stereoRectify()

```
void cv::stereoRectify ( InputArray   cameraMatrix1,
                         InputArray   distCoeffs1,
                         InputArray   cameraMatrix2,
                         InputArray   distCoeffs2,
                         Size         imageSize,
                         InputArray   R,
                         InputArray   T,
                         OutputArray  R1,
                         OutputArray  R2,
                         OutputArray  P1,
                         OutputArray  P2,
                         OutputArray  Q,
                         int          flags = CALIB_ZERO_DISPARITY ,
                         double       alpha = -1 ,
                         Size         newImageSize = Size() ,
                         Rect *       validPixROI1 = 0 ,
                         Rect *       validPixROI2 = 0
                       )
```

Python:

```
cv.stereoRectify( cameraMatrix1, distCoeffs1, cameraMatrix2, distCoeffs2, imageSize, R, T[, R1[, R2[, P1[, P2[, Q[, flags[, alpha[, newImageSize]]]]]]]] ) -> R1, R2, P1, P2, Q, validPixROI validPixROI
```

```
#include <opencv2/calib3d.hpp>
```

Computes rectification transforms for each head of a calibrated stereo camera.

Parameters

cameraMatrix1 First camera intrinsic matrix.

distCoeffs1 First camera distortion parameters.

cameraMatrix2 Second camera intrinsic matrix.

distCoeffs2 Second camera distortion parameters.

imageSize Size of the image used for stereo calibration.

R Rotation matrix from the coordinate system of the first camera to the second camera, see **stereoCalibrate**.

T Translation vector from the coordinate system of the first camera to the second camera, see **stereoCalibrate**.

R1 Output 3x3 rectification transform (rotation matrix) for the first camera. This matrix brings points given in the unrectified first camera's coordinate system to points in the rectified first camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified first camera's coordinate system to the rectified first camera's coordinate system.

R2 Output 3x3 rectification transform (rotation matrix) for the second camera. This matrix brings points given in the unrectified second camera's coordinate system to points in the rectified second camera's coordinate system. In more technical terms, it performs a change of basis from the unrectified second camera's coordinate system to the rectified second camera's coordinate system.

P1 Output 3x4 projection matrix in the new (rectified) coordinate systems for the first camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified first camera's image.

P2 Output 3x4 projection matrix in the new (rectified) coordinate systems for the second camera, i.e. it projects points given in the rectified first camera coordinate system into the rectified second camera's image.

Q Output 4×4 disparity-to-depth mapping matrix (see **reprojectImageTo3D**).

flags Operation flags that may be zero or **CALIB_ZERO_DISPARITY** . If the flag is set, the function makes the principal points of each camera have the same pixel coordinates in the rectified views. And if the flag is not set, the function may still shift the images in the horizontal or vertical direction (depending on the orientation of epipolar lines) to maximize the useful image area.

alpha Free scaling parameter. If it is -1 or absent, the function performs the default scaling. Otherwise, the parameter should be between 0 and 1. alpha=0 means that the rectified images are zoomed and shifted so that only valid pixels are visible (no black areas after rectification). alpha=1 means that the rectified image is decimated and shifted so that all the pixels from the original images from the cameras are retained in the rectified images (no source image pixels are lost). Any intermediate value yields an intermediate result between those two extreme cases.

newImageSize New image resolution after rectification. The same size should be passed to **initUndistortRectifyMap** (see the stereo_calib.cpp sample in OpenCV samples directory). When (0,0) is passed (default), it is set to the original imageSize . Setting it to a larger value can help you preserve details in the original image, especially when there is a big radial distortion.

validPixROI1 Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below).

validPixROI2 Optional output rectangles inside the rectified images where all the pixels are valid. If alpha=0 , the ROIs cover the whole images. Otherwise, they are likely to be smaller (see the picture below).

The function computes the rotation matrices for each camera that (virtually) make both camera image planes the same plane. Consequently, this makes all the epipolar lines parallel and thus simplifies the dense stereo correspondence problem. The function takes the matrices computed by **stereoCalibrate** as input. As output, it provides two rotation matrices and also two projection matrices in the new coordinates. The function distinguishes the following two cases:

- **Horizontal stereo**: the first and the second camera views are shifted relative to each other mainly along the x-axis (with possible small vertical shift). In the rectified images, the corresponding epipolar lines in the left and right cameras are horizontal and have the same y-coordinate. P1 and P2 look like:

$$\mathtt{P1} = \begin{bmatrix} f & 0 & cx_1 & 0 \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathtt{P2} = \begin{bmatrix} f & 0 & cx_2 & T_x \cdot f \\ 0 & f & cy & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\mathtt{Q} = \begin{bmatrix} 1 & 0 & 0 & -cx_1 \\ 0 & 1 & 0 & -cy \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_x} & \frac{cx_1 - cx_2}{T_x} \end{bmatrix}$$

  where T_x is a horizontal shift between the cameras and $cx_1 = cx_2$ if **CALIB_ZERO_DISPARITY** is set.

- **Vertical stereo**: the first and the second camera views are shifted relative to each other mainly in the vertical direction (and probably a bit in the horizontal direction too). The epipolar lines in the rectified images are vertical and have the same x-coordinate. P1 and P2 look like:

$$\mathtt{P1} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

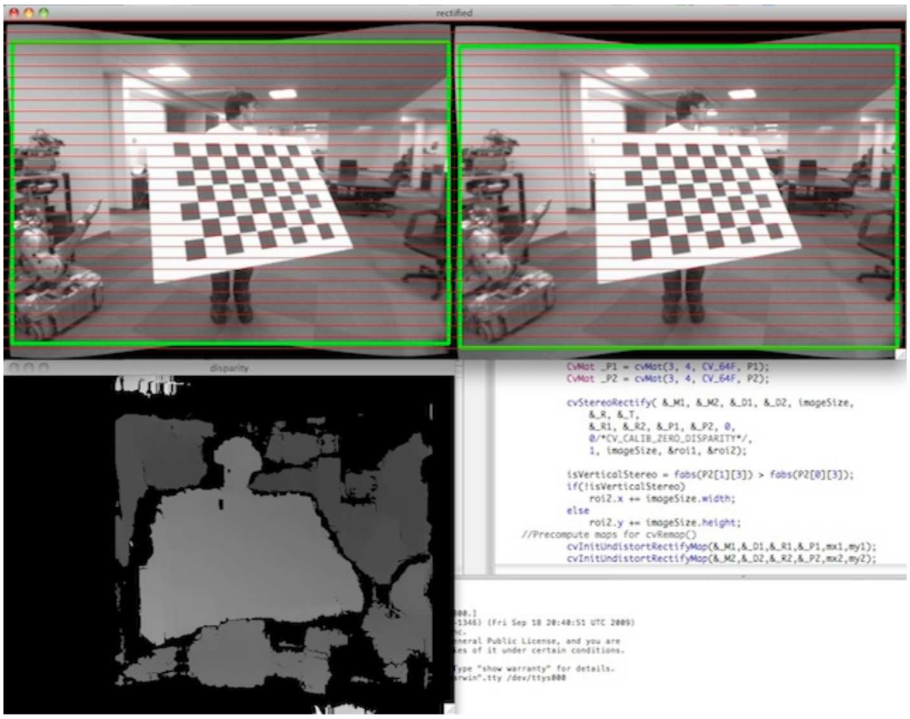
$$\mathtt{P2} = \begin{bmatrix} f & 0 & cx & 0 \\ 0 & f & cy_2 & T_y \cdot f \\ 0 & 0 & 1 & 0 \end{bmatrix},$$

$$\mathtt{Q} = \begin{bmatrix} 1 & 0 & 0 & -cx \\ 0 & 1 & 0 & -cy_1 \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{T_y} & \frac{cy_1 - cy_2}{T_y} \end{bmatrix}$$

  where T_y is a vertical shift between the cameras and $cy_1 = cy_2$ if **CALIB_ZERO_DISPARITY** is set.

As you can see, the first three columns of P1 and P2 will effectively be the new "rectified" camera matrices. The matrices, together with R1 and R2 , can then be passed to **initUndistortRectifyMap** to initialize the rectification map for each camera.

See below the screenshot from the stereo_calib.cpp sample. Some red horizontal lines pass through the corresponding image regions. This means that the images are well rectified, which is what most stereo correspondence algorithms rely on. The green rectangles are roi1 and roi2 . You see that their interiors are all valid pixels.



```
        CvMat _P1 = cvMat(3, 4, CV_64F, P1);
        CvMat _P2 = cvMat(3, 4, CV_64F, P2);

        cvStereoRectify( &_M1, &_M2, &_D1, &_D2, imageSize,
            &_R, &_T,
            &_R1, &_R2, &_P1, &_P2, 0,
            0/*CV_CALIB_ZERO_DISPARITY*/,
            1, imageSize, &roi1, &roi2);

        isVerticalStereo = fabs(P2[1][3]) > fabs(P2[0][3]);
        if(!isVerticalStereo)
            roi2.x += imageSize.width;
        else
            roi2.y += imageSize.height;
    //Precompute maps for cvRemap()
        cvInitUndistortRectifyMap(&_M1,&_D1,&_R1,&_P1,mx1,my1);
        cvInitUndistortRectifyMap(&_M2,&_D2,&_R2,&_P2,mx2,my2);
```

```
00.]
-1346) (Fri Sep 18 20:40:51 UTC 2009)
nc.
eneral Public License, and you are
ies of it under certain conditions.

Type "show warranty" for details.
arwin".tty /dev/ttys000
```

image