## initUndistortRectifyMap()

```
void cv::initUndistortRectifyMap ( InputArray   cameraMatrix,
                                   InputArray   distCoeffs,
                                   InputArray   R,
                                   InputArray   newCameraMatrix,
                                   Size         size,
                                   int          m1type,
                                   OutputArray  map1,
                                   OutputArray  map2
                                 )
```

**Python:**

```
cv.initUndistortRectifyMap( cameraMatrix, distCoeffs, R, newCameraMatrix, size, m1type[, map1[, map2]] ) -> map1, map2
```

```
#include <opencv2/imgproc.hpp>
```

Computes the undistortion and rectification transformation map.

The function computes the joint undistortion and rectification transformation and represents the result in the form of maps for remap. The undistorted image looks like original, as if it is captured with a camera using the camera matrix =newCameraMatrix and zero distortion. In case of a monocular camera, newCameraMatrix is usually equal to cameraMatrix, or it can be computed by **getOptimalNewCameraMatrix** for a better control over scaling. In case of a stereo camera, newCameraMatrix is normally set to P1 or P2 computed by **stereoRectify** .

Also, this new camera is oriented differently in the coordinate space, according to R. That, for example, helps to align two heads of a stereo camera so that the epipolar lines on both images become horizontal and have the same y- coordinate (in case of a horizontally aligned stereo camera).

The function actually builds the maps for the inverse mapping algorithm that is used by remap. That is, for each pixel $(u, v)$ in the destination (corrected and rectified) image, the function computes the corresponding coordinates in the source image (that is, in the original image from camera). The following process is applied:

$$\begin{array}{l}
x \leftarrow (u - {c'}_x)/{f'}_x \\
y \leftarrow (v - {c'}_y)/{f'}_y \\
{[X\,Y\,W]}^T \leftarrow R^{-1}*[x\,y\,1]^T \\
x' \leftarrow X/W \\
y' \leftarrow Y/W \\
r^2 \leftarrow x'^2 + y'^2 \\
x'' \leftarrow x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2 x'^2) + s_1 r^2 + s_2 r^4\\
y'' \leftarrow y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 (r^2 + 2 y'^2) + 2 p_2 x' y' + s_3 r^2 + s_4 r^4 \\
s\vecthree{x'''}{y'''}{1} = \vecthreethree{R_{33}(\tau_x, \tau_y)}{0}{-R_{13}((\tau_x, \tau_y)}{0}{R_{33}(\tau_x, \tau_y)}{-R_{23}(\tau_x, \tau_y)}{0}{0}{1} R(\tau_x, \tau_y) \vecthree{x''}{y''}{1}\\
map_x(u,v) \leftarrow x''' f_x + c_x \\
map_y(u,v) \leftarrow y''' f_y + c_y
\end{array}$$

where $(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$ are the distortion coefficients.

In case of a stereo camera, this function is called twice: once for each camera head, after stereoRectify, which in its turn is called after **stereoCalibrate**. But if the stereo camera was not calibrated, it is still possible to compute the rectification transformations directly from the fundamental matrix using **stereoRectifyUncalibrated**. For each camera, the function computes homography H as the rectification transformation in a pixel domain, not a rotation matrix R in 3D space. R can be computed from H as

$$\texttt{R} = \texttt{cameraMatrix} ^{-1} \cdot \texttt{H} \cdot \texttt{cameraMatrix}$$

where cameraMatrix can be chosen arbitrarily.

**Parameters**

| | |
|---|---|
| **cameraMatrix** | Input camera matrix $A=\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ . |

| | |
|---|---|
| **distCoeffs** | Input vector of distortion coefficients $(k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6[, s_1, s_2, s_3, s_4[, \tau_x, \tau_y]]]])$ of 4, 5, 8, 12 or 14 elements. If the vector is NULL/empty, the zero distortion coefficients are assumed. |
| **R** | Optional rectification transformation in the object space (3x3 matrix). R1 or R2 , computed by **stereoRectify** can be passed here. If the matrix is empty, the identity transformation is assumed. In **initUndistortRectifyMap** R is assumed to be an identity matrix. |
| **newCameraMatrix** | New camera matrix $A' = \begin{bmatrix} f'_x & 0 & c'_x \\ 0 & f'_y & c'_y \\ 0 & 0 & 1 \end{bmatrix}$. |
| **size** | Undistorted image size. |
| **m1type** | Type of the first output map that can be CV_32FC1, CV_32FC2 or CV_16SC2, see **convertMaps** |
| **map1** | The first output map. |
| **map2** | The second output map. |