Omnidirectional Camera Calibration Toolbox for Matlab

OcamCalib Toolbox

Click here to download the OcamCalib Toolbox

Author:
Davide Scaramuzza

PhD Student at
Autonomous Systems Laboratory
Swiss Federal Institute of Technology Lausanne (EPFL), Switzerland
Email:
davide.scaramuzza@epfl.ch

Last update: May 12, 2006

Index of the tutorial:

1. Introduction to the Toolbox

The **OcamCalib Toolbox for Matlab** allows the user to calibrate any central omnidirectional camera, that is, any panoramic camera having a single effective viewpoint [3]. The Toolbox implements the procedure described in the paper [1] and further refined in [2]. A detailed introduction to this model is at the end of this Tutorial.
The Toolbox permits easy and practical calibration of the omnidirectional camera through two steps. At first, it requires the user to take a few pictures of a checkerboard shown at

different positions and orientations. Then, the user is asked to click manually on the corner points. After these two steps, calibration is completely automatically performed.
After calibration, the Toolbox provides the relation between a given pixel point and the 3D vector emanating from the single effective view point. This relation clearly depends on the mirror shape and on the intrinsic parameters of the camera. The novel aspects of the OcamCalib Toolbox with respect to other implementations are the following:

最后得到的是什么？

1. The Toolbox does not require a priori knowledge about the mirror shape.
2. It does not require calibrating separately the perspective camera: the system camera+mirror is treated as a unique compact system.
3. The detection of the image center is performed automatically. It does not require the visibility of the circular external boundary of the mirror. Unlike other implementations, which require seeing the external boundary of the mirror to determine the center of the panoramic picture, the OcamCalib Toolbox automatically identifies the center without any user interaction.

The calibration performed by the OcamCalib Toolbox is based on the following hypotheses:

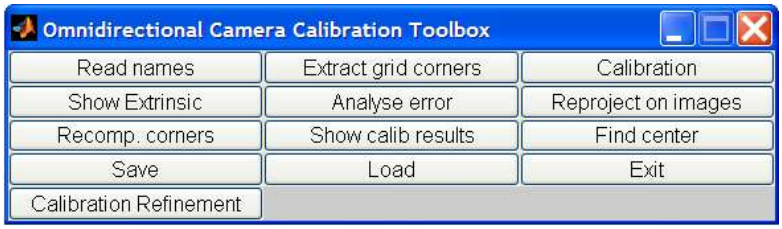
1. The system camera-mirror possesses a single effective viewpoint (see here for a definition), or also a “quasi” single viewpoint. In fact, the Toolbox is able to provide an optimal solution even when the “single view point property” is not perfectly verified (as for spherical mirrors). The Toolbox showed to give good calibration results even in the latter case.

2. Software requirements

The **OcamCalib Toolbox** for Matlab has been successfully tested under *Matlab, version 6.5 and 7.0 for Windows*.
The **Calibration Refinement** routine requires the *Matlab Optimization Toolbox*, in particular the function `lsqnonlin`, which you should have by default.

3. Download, install and run the Toolbox

You can download the OcamCalib Toolbox from here:
http://asl.epfl.ch/~scaramuz/OCamCalibration/Scaramuzza_OCamCalib.zip
Unzip the file, run Matlab, and type **`ocam_calib`**.



4. Load images

The first step to calibrate your omnidirectional camera consists in loading the images of a checkerboard shown at different positions and orientations. In the following figure you can see a few images of the checkerboard I will use in this tutorial as an example.

In order to obtain good calibration results, I suggest the following:

标定图像的一些配置，利于标定

1. Approach the checkerboard to the mirror as much as you can. Make sure that every corner of the checkerboard is visible in each image.
2. Take pictures of the checkerboard from all **around the mirror**. By doing this, you allow calibration to compensate for possible misalignments between the camera and mirrors axes. The second and most important reason for doing this is that it helps the automatic detection of the center of the omnidirectional image.

Ok… if your images are now ready, you can start loading them and calibrating you camera! Before loading the images, make sure that they are in the same folder of the toolbox files. Then, click on the button **Read names**. You will see a message on the Matlab shell:

```
Basename camera calibration images (without number nor suffix):
```

This message asks you to tape the basename of your image files, without the file format. For example, the images that you will find in the OcamCalib Toolbox are of the type: **`VMRImage0.gif, VMRImage1.gif … VMRImage9.gif`**. That means that the basename of the images is **`VMRImage`**. Thus, type:

```
Basename camera calibration images (without number nor suffix):
VMRImage
```

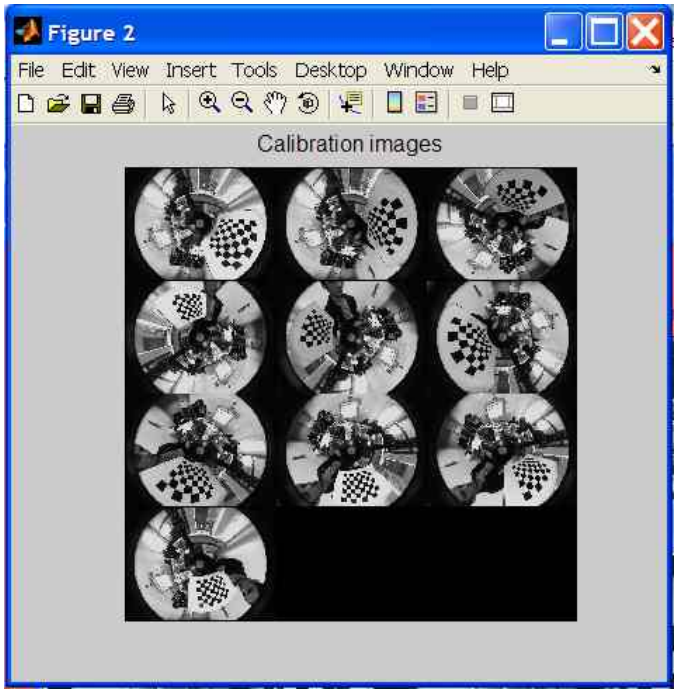
Then, it will ask you to type the image format. So type the letter associated to your format:

```
Image format: ([]='r'='ras', 'b'='bmp', 't'='tif', 'g'='gif',
'p'='pgm', 'j'='jpg', 'm'='ppm') >>  g
```

In our case, the image format is “gif”, so you will need to type **`g`**.
At this point, the Toolbox will load all images having that basename:

```
Loading image 1...2...3...4...5...6...7...8...9...10...
Done
```

At the end, the Toolbox will show the thumbnails of your calibration images, something like this:



If everything was all right.. then you can pass to the next step!

5. Extraction of grid corners

The extraction of grid corners is the most important phase for calibration, as the calibration results depend on the position of the corners of the checkerboard in each image. In this phase, the OcamCalib Toolbox will ask you to click in the corner points of each image of the checkerboard.
To facilitate the corner extraction by clicking, the Toolbox takes advantage of a corner detector to search for and interpolate the best position of the grid corner around the point you clicked on.
If you are ready and patient to start the extraction of the grid points, click on the button **Extract grid corners**. The Toolbox will ask if you want to process all images or just a fews:

```
Extraction of the grid corners on the images

Type the images you want to process (e.g. [1 2 3], [] = all images)
=
```

Type ENTER if you want to process every image, or type the vector containing the numbers of images you want to process. In our tutorial we want to process all the images, so we have just to press ENTER.
The next message is:

```
Number of squares along the X direction ([]=10) =
```

Type the number of squares present along the X direction; say the vertical direction in the reference frame of the checkerboard. In our case, for instance, we want to use 5 filled squares along the vertical direction and 6 squares along Y (say the horizontal direction). So type:

```
Number of squares along the X direction ([]=10) =  5
Number of squares along the Y direction ([]=10) =  6
```

Now type respectively the size of the square along the X and Y directions. In our images this is 30 mm.

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

If you just press ENTER, the Toolbox loads the default parameters (`[]=30mm`) .

By the next message, the Toolbox will ask the position (rows, columns) of the center of the omnidirectional image. Because the OcamCalib Toolbox is able to automatically determine the location of the center, you can just press ENTER. In this case, the Toolbox will choose as a center the real center of the image (that is, height/2, width/2). You do not have to care about this because, by using the button **Find center** after calibration, the Toolbox will search for the optimum center of the omnidirectional image. However, if you are sure about the location of the center, then just type its (X,Y) position.

```
X coordinate (along height) of the omnidirectional image center =
([]=384) =
Y coordinate (along width) of the omnidirectional image center =
([]=512) =
```

The next message is:

```
You will be soon required to click on ALL corner points.
Do you want your clicking to be assisted by a corner detector ? (
[] = yes, other = no )
```

By this message, the Toolbox informs that you will be soon asked to click on every grid corner, and asks if you want the click to be assisted by a corner detector. Just press ENTER if you want, or press another number if you do not. I suggest using the corner detector if the edges of the checkerboard are well defined. In this case, the corner detector will refine the position of the grid point you clicked on, that is, it will search for and

interpolate the best location of the grid corner around the point you clicked on. Otherwise, you may want to choose the other option, thus, the position of the point you clicked on is taken as it is.

If you answered yes, the Toolbox will ask to type the size of the window of the corner detector. The window is the size `(2*winx+1)*(2*winy+1)`

```
Window size for corner finder (wintx and winty):
wintx ([] = 8) =
winty ([] = 8) =
```

Usually, the values given as default should work fine, otherwise, (if for instance the location chosen by the corner detector is too far from the point you clicked on) you can try to choose a smaller value. Conversely, if the resolution of your pictures is very high (up to 5 Mega pixels!) you may want to choose bigger values.

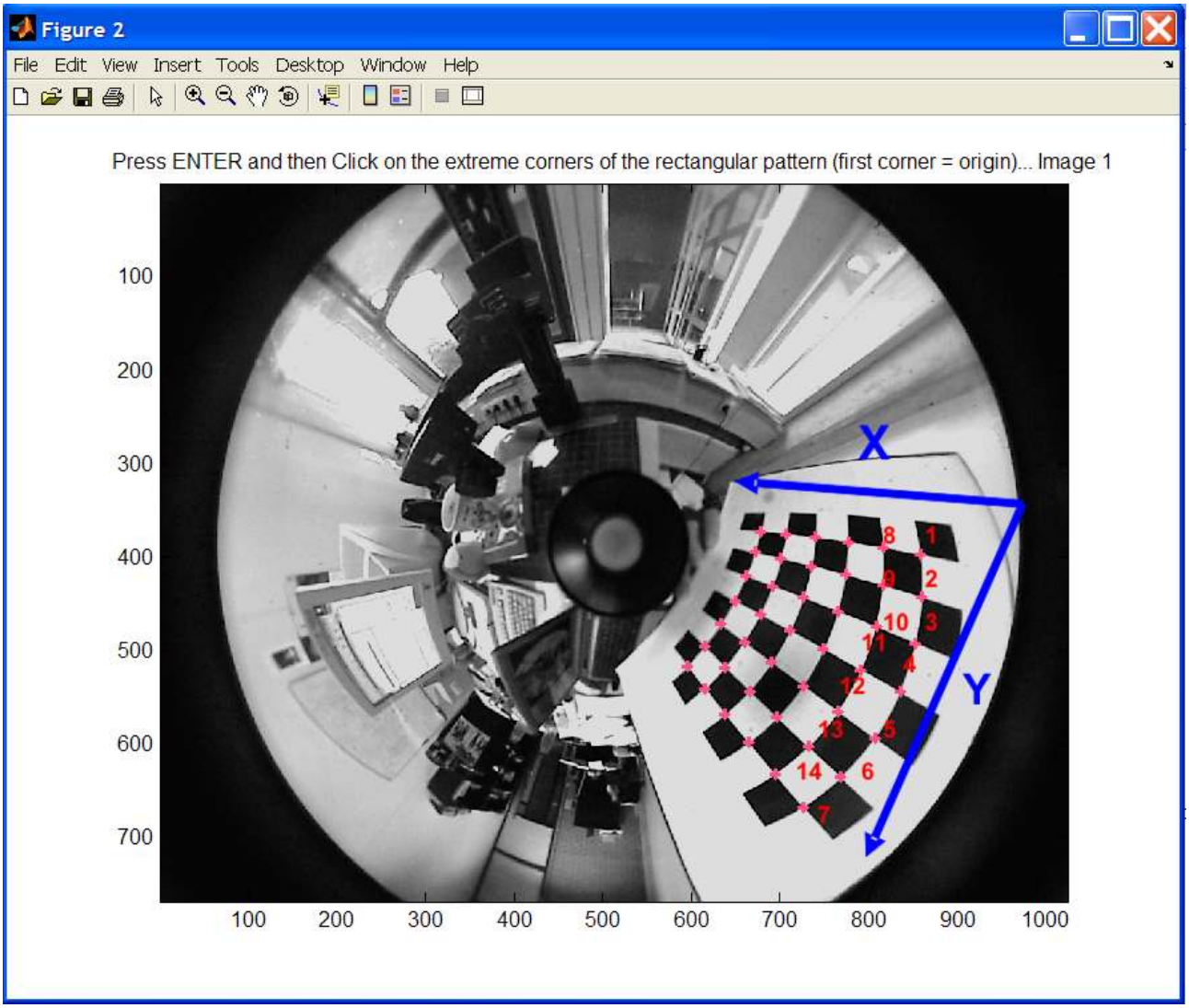
Coming back to our example, accept the suggested values by pressing ENTER. You will have something like this:

```
Window size for corner finder (wintx and winty):
wintx ([] = 8) =
winty ([] = 8) =

Window size = 17x17

Processing image 1...
Using (wintx,winty)=(8,8) - Window size = 17x17
Press ENTER and then Click on the extreme corners of the rectangular complete pattern (the first clicked corner is the origin)...
```

This is the last but most important step before calibration. In fact, you are now asked to click on every grid point. Unlike other calibration tools, which ask only to click of 4 points, here you are required to click on every corner point. This is due to the fact that the OcamCalib Toolbox does not use any prior knowledge about the mirror shape, and so, the position of all corner points cannot be inferred from a few of points alone. Anyway, this just requires a little bit of patience more. Moreover, as we say in Italy, "*who does not have patience, has nothing*"! Furthermore, the quality of the result obtained justifies the wasted time! But let's go on…

为啥2.0版本可以自动提取

Before start clicking on the grid corners, you are allowed to zoom into the region of the image, which contains the checkerboard. When you have zoomed in, press ENTER. By doing so, the shape of the cursor changes into a square, meaning that you are in the click mode.
So, start clicking on every corner point, remembering that the first point identifies the origin of the X-Y axes of the reference frame of the checkerboard (point number 1 in the next figure). The clicking has to be done moving along the Y direction, following the ordering shown in the figure. The grid corners are highlighted by the red crosses, while the order of the click is given by the numbers (see figure below).



ATTENCTION: In processing the remaining images, be careful to preserve the same correspondences of clicked points. What I mean is that, for instance, the first selected point of image 1 has to be the first selected point of image 2, 3, and so on. So, make sure to preserve the point correspondences. This makes sure that the reference axes of every checkerboard maintain the same orientation.
The grid corner extraction will continue until you process all set of images you selected at the beginning. In this tutorial, for instance, we chose to process all images.

If you finished the corner extraction, and you didn't get any error, then you can finally pass to the calibration phase!

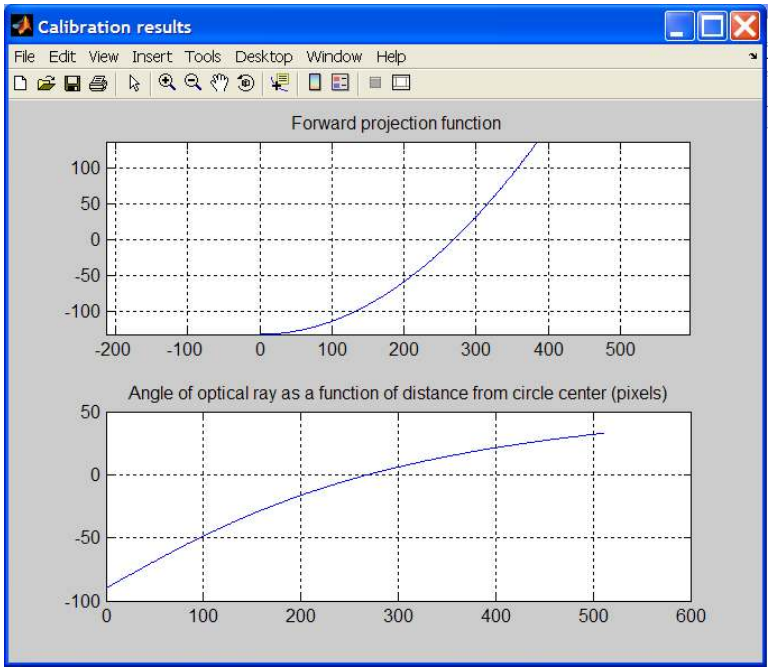
6. Calibration

If you got up to here, I assume you have loaded the images and extracted the grid corners. So, you are finally ready to calibrate you omnidirectional camera.
To do this, click on the button **Calibration**. You will receive the following message:

```
Degree of polynomial expansion ([]=4) =
```

If you read the paper in [1], which describes the calibration procedure, you know what this parameter is.. if you did not, actually this parameter permits you to choose the maximum order of the polynomial, which approximates the function that back projects every pixel point into the 3D space. Several experiments on different camera models showed that a polynomial order=4 gives the best results. If you are not satisfied with the results obtained,

try to decrease or increase the polynomial order. In this tutorial we set the value to 4. Once you have chosen the polynomial order, the calibration is performed instantaneously because a least square linear minimization method is used.
At the end of calibration, the Toolbox displays the following graph, which shows the plot of function ***F***, and the plot of angle ***THETA*** of the corresponding 3D vector with respect to the horizon.



7. Calibration results

You also obtain the calibration results for the linear method:

```
Average reprojection error computed for each chessboard [pixels]:

 0.44 ± 0.27
 0.38 ± 0.24
 0.39 ± 0.25
 0.42 ± 0.23
 0.29 ± 0.17
 0.33 ± 0.15
 0.33 ± 0.19
 0.47 ± 0.23
 0.39 ± 0.29
 0.36 ± 0.20

 Average error [pixels]

 0.381643

 Sum of squared errors

 83.130449

ss =

  1.0e+002 *

  -1.30607676451844
                   0
   0.00001734953635
```

```
   0.00000000887178
  -0.00000000001748

>>
```

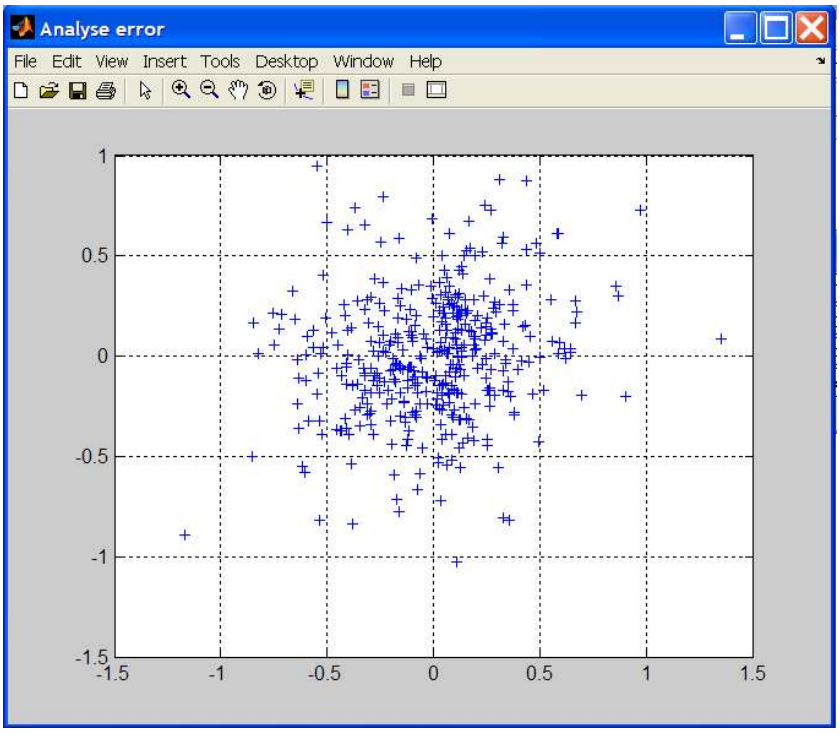
The average error is the mean of the reprojection error computed over all checkerboards, while “Sum of squared errors” is obviously the sum if squared reprojection errors.
The calibration parameters are the variable **ss**. This variable contains the polynomial coefficients of function ***F***.
I remember for ***F*** the following form:

$$F = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + a_4\rho^4$$

Where, for instance, I used a 4th order polynomial, and ***ρ*** is the distance from the center of the omnidirectional image, measured in pixels.
In **ss** the coefficients are stored from the minimum to the maximum order, that is, **ss**=[a0, a1, a2…].

8. Analyse error

If now you click on the button **Analyìse error** you can see the distribution of the reprojection error of each point for all the checkerboards.



9. Detection of image center

If during the **grid corner extraction** you didn’t set the correct values for the center of the omnidirectional image, then you can use the automatic detection of the center. For doing it, just click on the button **Find center**, and OcamCalib Toolbox will start an iterative method for computing the image center, which minimizes the reprojection error of all grid points. The automatic center detection takes only a few seconds.

```
Iteration 1...2...3...4...5...6...7...8...9...
```

At the end, the Toolbox recomputes all calibration parameters for the new position of the center, and outputs the new coordinates of the center:

Output of the **Find center** tool after iteration:

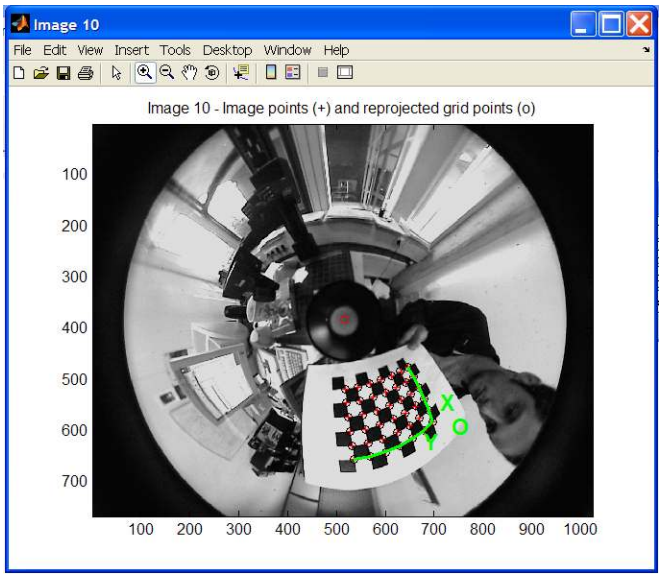
```
 0.44 ± 0.28
 0.37 ± 0.25
 0.38 ± 0.24
 0.42 ± 0.21
 0.29 ± 0.18
 0.32 ± 0.14
 0.33 ± 0.18
 0.46 ± 0.22
 0.40 ± 0.31
 0.36 ± 0.20

 Average error [pixels]

 0.377502

 Sum of squared errors

 81.896493

xc =

    3.832866677912270e+002


yc =

    5.163646215408636e+002

>>
```

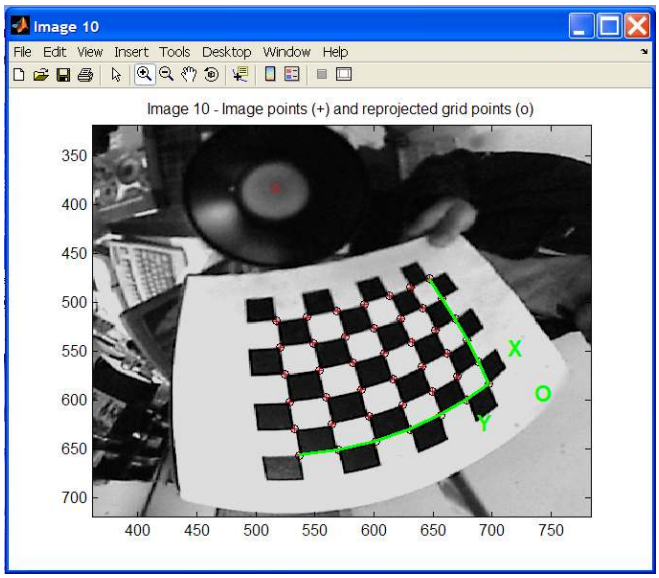
If at any time you would like to modify the coordinates of the center, you can simply modify the value of the variables **xc** and **yc**, which respectively contain row and column of the center location.

ATTENCTION!!! Use the **Find center** tool always **BEFORE** using the **Calibration Refinement**. In fact, the automatic detection of center is done by iteratively applying a linear estimation method, which is suboptimum. So, once you estimate the image center, then you can run **Calibration Refinement**, which refines both all calibration parameters and the position of the center using a non-linear method.

10. Reproject on images

By clicking on the button **Reproject on images**, the Toolbox will reproject the all grid corners according to the new calibration parameters just estimated.





In the left figure above, you can see the center of the image, indicated by the red round, which has been computed using the automatic center detection.
While, on the right figure, you can see a detail of the checkerboard, with all corner grid highlighted, and the X-Y axes of the reference frame.
The red crosses are the grid corners you clicked on, while the rounds are the grid corners reprojected onto the image, after calibration.

11. Calibration Refinement

By clicking on the button **Calibration Refinement**, the Toolbox will start the non-linear refinement of the calibration parameters, by using the Levenberg-Marquadt algorithm. The non-linear refinement over parameters **ss**, **xc** and **yc**. The optimization is performed attempting to minimize the sum squared reprojection errors.
The Calibration refinement is done using the Matlab Optimization Toolbox, and, in particular, it requires function `lsqnonlin`, which you should have by default.
The non-linear refinement is done in two steps. At first it starts refining the extrinsic camera parameters, that is, rotation and translation matrices of each checkerboard with respect to the camera. Then, it starts refining the camera intrinsic parameters (i.e. **ss**, **xc** and **yc**).

Once you click on the button, **Calibration Refinement**, the Toolbox asks if you are sure to want to start the non-linear refinement, and inform you tjat the procedure can take several seconds. Just press ENTER, and wait the results!

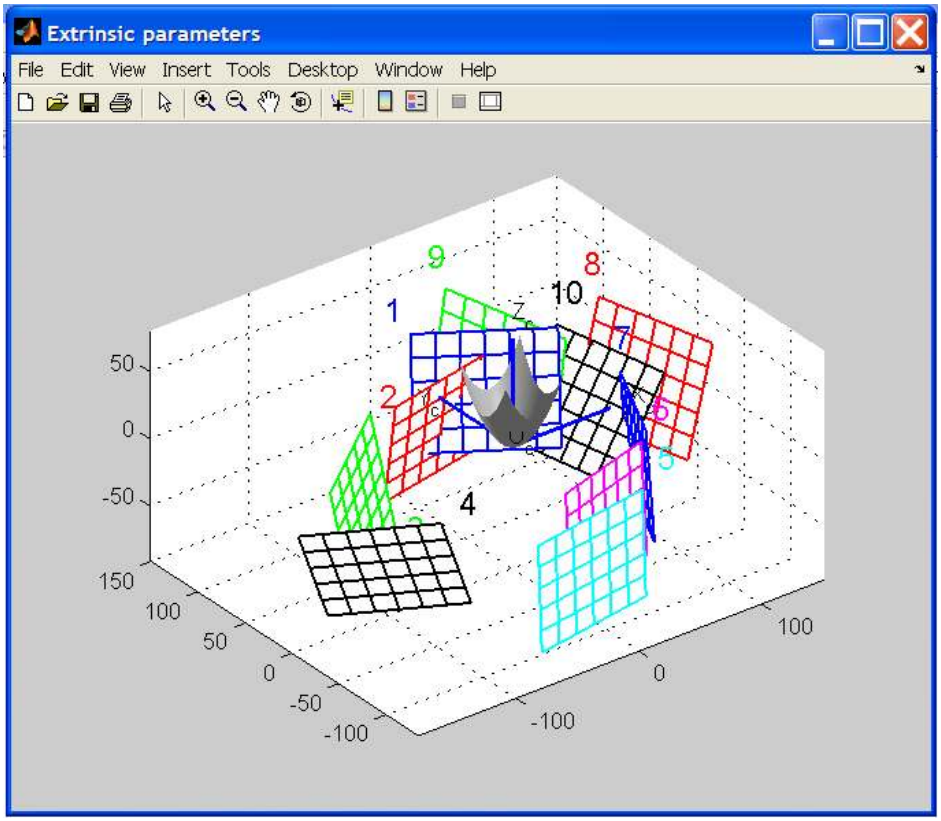
```
This function refines calibration parameters (both EXTRINSIC and
INTRINSIC)
by using a non linear minimization method
Because of the computations involved this refinement can take some
seconds
Press ENTER to continue OR Crtl-C if you do not want

Starting refinement of EXTRINSIC parameters

Optimization terminated: search direction less than TolX.
Chessboard pose 1 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 2 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 3 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 4 optimized
Optimization terminated: search direction less than TolX.
```

```
Chessboard pose 5 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 6 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 7 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 8 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 9 optimized
Optimization terminated: search direction less than TolX.
Chessboard pose 10 optimized

Starting refinement of INTRINSIC parameters
```

As you will see, the last step, which concerns the refinement of the INTRINSIC parameters of the camera, is the phase requiring most of the time, but in general it should always take several seconds.
Once the non-linear refinement is terminated, you can again you use buttons **Analyse error** and **Show calib results** to display the reprojection error or the calibration results.

12. Show Extrinsic

By clicking on the button **Show Extrinsic**, the Toolbox will display the position of every checkerboard with respect to the reference frame of the omnidirectional camera.



13. Recompute corners

By using this tool, the Toolbox will perform the automatic detection of every grid corner around the reprojected points. This function is very useful if during the extraction of grid corners you did some mistakes, or the automatic corner detector did. By using the button **Recomp. Corners**, the Toolbox will attempts to recompute the positions of every corner point you clicked on, by using the reprojected grid as initial guess locations for the corners.

14. Load and save calibration results

To load and save your calibration results, just click on the corresponding buttons.
The calibration results will be save under the name **Omni_Calib_Results.mat**.

15. Workspace variables

The most important variables of the Matlab workspace, which are used by the OcamCalib Toolbox are the following:

`ss`	Contains the polynomial coefficients of function ***F*** (see here for explanation)
`xc` `yc`	Are the **row**, **column** coordinates of the center of the omnidirectional image
`c, d, e`	Affine transformation parameters (see here for explanation)
`width, height`	Width and height of the images under process

`ocam_model = { ss`
`xc`
`yc`
`c`
`d`
`e`
`width`
`height`
`}`
	Is a structure having all variable so far as members

`Xp_abs` `Yp_abs`	Are **Nx1xM** matrices, which contain the **row** and **column** coordinates of the grid corners (i.e. the image coordinates of the points you clicked on during the extraction of the grid corners). **N** is the number of points of each checkerboard. **M** is the number of images you processed.
`Xt` `Yt`	Is a **Nx1** vector containing the X-Y coordinates (in mm) of the grid corners, in the reference frame of the checkerboard
`ima_proc`	Contains the number of images currently used for calibration (this number is initialized during the grid corner extraction
`RRfin`	Contains the extrinsic parameters of every checkerboard with the respect to the camera reference frame. RRfin is a **3x3xM** matrix, where **M** correspond to the respective image number. The first two columns of RRfin identify the ROTATION matrix (which is given by the column vectors Rx, Ry, Rz can be obtained as cross product between Rx and Ry). The third column of RRfin identify the TRANSLATION vector of the checkerboard (i.e. [tx;ty;tz]
`err`	Is the mean error of reprojection of each checkerboard, which has been processed.

16. Useful functions to use after calibration

Once you finish to calibrate your camera, you can use two functions to respectively project a 3D point onto the image, and, vice versa, to back project a pixel point into the space.
This two functions are:

`m = world2cam(M, ocam_model)`	Projects a 3D point on to the image and returns the pixel coordinates. **M** is a **3xN** matrix containing the coordinates of the 3D points: **M=[X;Y;Z]**. **ocam_model** contains the model of the calibrated camera. **m=[rows;cols]** is a **2xN** matrix containing the returned rows and columns of the points after being

	reproject onto the image.
`M = cam2world(m, ocam_model)`	Returns the 3D coordinates of the vector emanating from the single effective viewpoint. **m=[rows;cols]** is a **2xN** matrix containing the pixel coordinates of the image Points. **ocam_model** contains the model of the calibrated camera. **M=[X;Y;Z]** is a **3xN** matrix with the coordinates of the 3D vectors. 以何处为原点？

17. References

[1] **Scaramuzza, D.**, Martinelli, A. and Siegwart, R., "**A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion**", in Proceedings of IEEE International Conference of Vision Systems (ICVS'06), New York, January 5-7, 2006. [pdf]

[2] **Scaramuzza, D.**, Martinelli, A. and Siegwart, R., "**A Toolbox for Easy Calibrating Omnidirectional Cameras**", SUBMITTED to IEEE International Conference on Intelligent Robots and Systems" (IROS 2006).

[3] S. Baker and S. Nayar, "**A theory of single-viewpoint catadioptric image formation**", International Journal of Computer Vision, 35(2), November 1999, pp. 175, 1996.

[4] B. Micusik and T. Pajdla. **"Estimation of omnidirectional camera model from epipolar geometry"**. In Proceedings. of the IEEE International Conference on Computer Vision and Pattern Recognition, CVPR'03, pp. 485.490, 2003.

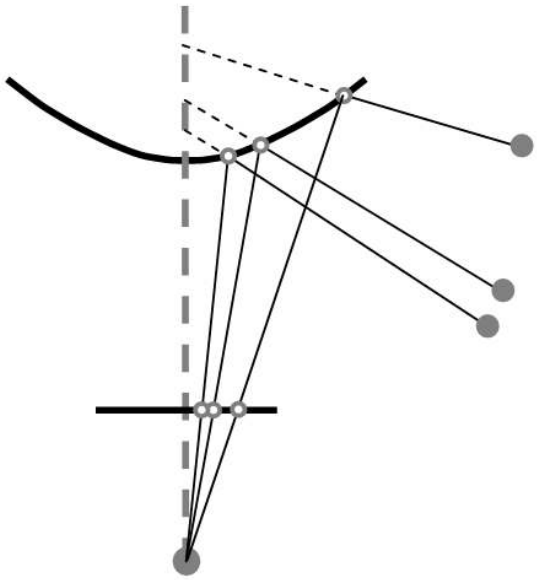
18. Acknowledgments

This work was conducted within the EU Integrated Project COGNIRON ("The Cognitive Robot Companion") and was funded by the European Commission Division FP6-IST Future and Emerging Technologies under Contract FP6-002020.
We also want thank you to Zoran Zivkovic and Olaf Booij, from Intelligent Systems Laboratory Amsterdam (University of Amsterdam), for providing the sample images included in the Toolbox.

19. Central and Non-Central Cameras

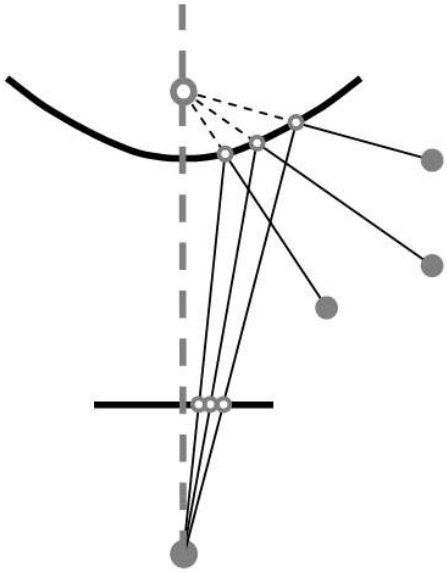
Non Central Cameras

Omnidirectional cameras are usually arranged by optimally combining mirrors and perspective cameras. A camera-mirror assembly is called non-central (i.e. non-single effective viewpoint) system when the optical rays coming from the camera and reflected by the mirror surface do not intersect into a unique point. Check the image below for a better understanding.



Central Cameras

Conversely, central cameras are systems such that the single effective viewpoint property is perfectly verified. That is, every optical ray, which is reflected by the mirror surface, intersects into a unique point, which is called **single effective viewpoint** (see image below for a better understanding). A complete definition and analysis of this kind of imaging systems is given in [3].



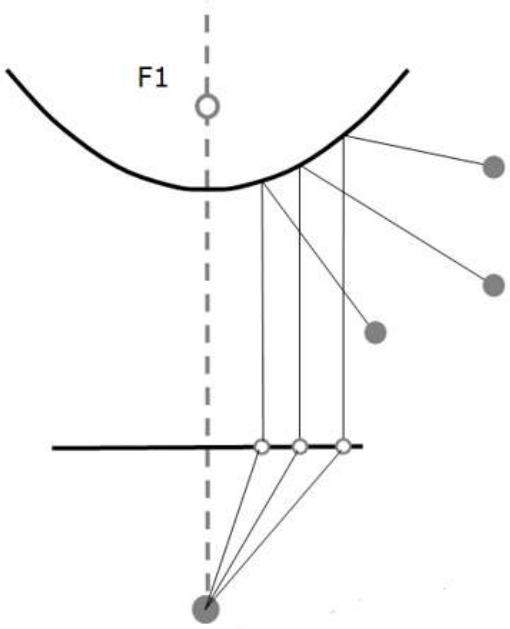
As outlined in [3], central omnidirectional cameras can be built by optimally combining a pinhole camera (perspective camera) with **hyperbolic**, **parabolic**, and **elliptical** mirrors. Recently, lens manufacturing is also providing fisheye lenses, which well approximate the single effective viewpoint property. These imaging systems require only a fisheye lens to enlarge the field of view of the camera, without requiring mirrors. The former cameras (using both camera and mirror) are called **catadioptric omnidirectional cameras**, while the latter cameras (using only a fisheye camera) are called **dioptric omnidirectional cameras**.

For a catadioptric camera to be a central system, the following arrangements have to be satisfied:

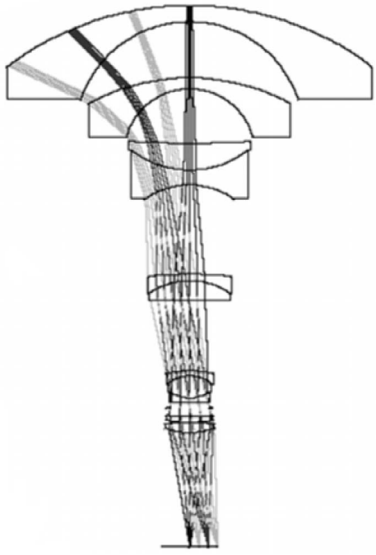
- **Camera + hyperbolic mirror**

The camera optical center (namely the center of the lens) has to coincide with the focus of the **hyperbola**. This assures the optical rays reflected by the mirror to intersect into a unique point (i.e. the internal focus of the hyperbola). For an example of camera + hyperbolic mirror see the image above.

- **Camera + parabolic mirror + orthographic lens**
  When using a **parabolic mirror** all reflected rays coming from the world into the camera are parallel to the mirror axis. This implies that a pin hole camera cannot in general be used as it is, because parallel rays do not converge towards the camera optical center. In order to provide a focused image onto the CCD plane, an orthographic lens has to be put between the camera and the parabolic mirror. Check the image below for a better understanding.



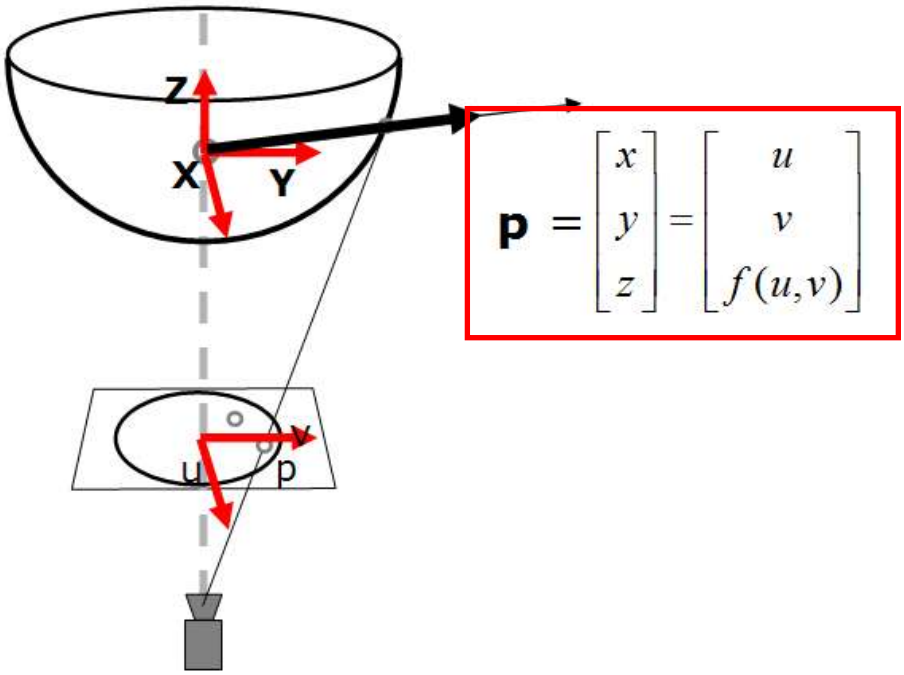
- **Camera + fisheye lens**
  An alternative method to enlarge the camera field of view without using mirrors consists in adding a fisheye lens above the camera CCD.
  A fisheye lens is a system of lenses which are able to enlarge the field of view of a camera up to 190° (see the image below).



Cameras using fisheye lenses are not in general **central systems**, but they very well approximate the single view point property.

20. Our Omnidirectional Camera Model

Calibrating an omnidirectional camera implies finding the relation between a given 2D pixel point **p** and the 3D vector **P** emanating from the mirror effective viewpoint (see figure below). In general, this process requires finding the camera intrinsic parameters and the mirror intrinsic parameters.
Our omnidirectional camera model treats the imaging system as a unique compact system; that is, it does not care if you are using a mirror or a fisheye lens in combination with your camera.



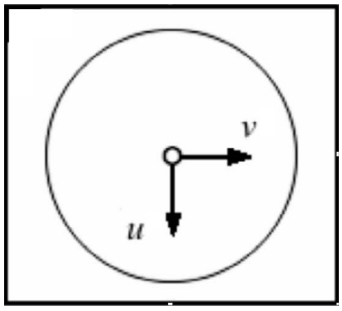
20.1 Assumptions

Our model is based on the following assumptions:

1. The mirror camera system is a central system, thus, there exist a point in the mirror where every reflected ray intersects in. This point is considered the axis origin of the camera coordinate system XYZ.
2. The camera and mirror axes are well aligned, that is, only small deviations of the rotation are considered into the model.
3. The mirror is rotationally symmetrical with respect to its axis.
4. The lens distortion of the camera is not considered in the model. Camera lens distortion has not been included because omnidirectional cameras using mirrors usually need large focal length to focalize the image on the mirror. Thus, lens distortion can be really neglected. If you are using fish-eye lenses, camera lens distortion is already integrated in the projection function ***f***.

20.2 The Omnidirectional camera model: partial model explanation

Now, I am going to explain the omnidirectional camera model.
For now, let us suppose that assumption 2 is perfectly verified, that is, camera and mirror axes are perfectly aligned. Later on, we will see how to overcome also this constraint.
Let **p** be a pixel point of your image, and (u,v) its pixel coordinates with respect to the center of the omnidirectional image (see image below). Let **P** be its corresponding 3D vector emanating from the single effective viewpoint, and (x,y,z) its coordinates with respect to the axis origin.



Because the camera and mirror axes are supposed to be perfectly aligned, observe that x and y are proportional to u and v respectively. Thus,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \alpha \cdot \begin{bmatrix} u \\ v \end{bmatrix}, \quad \alpha > 0 .$$

The function we want the calibration to estimate is the function, which maps an image point **p** into its corresponding 3D vector **P** . So we can write:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \alpha \cdot u \\ \alpha \cdot v \\ f(u,v) \end{bmatrix}$$

You would have probably observed that we can include α into the function ***f*** , and so we can equally write:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix}$$

Indeed, remember that ***P*** is not a 3D point, but a vector; hence the last simplification is allowed!
Furthermore, because the mirror is rotationally symmetric, function $f(u,v)$ depends only on the distance of a point from the image center $\rho = \sqrt{u^2 + v^2}$.
So, we can still simplify the previous equation into the following one:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ f(\rho) \end{bmatrix}, \quad \rho = \sqrt{u^2 + v^2}$$

If you got what I explained so far, what we need for calibration is just the function $f(\rho)$.
Now, let's go to introduce what this function looks like.
Our model describes function $f(\rho)$ by means of a polynomial, whose coefficients are the calibration parameters to be estimated. That is:

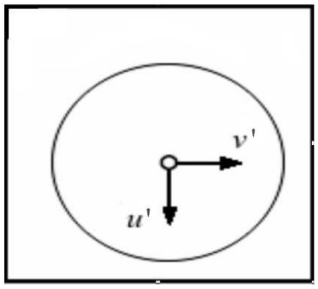
$$f(\rho) = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + a_4\rho^4 + \ldots$$

有点泰勒展开的意思？

So the parameters to estimate are: $a_0, a_1, a_2, a_3, a_4, \ldots$ Actually, the OcamCalib Toolbox asks to specify the polynomial degree you want to use. Actually, the more one augments polynomial order the more the accuracy of calibration increases. This is not true for high order polynomials. By using a lot this Toolbox I experienced that 4th order polynomials give the best calibration results.

20.3 The Omnidirectional camera model: complete model explanation

If you remember, so far we have assumed that the camera and mirror axes were perfectly aligned. Actually, because of natural errors in the camera-mirror settings, a small deviation from this hypothesis may occur. Moreover, because of the digitizing process of the camera, the pixels may not be square.
The natural consequence of these problems is that the circular external border of the mirror appears as an ellipse, as in the image below (the distortion effect in this image has been intentionally emphasized).



In order to take into account these considerations, I chose to model the misalignments errors and digitizing artefacts through an affine transformation:

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \begin{bmatrix} c & d \\ e & 1 \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} xc' \\ yc' \end{bmatrix}.$$

This equation relates the real distorted coordinates (u', v') to the ideal undistorted ones (u, v).

[Home]