **1** to select that new export format. Enter then two file basenames: one for the 3D rig coordinates (**Model**) and one for the image coordinates (**data**). The program created then a set of text files (**Model1.txt**, **data1.txt**,...,**Model25.txt**, **data25.txt**) that can be read by Zhang's code. After export, your matlab window should look like:

```
Tool that exports calibration data to Willson-Heikkila or Zhang formats
Two possible formats of export: 0=Willson and Heikkila, 1=Zhang
Format of export (enter 0 or 1): 1

Export of calibration data to text files (Zhang's format)
File basename for the 3D world coordinates: Model
File basename for the 2D image coordinates: data
Exporting calibration data of image 1 to files Model1.txt and data1.txt...
Exporting calibration data of image 2 to files Model2.txt and data2.txt...
Exporting calibration data of image 3 to files Model3.txt and data3.txt...
Exporting calibration data of image 4 to files Model4.txt and data4.txt...
Exporting calibration data of image 5 to files Model5.txt and data5.txt...
Exporting calibration data of image 6 to files Model6.txt and data6.txt...
Exporting calibration data of image 7 to files Model7.txt and data7.txt...
Exporting calibration data of image 8 to files Model8.txt and data8.txt...
Exporting calibration data of image 9 to files Model9.txt and data9.txt...
Exporting calibration data of image 10 to files Model10.txt and data10.txt...
Exporting calibration data of image 11 to files Model11.txt and data11.txt...
Exporting calibration data of image 12 to files Model12.txt and data12.txt...
Exporting calibration data of image 13 to files Model13.txt and data13.txt...
Exporting calibration data of image 14 to files Model14.txt and data14.txt...
Exporting calibration data of image 15 to files Model15.txt and data15.txt...
Exporting calibration data of image 16 to files Model16.txt and data16.txt...
Exporting calibration data of image 17 to files Model17.txt and data17.txt...
Exporting calibration data of image 18 to files Model18.txt and data18.txt...
Exporting calibration data of image 19 to files Model19.txt and data19.txt...
Exporting calibration data of image 20 to files Model20.txt and data20.txt...
Exporting calibration data of image 21 to files Model21.txt and data21.txt...
Exporting calibration data of image 22 to files Model22.txt and data22.txt...
Exporting calibration data of image 23 to files Model23.txt and data23.txt...
Exporting calibration data of image 24 to files Model24.txt and data24.txt...
Exporting calibration data of image 25 to files Model25.txt and data25.txt...
done
```

[Back to main calibration page]