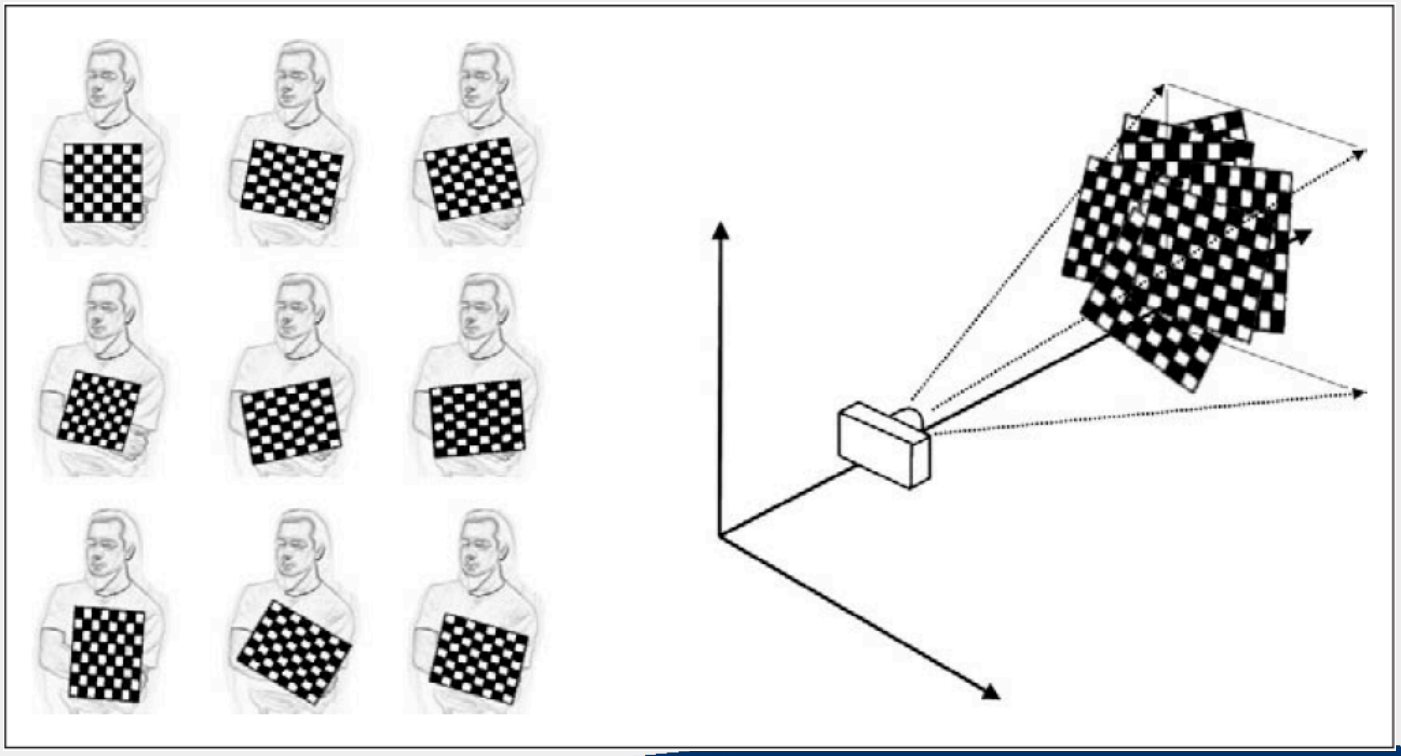
Stereo Calibration & Rectification

Calibration Function

double calibrateCamera(const vector<vector<Point3f> >& objectPoints,
const vector<vector<Point2f> >& imagePoints, Size imageSize,
Mat& cameraMatrix, Mat& distCoeffs, vector<Mat>& rvecs, vector<Mat>& tvecs,
int flags=0);



Brigham Young University ROBOT C VISION LAB

Stereo Calibration

- Using homography calibration method, we can get multiple sets of extrinsic parameters, one for each chessboard view (between the camera and chessboard).
- Unique camera intrinsic and distortion parameters are calculated through homography calibration method
- If a 3D known object is available, then solvePnP() can be used to get a set of R and T (between the camera and the calibration object) for each camera.
- Alternatively, we can obtain one set of R and T (between the camera and the calibration object) for each camera by using the direct or indirect camera calibration method.

Stereo Calibration

- Assuming that we used direct or indirect method or solvePnP() to obtain a set of unique extrinsic parameters for each camera, then

$$P_l = R_l P_w + T_l \quad and \quad P_r = R_r P_w + T_r$$

- P_w is the 3D point seen by both cameras.
- P_l is the same 3D point seen by the left camera
- P_r is the same 3D point seen by the right camera
- R_l *and* T_l *and* R_r *and* T_r define the 3D relationship between the calibration object and the left and right cameras, respectively.

$$P_l = R_l P_w + T_l \quad \text{and} \quad P_r = R_r P_w + T_r$$

$$P_l - T_l = R_l P_w \Rightarrow R_l^T (P_l - T_l) = P_w$$

$$P_r = R_r R_l^T (P_l - T_l) + T_r = (R_r R_l^T) P_l - (R_r R_l^T T_l - T_r)$$

$$= RP_l - (RT_l - T_r) = RP_l - RR^T (RT_l - T_r)$$

$$= RP_l - R(R^T RT_l - R^T T_r) = RP_l - R(T_l - R^T T_r)$$

$$= RP_l - RT = R(P_l - T)$$

The relation between P_l and P_r

$$\therefore R = R_r R_l^T \quad \text{and} \quad T = T_l - R^T T_r$$

If the left camera is chosen as the reference frame,

$$R = R_r R_l^T \quad \text{and} \quad T = T_r - RT_l$$

$$\text{or} \quad R = R_l R_r^T \quad \text{and} \quad T = T_r - R^T T_l \qquad (R_r R_l^T \neq R_l R_r^T)$$

Brigham Young University ROBOT C VISION LAB

OpenCV Functions

- Single camera calibration

double calibrateCamera(const vector<vector<Point3f> >& objectPoints,
const vector<vector<Point2f> >& imagePoints, Size imageSize,
Mat& cameraMatrix, Mat& distCoeffs, vector<Mat>& rvecs, vector<Mat>& tvecs,
int flags=0);

- Stereo calibration

double stereoCalibrate(const vector<vector<Point3f> >& objectPoints,
const vector<vector<Point2f> >& imagePoints1,
const vector<vector<Point2f> >& imagePoints2,
Mat& cameraMatrix1, Mat& distCoeffs1, Mat& cameraMatrix2, Mat& distCoeffs2,
Size imageSize, Mat& R, Mat& T, Mat& E, Mat& F,
TermCriteria term crit = TermCriteria(TermCriteria::COUNT+
TermCriteria::EPS, 30, 1e-6), int flags=CALIB FIX INTRINSIC);

OpenCV Functions

- calibrateCamera() outputs one set of intrinsic and distortion parameters and multiple sets of extrinsic parameters (between the camera and the chessboard views).
- stereoCalibrate() internally performs calibrateCamera() for left and right cameras separately and use those multiple sets of extrinsic parameters to calculate a single rotation matrix and translation vector (between the two cameras).

stereoCalibrate()

- Because of image noise and rounding error, each chessboard pair results in slightly different values for *rotation* and *translation.*
- *stereoCalibrate()* uses an iterative algorithm to find the minimum of the reprojection error of the chessboard corners for both camera views.
- The resulting rotation matrix and translation vector does not make images row-aligned as they would have been seen using canonical configuration.

stereoCalibrate()

- Chessboard size is critical
- It affects Tx and 3D Z measurement
- If the chessboard size used is half of the actual size, then
  - the system will think the chessboard is at half of the actual distance (it appears on the image twice as big for the wrong half size).
  - Since the distance is thought to be half, the baseline is reported half of the actual length (half of the actual Tx)

StereoCalibrate()

- objectPoints is the vector of vectors of points on the calibration pattern in its coordinate system, one vector per view. If the same calibration pattern is shown in each view and it's fully visible then all the vectors will be the same, although it is possible to use partially occluded patterns, or even different patterns in different views - then the vectors will be different.
- objectPoints are 3D points, but since they are in the pattern coordinate system, then if the rig is planar, it may make sense to put the model to the XY coordinate plane, so that Z-coordinate of each input object point is 0.
- imagePoints1 and imagePoints2 are the vector of vectors of corresponding image (corner) points.

StereoCalibrate()

- cameraMatrix1, cameraMatrix2, distCoeffs1, distCoeffs2 are the matrices of two intrinsic parameters and two vectors of distortion parameters.
- The function can also perform full calibration of each of the 2 cameras.
- However, because of the high dimensionality of the parameter space and noise in the input data the function can diverge from the correct solution.
- The intrinsic and distortion parameters should be first estimated using single camera calibration and then input to stereoCalibrate () function with the CV_ CALIB_FIX_INTRINSIC flag set.
- If all the parameters are estimated at once, CV CALIB SAME FOCAL LENGTH and CV CALIB ZERO TANGENT DIST flags should be set.
- The function outputs a unique set of rotation matrix and translation vector between the two cameras.
- The function also outputs the essential and fundamental matrices.

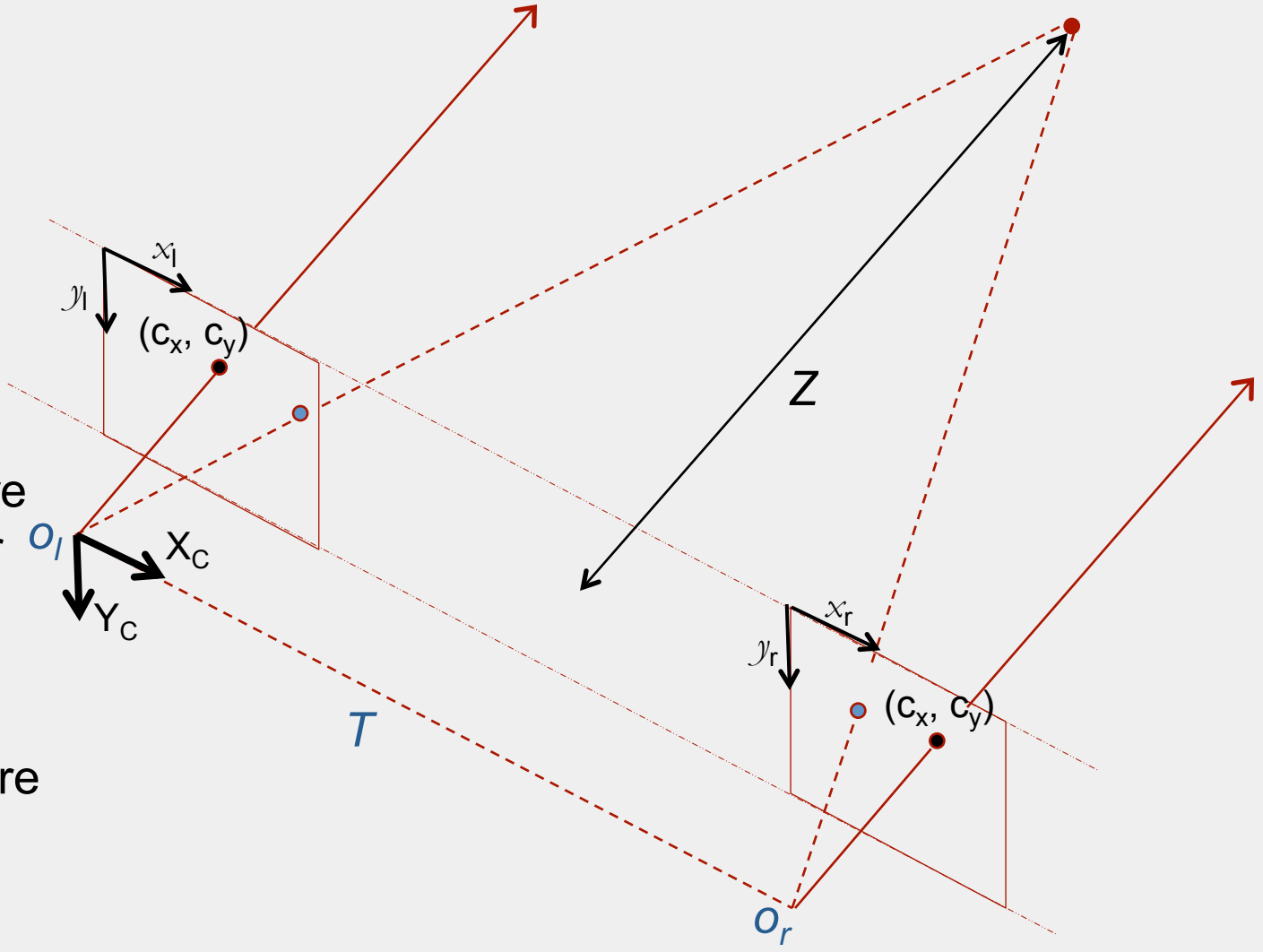
OpenCV use this coordinate system.

Left camera is the reference frame.

Images are undistorted.

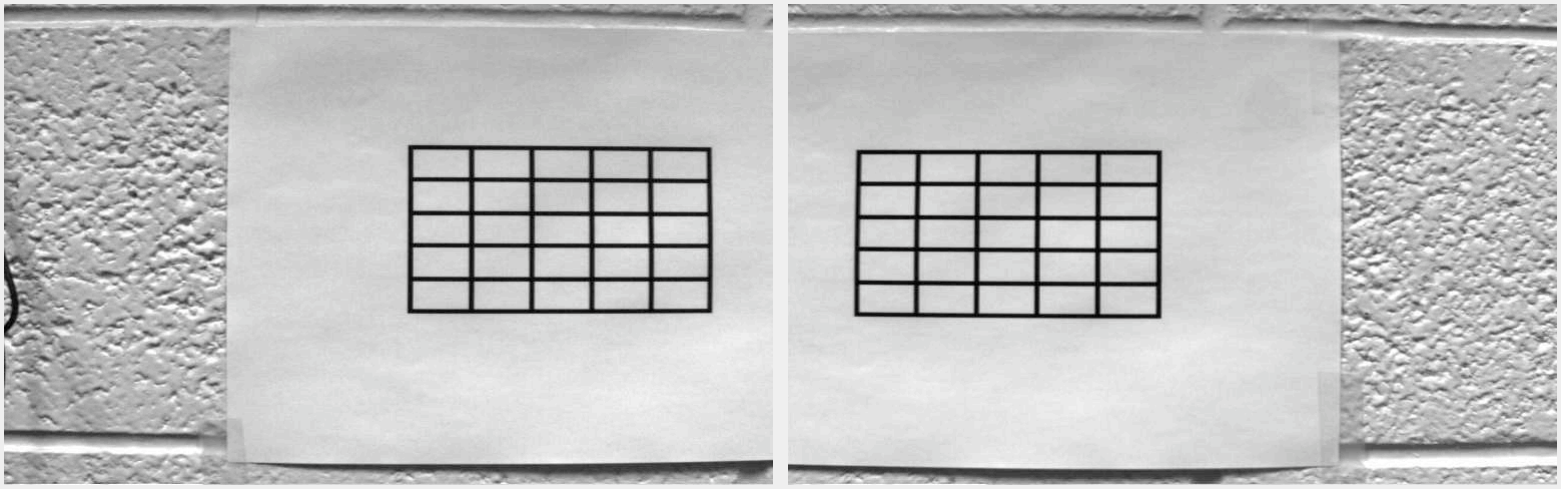
Pixel coordinates are relative to the upper left corner of the image.

Since image rows are aligned, finding correspondences becomes easier



BRIGHAM YOUNG UNIVERSITY ROBOT C VISION LAB

- Try to set up the stereo system as close to canonical configuration as possible.
- Adjust one or both cameras so that the only difference between two images is the horizontal translation.
- Correct for rotation, scaling, and perspective distortion as well.



- It is impossible to align two image planes perfectly
- Rectification can take care of the rest

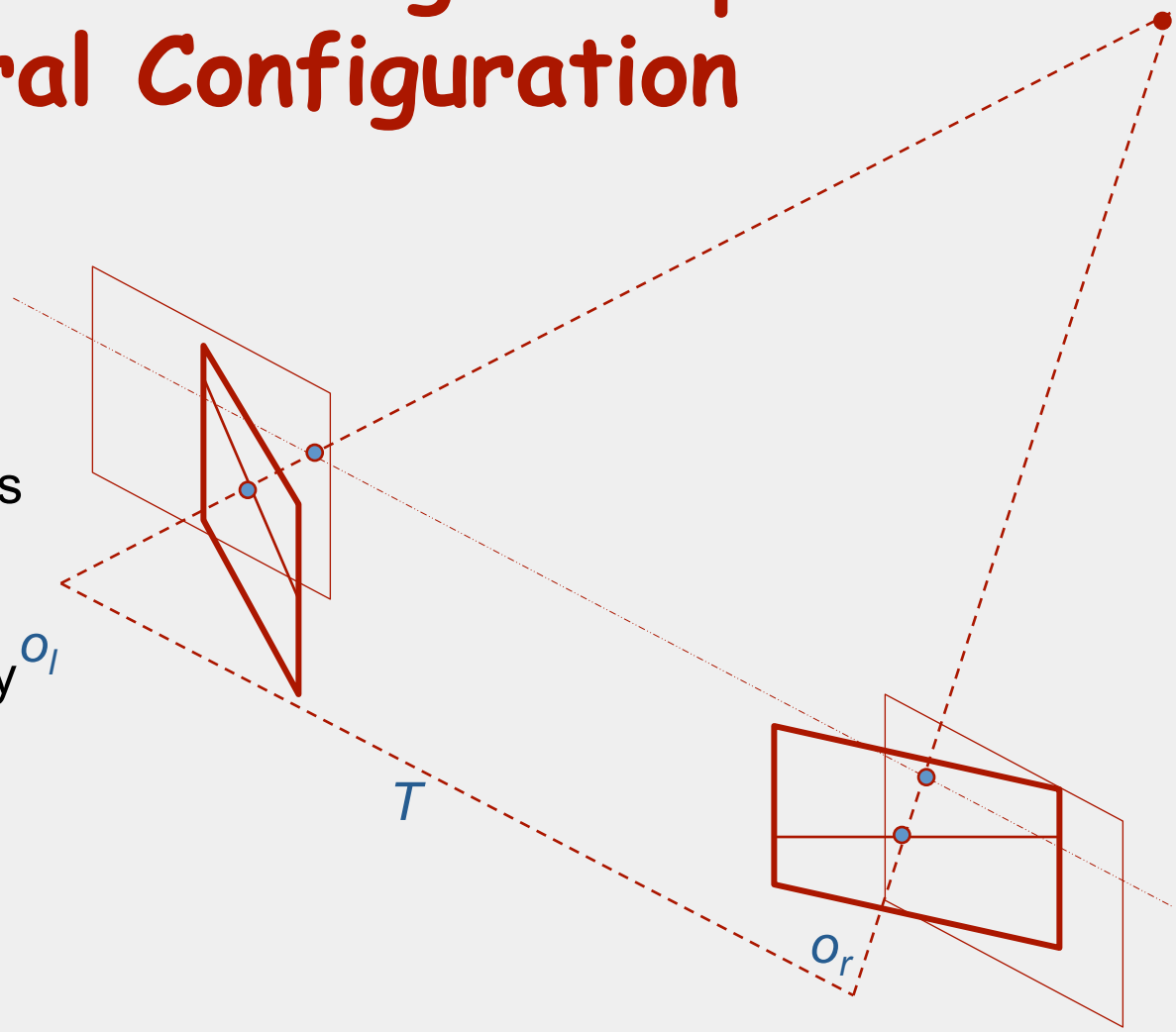
Image Rectification

- Why?
  - To change a 2-D search problem in general (for finding corresponding points) into a 1-D search problem (more reliable and can save time)
- How?
  - The rectified images can be thought of as acquired by a new stereo rig, obtained by rotating the original cameras around their optical centers.
  - Reproject both image planes so that they resides in the exact same plane and image rows perfectly aligned into a frontal parallel (canonical) configuration.

Rectification of Images Captured Using General Configuration

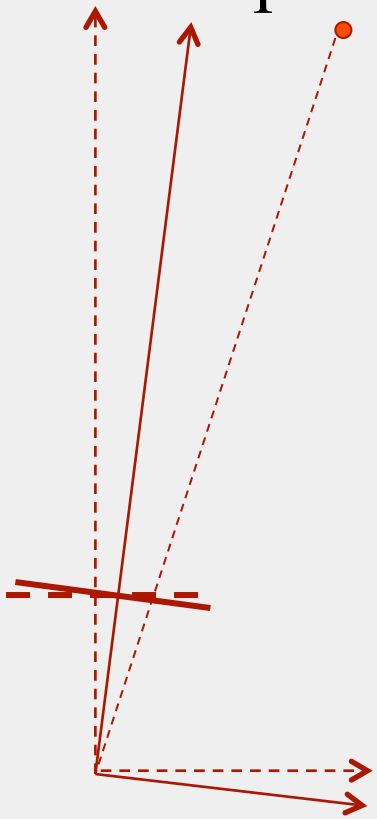
General configuration is more real.

Need to mathematically (not physically) align the two cameras into one viewing plane so that pixel rows are aligned.



Brigham Young University ROBOT C VISION LAB

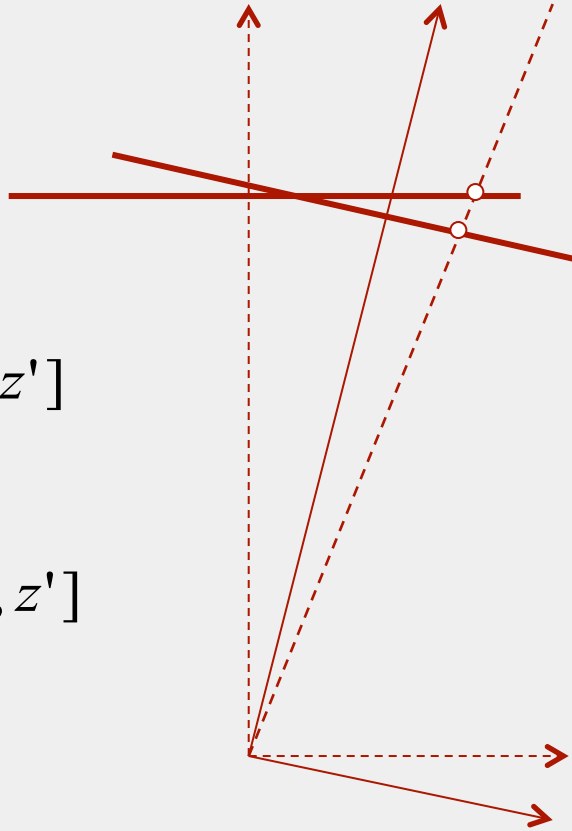
Convert pixel coordinates to camera Coordinates



$$p_l = [x, y, f]^T$$

$$R_l p_l = [x', y', z']$$

$$p'_l = \frac{f}{z'}[x', y', z']$$



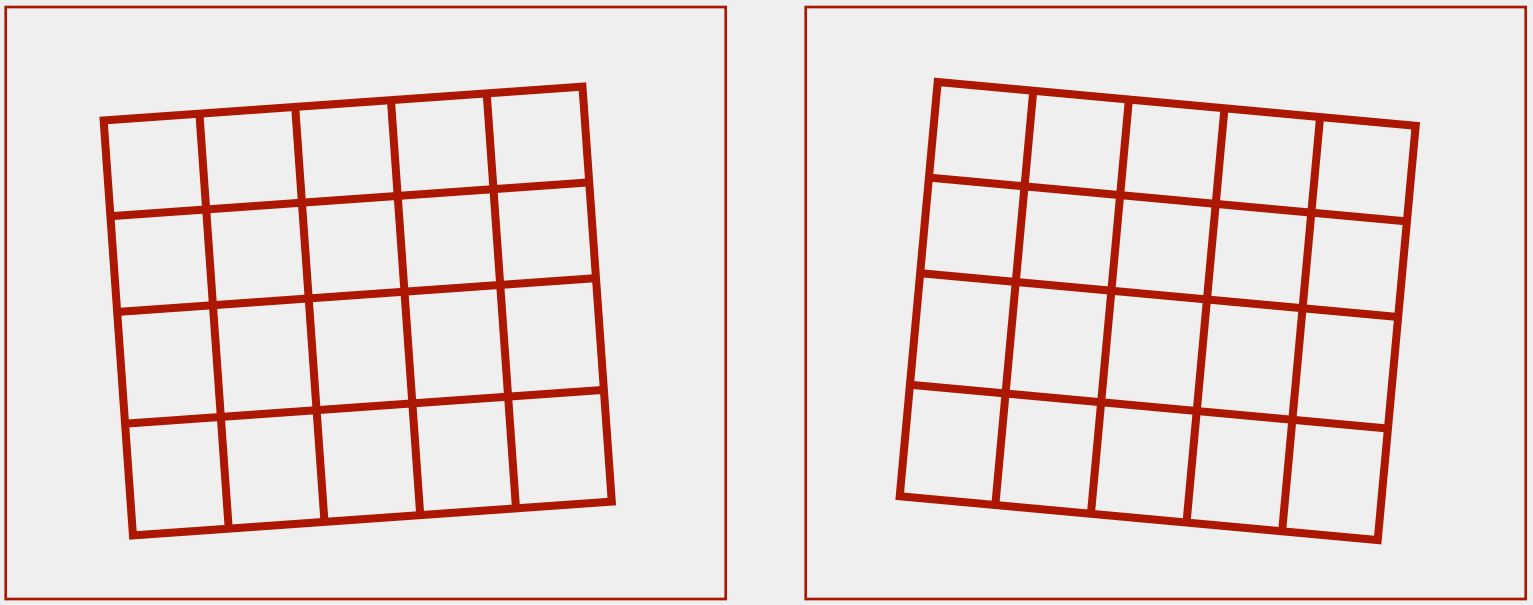
Same 3D point projected on two different image planes

Corresponding Rectified Point in pixel coordinates

BRIGHAM YOUNG UNIVERSITY ROBOT C VISION LAB

Image Rectification

- The geometry transformation that changes a general camera configuration with non-parallel epipolar lines to the canonical one.
- Epipolar lines become collinear and parallel to the image horizontal axis. Where are the epipoles?
- It is often used when stereo correspondence is to be determined by finding line-wise matching points.
- For high precision reconstruction, image rectification induces re-sampling and interpolation that cause loss of resolution.
- The rectified images can be thought of as acquired by rotated cameras.



Apply "inverse" transformation to find coordinates of the original image and then use interpolation to determine the rectified image. Keep image the same size by adjusting the focal length.

Brigham Young University ROBOT C VISION LAB

Illustration (Copyright Syntim-Inria)

Left

Right

Rectified Left

Rectified Right

Brigham Young University ROBOT C VISION LAB

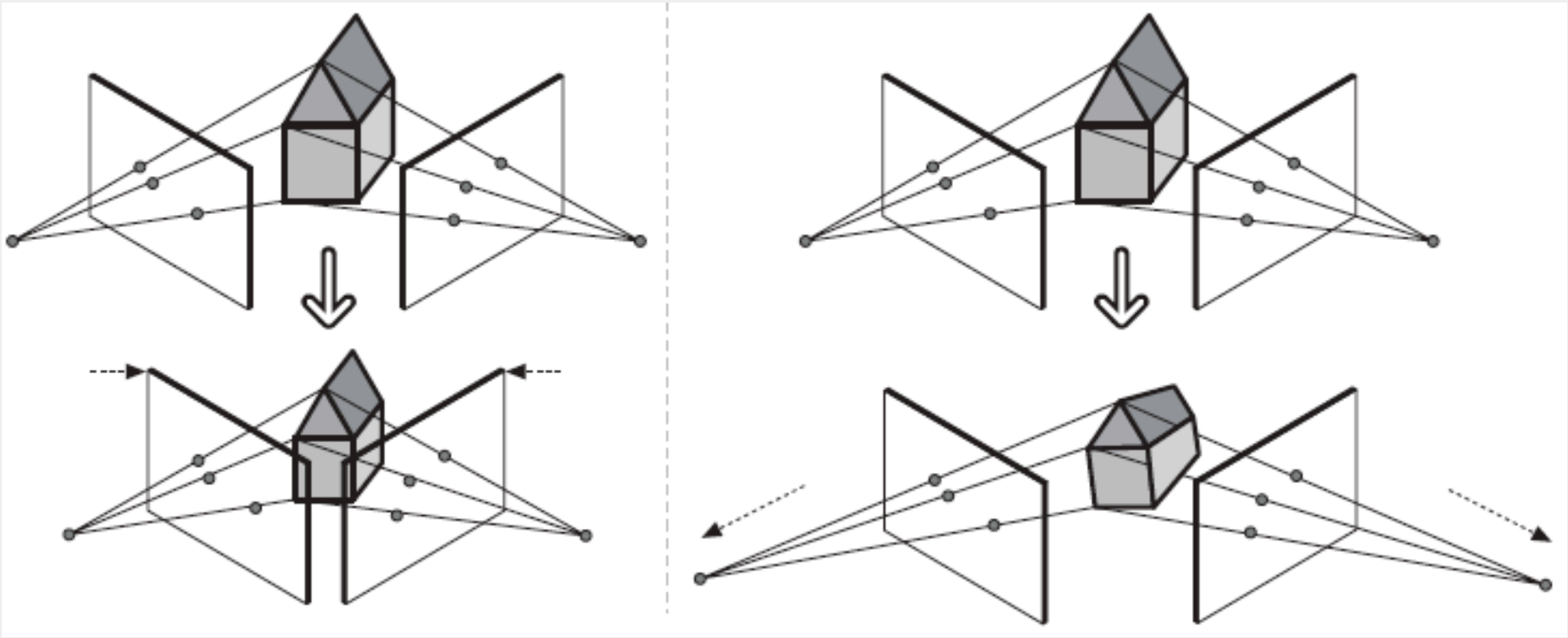
How is it done?

- There are an infinite number of possible frontal parallel planes to choose from. (as long as they are parallel)
- Choices include maximizing overlap and/or minimizing distortion.
- Rectification can be done with
  - Fundamental matrix of two uncalibrated cameras
    - Can be used for structure from motion using a single camera
    - Most likely distorted result
  - Rotation and translation parameters from two calibrated cameras
    - Result is better
    - It is preferable if stereo calibration can be done to get rotation matrix and translation vector.

Uncalibrated Stereo Rectification

- Bypass the computation of camera intrinsic parameters
- Intrinsic parameters are encoded in the fundamental matrix.
- Fundamental matrix can be obtained from seven or more matched points between two views using OpenCV functions (findFundamentalMat or stereoCalibrate)
- Easy to do but no sense of image scale (large object far away will appear the same as small object nearby)
- No explicit intrinsic parameters so can only obtain 3D object reconstruction only up to a projective transform (different scales or projections of an object could appear the same, same 2D from different 3D points)

Uncalibrated Stereo Rectification



Large object far away appears in 2D the same as small object nearby

Object from different perspectives could appear the same in 2D with different focal lengths and projection centers

Brigham Young University ROBOT C VISION LAB

Uncalibrated Stereo Rectification - Hartley's Algorithm

- Use the fundamental matrix *F* to find the two epipoles using $F\,\bar{e}_l = 0 \quad \text{and} \quad (\bar{e}_r)^T F = 0$
- Find a homography H_r that maps the right epipole to the 2D homogeneous point at infinity
- Find a matching homography H_l that send the left epipole to infinity
- The result will be very close to the calibrated case if 1) the two cameras have roughly the same parameters and the stereo system is very close to canonical configuration or 2) the object size is known.

Brigham Young University ROBOT C VISION LAB

OpenCV Function

bool stereoRectifyUncalibrated(const Mat& points1, const Mat& points2, const Mat& F, Size imgSize, Mat& H1, Mat& H2, double threshold=5);

- F can be computed by findFundamentalMat()
- streoRectifyUncalibrated() computes rectification transformations H1 and H2 without intrinsic parameters of the cameras and their relative position in space.
- The rectification transformations are actually planar perspective transformations encoded by the homography matrices
- Camera lens distortion could significantly affect the accuracy.

OpenCV Function - Rectification

```
void initUndistortRectifyMap( const Mat& cameraMatrix, const Mat& distCoeffs,
    const Mat& R, const Mat& newCameraMatrix, Size size, int m1type,
    Mat& map1, Mat& map2 );
void remap( const Mat& src, Mat& dst, const Mat& map1, const Mat& map2,
    int interpolation, int borderMode=BORDER_CONSTANT, const Scalar&
    borderValue=Scalar());
```

- We still need cameraMatrix (M) and distCoeffs to rectify the image. They can be an estimate (guess).
- For uncalibrated camera, newCameraMatrix (M_{rect}) can be the same as cameraMatrix (M)
- $R_1 = M_1^{-1} H_1 M_1$ and $R_2 = M_2^{-1} H_2 M_2$
- * remap( ) is used for remapping the entire image if the map (transformation) is available.

Brigham Young University ROBOT C VISION LAB

Calibrated Stereo Rectification

- Given a stereo pair of images, intrinsic parameters of each camera and extrinsic parameters of the stereo system – compute the transformation that makes epipolar lines parallel to the image horizontal axis.
- The rectified images can be thought of as acquired by a new stereo rig, obtained by rotating the original cameras around their optical centers.
- Attempts to minimize reprojection distortion

Procedure

- Camera Calibration to get Intrinsic/Extrinsic Parameters.
- Build a rotation matrix R_{rect} , which can rotate left camera to make left epipole go to infinity (epipolar lines become horizontal).
- Apply the same rotation to the right camera
- Rotate the right camera by R (between cameras from calibration)
- For each left-camera point, calculate a corresponding point in the new stereo rig.
- Do the same thing for the right camera.
- Adjust the scale in both camera reference frames.

Rectification Step 1

- Calibration

$$P_l = R_l P_w + T_l \quad and \quad P_r = R_r P_w + T_r$$

Use either Direct Parameter Calibration, Camera Parameters from the projection matrix (indirect), or homography-based calibration, to estimate f_X, f_Y, O_X, O_Y for left and right camera and R_l, T_l, R_r, T_r.

Brigham Young University ROBOT C VISION LAB

Rectification Step 2

- Calculate Parameters for the stereo system. With the parameters from Step 1,

$$R = R_r R_l^T \quad and \quad T = T_l - R^T T_r$$

- composeRT() can be used to solve R and T
- Use SVD to decompose $R=UDV^T$
- Replace D with an identity matrix to reconstruct a better R
- *stereoCalibrate()* uses an iterative algorithm to find the minimum of the reprojection error of the chessboard corners for both camera views
- We can use *stereoCalibrate()* to calculate *R* and *T*.

Rectification Step 3

- Build the rotation matrix R_{rect} for the left camera so that the left epipole goes to infinity and epipolar lines become horizontal

$$e_1 = \frac{T}{\|T\|}$$

A unit vector along the baseline

$$e_2 = \frac{1}{\sqrt{T_x^2 + T_y^2}}[-T_y, T_x, 0]^T$$

A normalized cross-product of e_1 with the optical axis that is on the image plane

$$e_3 = e_1 \times e_2$$

$$R_{rect} = \begin{pmatrix} e_1^T \\ e_2^T \\ e_3^T \end{pmatrix}$$

Brigham Young University ROBOT C VISION LAB

- Why the rotation matrix is reconstructed this way? Apply the rectification matrix to both sides of

$$P_l = R^T P_r + T$$

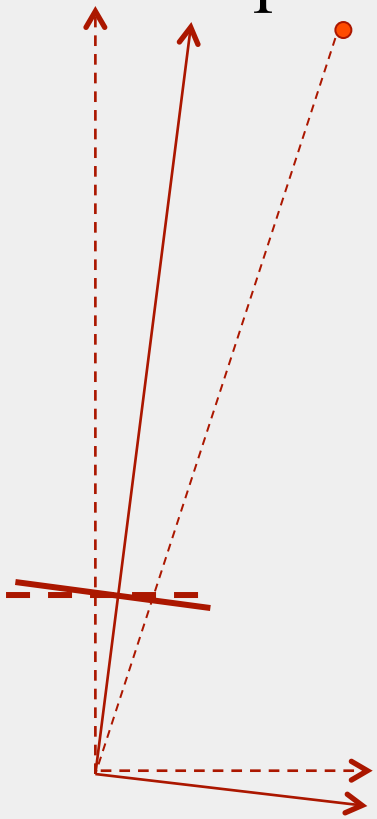
$$R_{rect} P_l = R_{rect} R^T P_r + R_{rect} T$$

$$R_{rect} T = \begin{pmatrix} \| T \| \\ 0 \\ 0 \end{pmatrix}$$

Only *x* has an offset

Brigham Young University ROBOT C VISION LAB

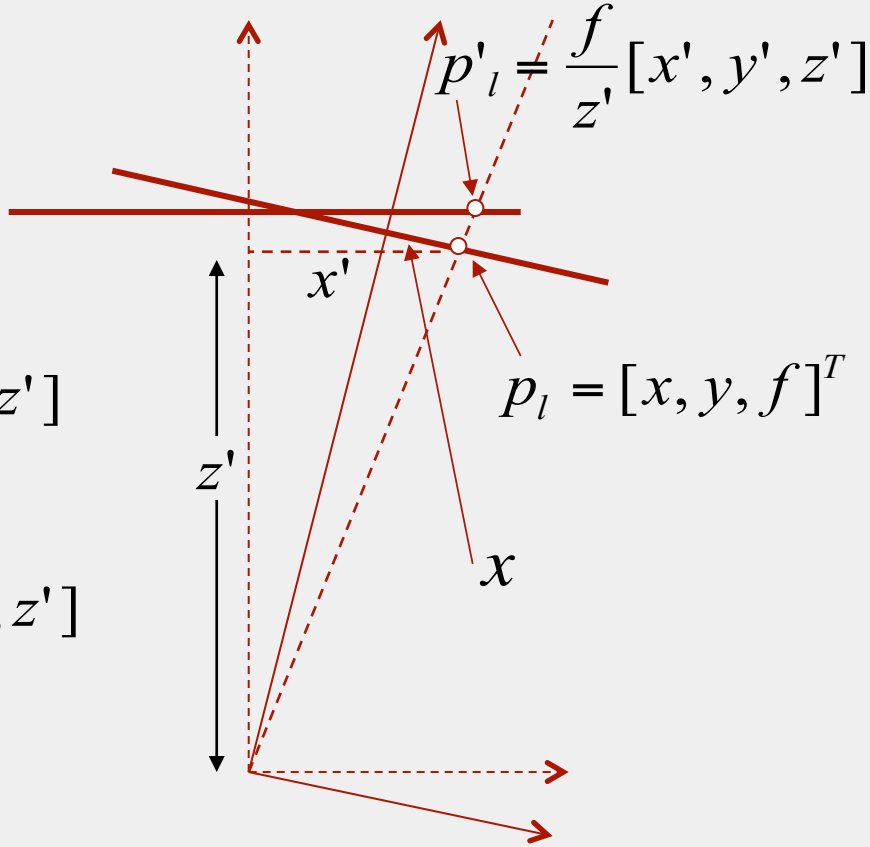
Convert pixel coordinates to camera Coordinates



$$p_l = [x, y, f]^T$$

$$R_l p_l = [x', y', z']$$

$$p'_l = \frac{f}{z'}[x', y', z']$$



Corresponding Rectified Point in pixel coordinates

Brigham Young University ROBOT C VISION LAB

Rectification Step 4-1

- For each left camera point $p_l = [x, y, f]^T$, compute:

$$R_{rect} p_l = [x', y', z']^T$$

and the coordinates of the corresponding rectified points, p_l' as

$$p_l' = \frac{f}{z'}[x', y', z']^T$$

Brigham Young University ROBOT C VISION LAB

Rectification Step 4-2

- For each right camera point $p_r = [x, y, f]^T$, compute:

$$RR_{rect}\, p_r = [x', y', z']^T$$

and the coordinates of the corresponding rectified points, p_r' as

$$p_r' = \frac{f}{z'}[x', y', z']^T$$

Brigham Young University ROBOT C VISION LAB

Rectification Step 5

But those p_l and p_l' (as well as p_r and p_r') are in camera coordinate system (not in pixels), how to associate them with pixel coordinates?

Through intrinsic parameters:

$$x_i - o_x = s_x \cdot x_c \qquad s_x = f_x / f$$

$$y_i - o_y = s_y \cdot y_c \qquad s_y = f_y / f$$

Brigham Young University ROBOT C VISION LAB

Through Calibration, we have f_X and f_Y ,but we don't have f, what to do now? It cancels out, so we don't really need it.

Brigham Young University ROBOT C VISION LAB

Rectification Step 6

Perform the last step backwards, which means for each pixel in the rectified image, we need to find the correspondent point in the original image. The reason is that if we don't do this, we will have a lot of holes in the rectified image.

Brigham Young University ROBOT C VISION LAB

$$(x_i - o_x)\frac{f}{f_x} = x_c$$

$$(y_i - o_x)\frac{f}{f_y} = y_c$$

$$R_{rect}\,p_l = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \quad p_l' = \begin{bmatrix} fx'/z' \\ fy'/z' \\ f \end{bmatrix}$$

$$\frac{fx'}{z'} = (x_i' - o_x)\frac{f}{f_x} \Rightarrow x' = (x_i' - o_x)\frac{f}{f_x}\cdot\frac{z'}{f} = (x_i' - o_x)\frac{z'}{f_x}$$

$$\frac{fy'}{y'} = (y_i' - o_x)\frac{f}{f_y} \Rightarrow y' = (y_i' - o_y)\frac{f}{f_y}\cdot\frac{z'}{f} = (y_i' - o_y)\frac{z'}{f_y}$$

Brigham Young University ROBOT C VISION LAB

$$\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = R_{rect}^{-1} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_{rect}^{T} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{21} & r_{31} \\ r_{12} & r_{22} & r_{32} \\ r_{13} & r_{23} & r_{33} \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

If use real image plane, f should be - f

$$f = (x_i' - o_x)\frac{z'}{f_x} r_{13} + (y_i' - o_y)\frac{z'}{f_y} r_{23} + r_{33} z'$$

$$\Rightarrow \frac{z'}{f} = \frac{1}{(x_i' - o_x)\frac{1}{f_x} r_{13} + (y_i' - o_y)\frac{1}{f_y} r_{23} + r_{33}}$$

Brigham Young University ROBOT C VISION LAB

$$(x_i - o_x)\frac{f}{f_x} = x_c = r_{11}x' + r_{21}y' + r_{31}z'$$

$$= (x_i' - o_x)\frac{z'}{f_x}r_{11} + (y_i' - o_y)\frac{z'}{f_y}r_{21} + r_{31}z'$$

$$(x_i - o_x) = (x'_i - o_x)\frac{z'}{f_x}\frac{f_x}{f}r_{11} + (y'_i - o_y)\frac{z'}{f_y}\frac{f_x}{f}r_{21} + r_{31}\frac{f_x}{f}z'$$

$$\Rightarrow x_i = (x'_i - o_x)\frac{z'}{f}r_{11} + (y'_i - o_y)\frac{z'}{f}\frac{f_x}{f_y}r_{21} - r_{31}\frac{z'}{f}f_x + o_x$$

Brigham Young University ROBOT C VISION LAB

$$(y_i - o_y)\frac{f}{f_y} = y_c = r_{12}x' + r_{22}y' + r_{32}z'$$

$$= (x'_i - o_x)\frac{z'}{f_x}r_{12} + (y'_i - o_y)\frac{z'}{f_y}r_{22} + r_{32}z'$$

$$(y_i - o_y) = (x'_i - o_x)\frac{z'}{f_x}\frac{f_y}{f}r_{12} + (y'_i - o_y)\frac{z'}{f_y}\frac{f_y}{f}r_{22} + r_{32}\frac{f_y}{f}z'$$

$$\Rightarrow y_i = (x'_i - o_x)\frac{z'}{f}\frac{f_x}{f_y}r_{12} + (y'_i - o_y)\frac{z'}{f}r_{22} - r_{32}\frac{z'}{f}f_y + o_y$$

Brigham Young University ROBOT C VISION LAB

1. Need to use T to construct R_{rect}.
2. *Use Step 4-2 and r*epeat Steps 5 and 6 to rectify right image.

$$R_{rect} = \begin{bmatrix} -0.9757 & 0.0311 & -0.2169 \\ -0.0319 & -0.9995 & 0.0000 \\ -0.2168 & 0.0069 & 0.9762 \end{bmatrix}$$

cos –(~0⁰)

$$R = \begin{bmatrix} -0.9999 & 0.0000 & 0.0000 \\ 0.0000 & -0.9999 & 0.0000 \\ -0.0000 & 0.0000 & 0.9999 \end{bmatrix}$$

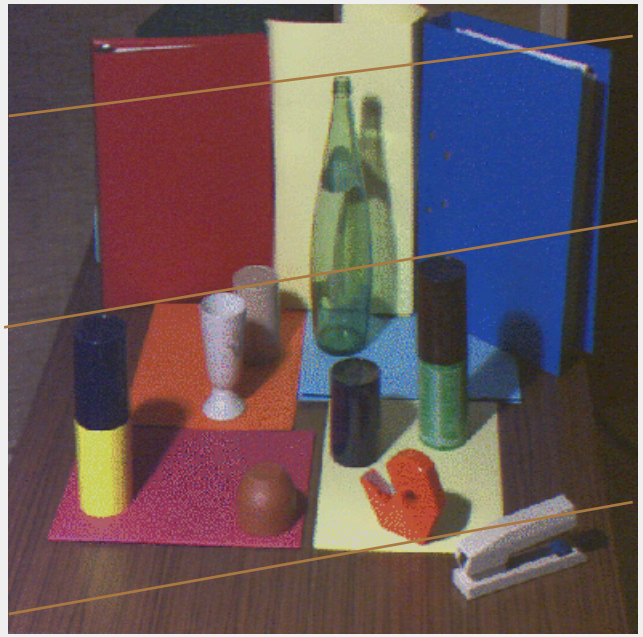
cos (~0⁰)

Sign does not matter when ~0⁰

Brigham Young University ROBOT C VISION LAB

Results (Case 1)

Left

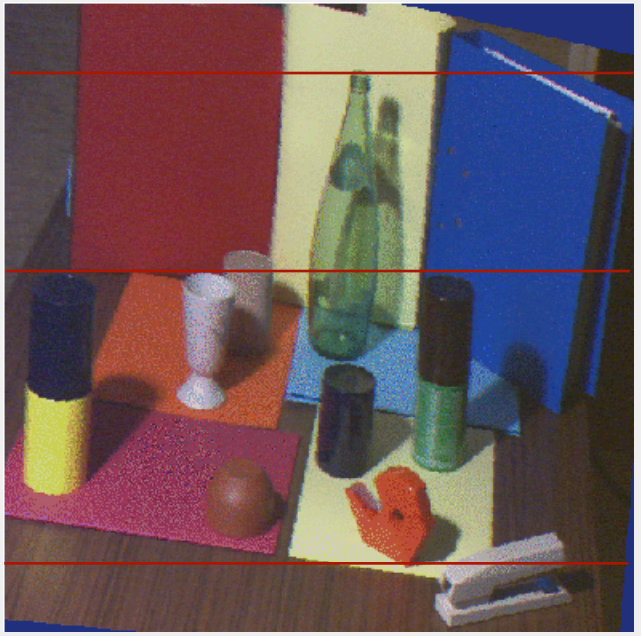


Right

Brigham Young University ROBOT C VISION LAB

Results (Case 1)

Rectified Left



Rectified Right

Brigham Young University ROBOT C VISION LAB

OpenCV Function - Rectification

```
void stereoRectify( const Mat& cameraMatrix1, const Mat& distCoeffs1,
    const Mat& cameraMatrix2, const Mat& distCoeffs2, Size imageSize,
    const Mat& R, const Mat& T, Mat& R1, Mat& R2, Mat& P1, Mat& P2, Mat& Q,
    int flags=CALIB ZERO DISPARITY );
void stereoRectify( const Mat& cameraMatrix1, const Mat& distCoeffs1,
    const Mat& cameraMatrix2, const Mat& distCoeffs2, Size imageSize,
    const Mat& R, const Mat& T, Mat& R1, Mat& R2, Mat& P1, Mat& P2, Mat& Q,
    double alpha, Size newImageSize=Size(), Rect* roi1=0, Rect* roi2=0,
    int flags=CALIB ZERO DISPARITY );
```

- stereoRectify() inputs 6 matrices from stereocalibrate() and outputs 2 rotation matrices (R1 and R2), 2 projection matrices (P1 and P2), and one reprojection matrix Q.
- Both stereoCalibrate() and stereoRectify() need only be done once.

OpenCV Function - Rectification

```
void initUndistortRectifyMap( const Mat& cameraMatrix, const Mat& distCoeffs,
    const Mat& R, const Mat& newCameraMatrix, Size size, int m1type,
    Mat& map1, Mat& map2 );
void remap( const Mat& src, Mat& dst, const Mat& map1, const Mat& map2,
    int interpolation, int borderMode=BORDER_CONSTANT, const Scalar&
    borderValue=Scalar());
```

- newCameraMatrix is usually the projection transformation matrix (P1 or P2) from stereoRectify()
- Called twice, one for each camera to calculate the map and remap the image.
- Map is calculated only once and can be reused for remap all input images later.
- * remap( ) is used for remapping the entire image if the map (transformation) is available.

Brigham Young University ROBOT C VISION LAB

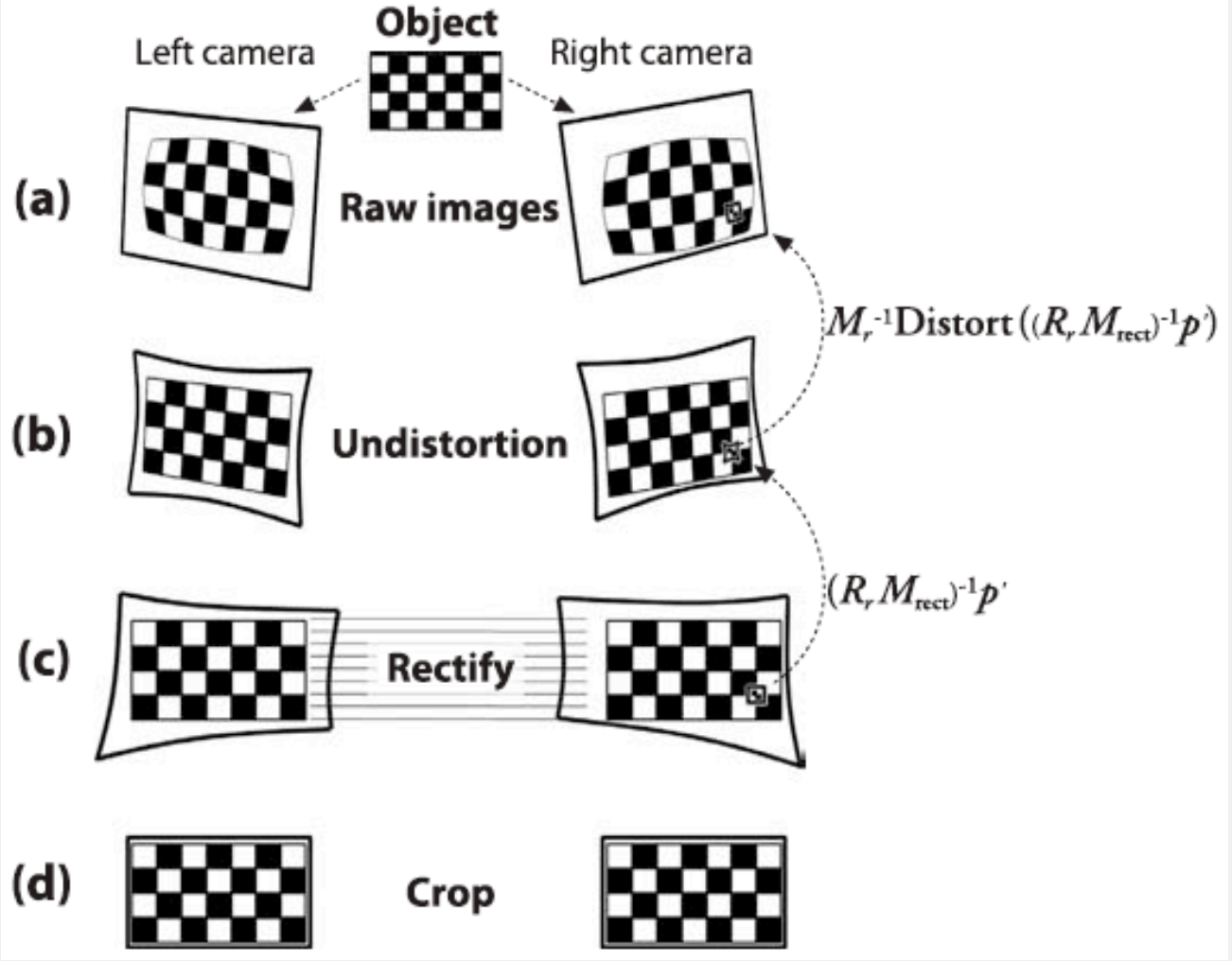
OpenCV Function - Rectification

```
undistortPoints( const Mat& src, vector<Point2f>& dst, const Mat& cameraMatrix,
    const Mat& distCoeffs, const Mat& R=Mat(), const Mat& P=Mat());
undistortPoints( const Mat& src, Mat& dst, const Mat& cameraMatrix, const Mat&
    distCoeffs, const Mat& R=Mat(), const Mat& P=Mat());
```

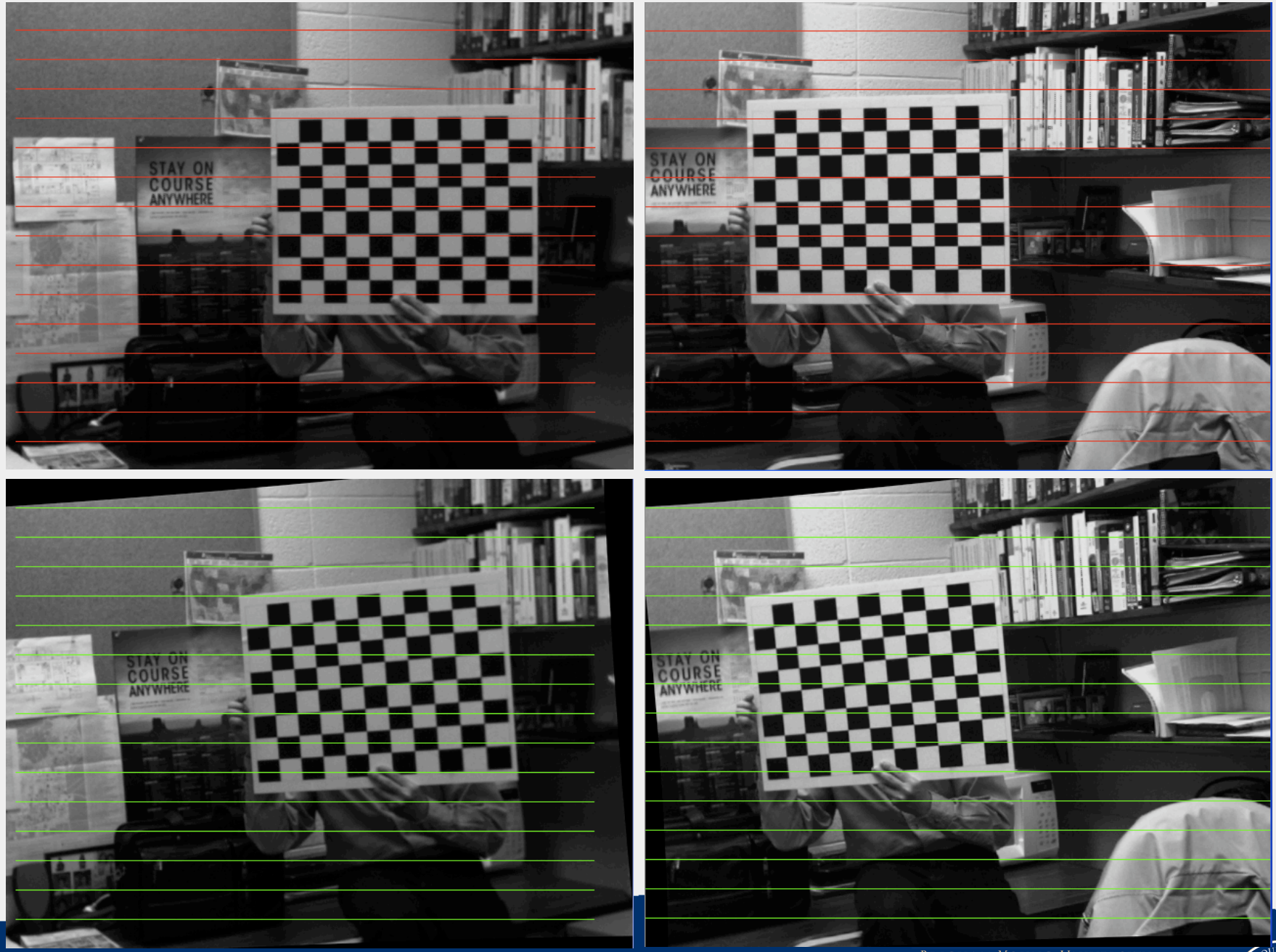
- Undistorts and rectifies points at the same time.
- Corrects lens distortion (undistort) only, if R and P are identity matrices.
- There is no need to rectify the entire image (tennis ball project).

- * undistort() does not take two extra parameters (R and P) and is only used for correcting lens distortion.

OpenCV



Brigham Young University ROBOT C VISION LAB



BRIGHAM YOUNG UNIVERSITY
ROBOT C
VISION LAB
BYU FOUNDED 1875 PROVO, UTAH