# EE 046746 - Technion - Computer Vision

Elias Nehme

## Tutorial 10 - Camera Calibration and Epipolar Geometry

---

## Agenda

---

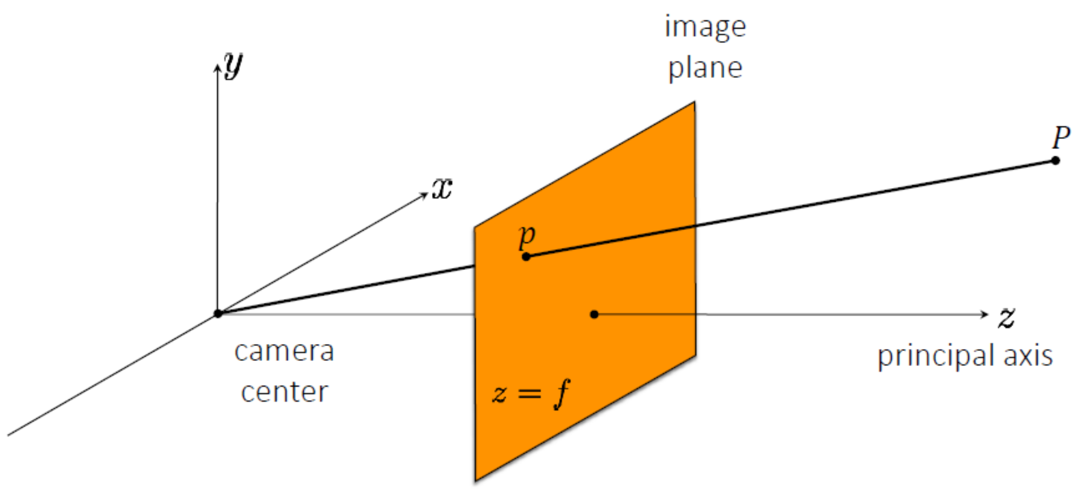

- Camera Model
- Camera Calibration
  - Estimating M
  - Chessboard Demo
  - Homography Quiz
- Epipolar Geometry
  - Essential Matrix
  - Fundamental Matrix
  - Fundamental Demo
- Recommended Videos
- Credits


## Camera Model

---

### The (rearranged) pinhole camera

---




What is the camera matrix **M** for a pinhole camera?

$$p = MP$$

- A 3D world point $P$ is projected by the camera matrix $M$ to the 2D image point $p$

### The (rearranged) pinhole camera

A camera is a mapping from:

the 3D world

to:

a 2D image

homogeneous coordinates

$$\underbrace{p}_{\text{2D image point}} = \underbrace{M}_{\text{camera matrix}}\underbrace{P}_{\text{3D world point}}$$

What are the dimensions of each variable?

## The (rearranged) pinhole camera

$$p = MP$$

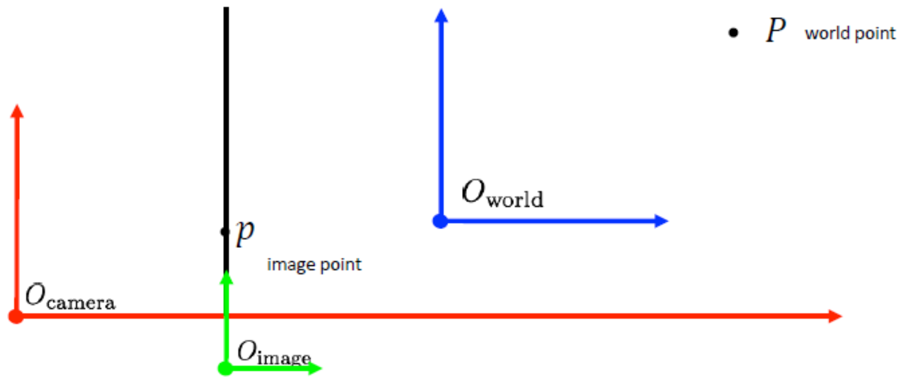
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 & m_4 \\ m_5 & m_6 & m_7 & m_8 \\ m_9 & m_{10} & m_{11} & m_{12} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

| homogeneous image coordinates | camera matrix | homogeneous world coordinates |
|---|---|---|
| 3 x 1 | 3 x 4 | 4 x 1 |

- What is the decomposed structure of $M$?

## The Camera Matrix

In general, there are *three*, generally different, coordinate systems.




We need to know the transformations between them.

- $M$ is a $3 \times 4$ matrix comprised of two sets of parameters: **Intrinsic** and **Extrinsic**.

## The Camera Matrix

$$\mathrm{M} = K\,[R|t]$$

$$\mathrm{M} = \underbrace{\begin{bmatrix} f & 0 & m_x \\ 0 & f & m_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic parameters}} \underbrace{\left[\begin{array}{ccc:c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \end{array}\right]}_{\text{extrinsic parameters}}$$

$$\mathbf{R} = \underbrace{\begin{bmatrix} r_1 & r_2 & r_3 \\ r_4 & r_5 & r_6 \\ r_7 & r_8 & r_9 \end{bmatrix}}_{\text{3D rotation}} \qquad \mathbf{t} = \underbrace{\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}}_{\text{3D translation}}$$

- How many degress of freedom so far?
- And after switching $f$ with $f_x$ and $f_y$ and adding skew $s$?

## Camera Calibration

- Estimation of $M$
- Separating Extrinsic and Intrinsic Parameters

## Geometric Camera Calibration: Estimating $M$

Given a set of matched points

$$\{\underbrace{\boldsymbol{P_i}}_{\text{point in 3D space}}, \underbrace{\boldsymbol{p_i}}_{\text{point in the image}}\}$$

and camera model

$$\boldsymbol{p} = \underbrace{\boldsymbol{f}}_{\text{projection model}}(\boldsymbol{P}; \underbrace{\boldsymbol{m}}_{\text{parameters}}) = \underbrace{\boldsymbol{M}}_{\text{Camera matrix}} \boldsymbol{P}$$

Find the (pose) estimate of

$$\boldsymbol{M}$$

We'll use a **perspective** camera model for pose estimation

- Where did we get such matched points?

## Estimating $M$

- Same trick as in the Homogrpahy tutorial $\rightarrow$ switch to row-wise representation of the unknowns:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Equivalently

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} - & m_1^T & - \\ - & m_2^T & - \\ - & m_3^T & - \end{bmatrix} P$$

## Estimating $M$

- Resulting equation for $x$ and $y$ in heterogeneous coordinates:

$$\tilde{x} = \frac{m_1^T P}{m_3^T P}, \tilde{y} = \frac{m_2^T P}{m_3^T P}$$

- Rearranging to solve for $m_i$:

$$m_1^T P - \tilde{x} m_3^T P = 0$$

$$m_2^T P - \tilde{y} m_3^T P = 0$$

- What is the dimension of $m_i^T P$?

## Estimating $M$

- Rearrange into a matrix for N points:

$$\begin{bmatrix} P_i^T & 0^T & -\tilde{x}_i P_i^T \\ 0^T & P_i^T & -\tilde{y}_i P_i^T \\ . & . & . \\ . & . & . \\ P_N^T & 0^T & -\tilde{x}_N P_N^T \\ 0^T & P_N^T & -\tilde{y}_N P_N^T \end{bmatrix} \begin{bmatrix} | \\ m_1 \\ | \\ m_2 \\ | \\ m_3 \\ | \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ . \\ . \\ 0 \\ 0 \end{bmatrix} \leftrightarrow Am = 0$$

- What are the dimensions? How much points N do we need?

## Estimating $M$

- boils down to the problem:

$$\hat{m} = \arg\min_m \|Am\|^2 \; s.t. \; \|m\|^2 = 1$$

- Solution via SVD of $A = U\Sigma V^T$:
  - $\hat{m}$ is the column of V corresponding to the smallest eigen-value.
- How about separating $M$ to $K\left[R|t\right]$?

## Decomposition of M to K, R & t

- rewrite $M$:

$$M = K\left[R|t\right] = K\left[R|-Rc\right] = \left[N|-Nc\right]$$

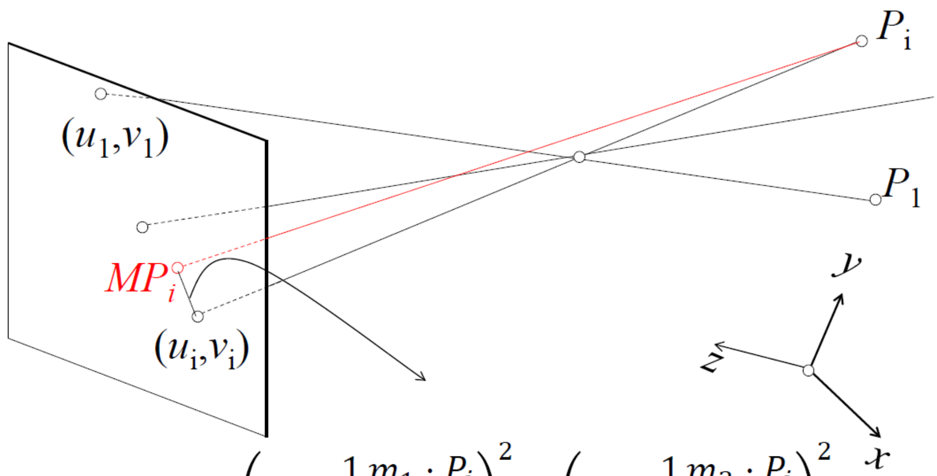
- $c$ can be found via SVD of $M$ due to the relation:

$$Mc = 0$$

- Then $N$ can be found and further decomposed into $N = KR$:
  - How? using QR decomposition becuase $K$ is upper triangular and $R$ is orthogonal
- However..
  - Does not take into account noise, radial distortions, hard to impose prior knowledge (e.g. $f$), etc.
  - Solution?

## Minimize reprojection error

# Minimizing reprojection error with radial distortion




Add distortions to reprojection error:

$$\left(u_i - \frac{1}{\lambda}\frac{m_1 \cdot P_i}{m_3 \cdot P_i}\right)^2 + \left(v_i - \frac{1}{\lambda}\frac{m_2 \cdot P_i}{m_3 \cdot P_i}\right)^2$$

- Where the radial distortion model is:

$$\lambda = 1 + k_1 r^2 + k_2 r^4 + k_3 r^6$$

## Minimize reprojection error

- Radial distortion is multiplicative:

$$x_{rad} = x\left[1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right]$$

$$y_{rad} = y\left[1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right]$$

- Usually we also include tangential distortion (additive):

$$x_{tan} = x + \left[2p_1 xy + p_2\left(r^2 + 2x^2\right)\right]$$

$$y_{tan} = y + \left[p_1\left(r^2 + 2y^2\right) + 2p_2 xy\right]$$

- We end up with 5 parameters to estimate:

$$\text{distortion coefficients} = \left[k_1, k_2, k_3, p_1, p_2\right]^T$$

## Chessboard Calibration in OpenCV

- Take a notebook and paste a chesspattern
- Capture this pattern from several angles and positions
- Calibrate using OpenCV

## Chessboard Calibration in OpenCV

- Getting the 3D to 2D points correspondences from a known planar object
- Chessboard has fixed distances between squares known appriori
- Camera static and chessboard moves ↔ chessboard static and camera moves

- Camera moves $\leftrightarrow$ Extrinsic parameters in each frame change
- Therefore we got the matches of real world points and camera points $\{P_i, p_i\}_{i=1}^{N}$!
    - $P_i = [X_i, Y_i, Z_i = 0]$, where, $X_i, Y_i$ set by periodicity of the chessbaord
    - $p_i = [x_i, y_i]$, detected corners in the image

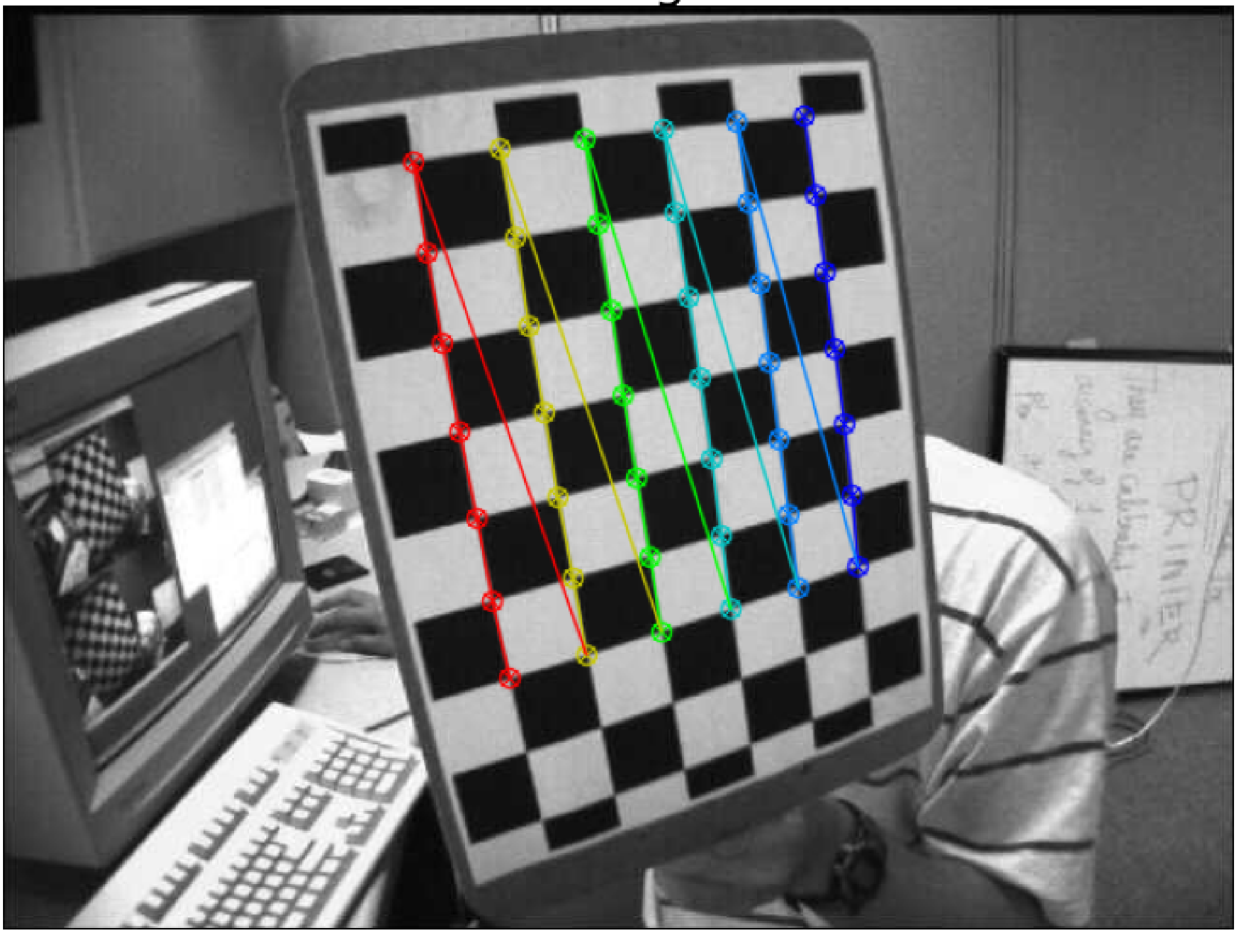
In [1]:

```
import numpy as np
import cv2
import glob
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob('./assets/left*.jpg')
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv2.findChessboardCorners(gray, (7,6), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv2.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners)
        # Draw and display the corners
        imlast = cv2.drawChessboardCorners(img, (7,6), corners2, ret)
        cv2.imshow('img', img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
```

In [2]:

```
# show the last image and the detected corners
import matplotlib.pyplot as plt
plt.figure(figsize=(13,10))
plt.imshow(imlast)
plt.axis('off')
plt.title('Calibration Target Corners', fontsize=30)
plt.show()
```

## Calibration Target Corners



In [3]:

```
# now that we have object points and and image points, we jsut apply OpenCV builtin function
ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)

# resulting camera matrix
print("M = ")
print(repr(mtx))

# distortion coeff.
print("distortion coeff = ")
print(repr(dist))

# Rodriguez rotation vectors and translation vectors
print("Rotation vector 1 = ")
print(rvecs[0])
print("Translation vector 1 = ")
print(tvecs[0])

print("\n . \n . \n . \n")

print("Rotation vector N = ")
print(rvecs[-1])
print("Translation vector N = ")
print(tvecs[-1])
```

```
M =
array([[534.07088364,   0.        , 341.53407554],
       [  0.        , 534.11914595, 232.94565259],
       [  0.        ,   0.        ,   1.        ]])
distortion coeff =
array([[-2.92971637e-01,  1.07706962e-01,  1.31038376e-03,
        -3.11018780e-05,  4.34798110e-02]])
Rotation vector 1 =
[[-0.43239599]
 [ 0.25603401]
 [-3.08832021]]
Translation vector 1 =
[[ 3.79739146]
 [ 0.89895018]
 [14.8593055 ]]
```

```
.
.
.

Rotation vector N =
[[-0.17288944]
 [-0.46764681]
 [ 1.34745198]]
Translation vector N =
[[ 1.81888151]
 [-4.2642919 ]
 [12.45728517]]
```

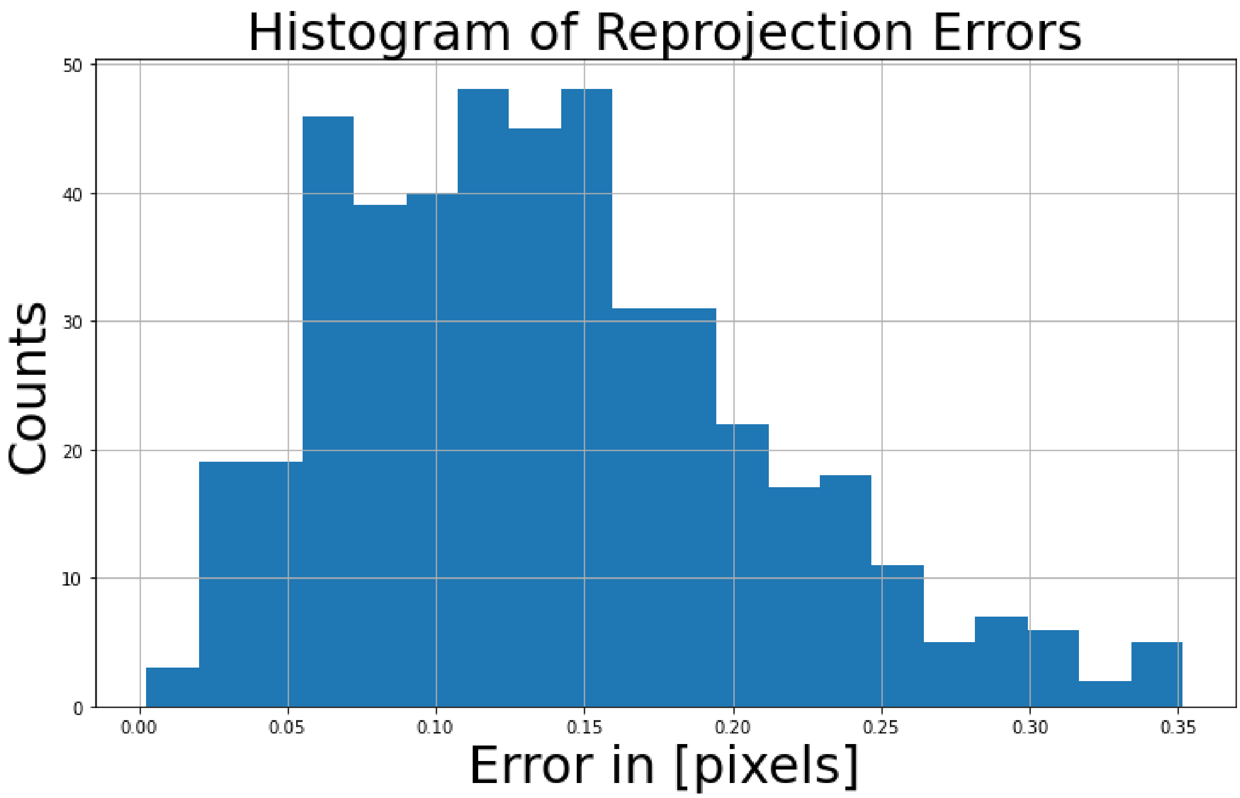
In [4]:

```python
# Let us examine the distortion on a given image
img = cv2.imread('./assets/left12.jpg')
h,  w = img.shape[:2]
newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

# undistort
dst = cv2.undistort(img, mtx, dist, None, newcameramtx)
# crop the image
x, y, w, h = roi
dst = dst[y:y+h, x:x+w]

# plot the image ebfore and after fixing distortion
plt.figure(figsize=(18,13))
plt.subplot(1,2,1)
plt.imshow(img)
plt.axis('off')
plt.title('Distorted', fontsize=30)
plt.subplot(1,2,2)
plt.imshow(dst)
plt.axis('off')
plt.title('Undistorted', fontsize=30)
plt.show()
```


In [5]:

```python
# checking reprojection errors for validation - ideally we should get ~0
mean_error = 0
errs_all = []
for i in range(len(objpoints)):
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)
    errs_all.append(np.squeeze(np.sqrt(np.sum((imgpoints[i] - imgpoints2)**2,axis=2))))

# all reprojection errors in absolute pixel values
errs_all = np.hstack(errs_all)
print("Mean error: {:.4f} px".format(errs_all.mean()))
fig = plt.figure(figsize=(12, 7))
ax = fig.add_subplot(1, 1 ,1)
ax.hist(errs_all, 20)
ax.set_title("Histogram of Reprojection Errors", fontsize=30)
ax.set_xlabel('Error in [pixels]', fontsize=30)
ax.set_ylabel('Counts', fontsize=30)
ax.grid()
```

```
Mean error: 0.1387 px
```




## Homography Quiz


## Homography Quiz


1. Prove that there exists a homogrpahy $H$ that satisfies:

$$p_1 \equiv Hp_2$$

   between the 2D points (in homogeneous coordinates) $p_1$ and $p_2$ in the images of a plane $\Pi$ captured by two $3 \times 4$ camera projection matrices $M_1$ and $M_2$, respectively. The symbol $\equiv$ stands for equality $up\ to\ scale$.

(Note: A degenerate case happens when the plan $\Pi$ contains both cameras' centers, in which case there are infinite choices of $H$ satisfying the equation. You can ignore this special case in your answer.)


## Homography Quiz


- Plane in 3D using homogeneous coordinates is given by:

$$n^T P = 0$$

  Where $n, P$ are homogeneous vectors (4 numbers each) and $n$ is the normal to the plane.

- Therefore, we can find a basis of 3 vectors $u_1, u_2, u_3$ in $R^4$, such that each point on the plane is given by:

$$P = \sum_{i=1}^{3} \alpha_i u_i$$

- The projection of 3D point $P$ to the $j^{th}$ image point $p_j$ is given by:

$$p_j = M_j P = \sum_{i=1}^{3} \alpha_i M_j u_i$$

## Homography Quiz

- If we denote $v_j^i = M_j u_i$ we get:

$$p_1 = \sum_{i=1}^{3} \alpha_i v_1^i$$

$$p_2 = \sum_{i=1}^{3} \alpha_i v_2^i$$

- Hence, the relation between the two points is a $3 \times 3$ matrix satisfying:

$$\begin{bmatrix} | & | & | \\ v_1^1 & v_1^2 & v_1^3 \\ | & | & | \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} | & | & | \\ v_2^1 & v_2^2 & v_2^3 \\ | & | & | \end{bmatrix}$$

## Homography Quiz

- Relation to camera calibration:
  - Recall that we have 11 degrees of freedom in $M$.
  - If all the calibration points are on a plane, we get at most 6 independent equations out of 3 pts.
  - Any 4th point will be a linear combination of the previous 3 on the plane.
  - Therefore, in estimating $M$ we can't rely on a single image of the chessboard.

## Homography Quiz

1. Prove that there exists a homography $H$ that satisfies the equation $p_1 = Hp_2$, given two cameras separated by a pure rotation. That is, for camera 1, $p_1 = K_1 [I|0] P$, and for camera 2, $p_2 = K_2 [R|0] P$. Note that $K_1$ and $K_2$ are the $3 \times 3$ intrinsic matrices of the cameras and are different. $I$ is $3 \times 3$ identity matrix, $0$ is a $3 \times 1$ zero vector and $P$ is a point in 3D space. $R$ is the $3 \times 3$ rotation matrix of the camera.

## Homography Quiz

- Since the last column is zero, we can see that:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R^{-1} K_2^{-1} p_2$$

- Substituting this in the second equation we get:

$$p_1 = K_1 R^{-1} K_2^{-1} p_2$$

- Therefore, the resulting homography is given by:

$$H = K_1 R^{-1} K_2^{-1}$$

## Homography Quiz

- Take away is 2 cameras differing only in rotation can't triangulate!

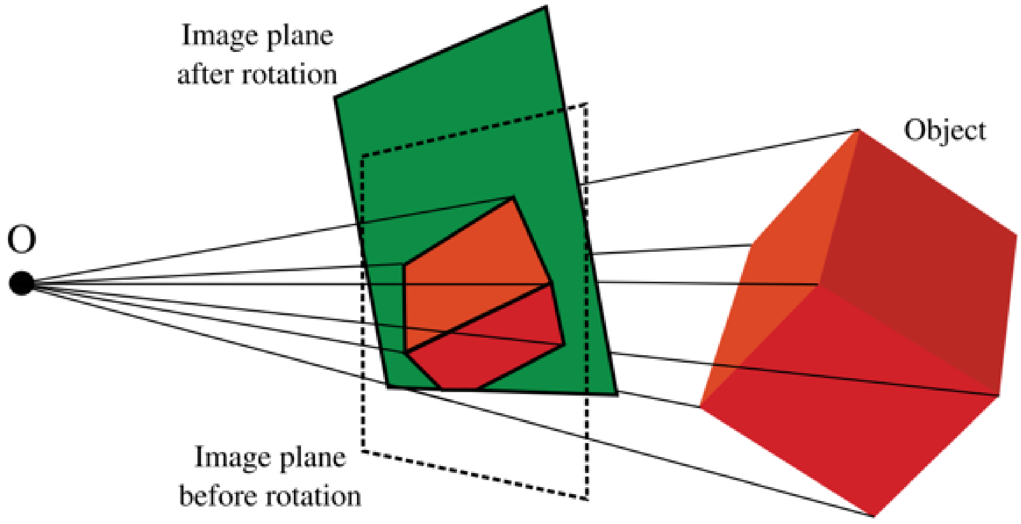



**Figure 15.14** Images under pure camera rotation. When the camera rotates but does not translate, the bundle of rays remains the same, but is cut by a different plane. It follows that the two images are related by a homography.

- Image Source - Prince.
- Remember where this was useful?
  - Panorama stitching! (There we did not care about recovering depth)

## Homography Quiz

1. Suppose that a camera is rotating about its center $C$, keeping the intrinsic parameters $K$ constant. Let $H$ be the homography that maps the view from one camera orientation to the view at a second orientation. Let $\theta$ be the angle of rotation between the two. Show that $H^2$ is the homography corresponding to a rotation of $2\theta$.

## Homography Quiz

- We have just shown that for such a scenario:

$$H_{2\to 1} = K_1 R_\theta^{-1} K_2^{-1}$$

$$H_{1\to 2} = K_2 R_\theta K_1^{-1}$$

- Applying the constraint $K_1 = K_2 \equiv K$ gets us:

$$H_{1\to 2} = K R_\theta K^{-1}$$

- Applying $H_{1\to 2}$ twice gets us:

$$H_{1\to 2}^2 = K R_\theta K^{-1} K R_\theta K^{-1} = K R_\theta R_\theta K^{-1}$$

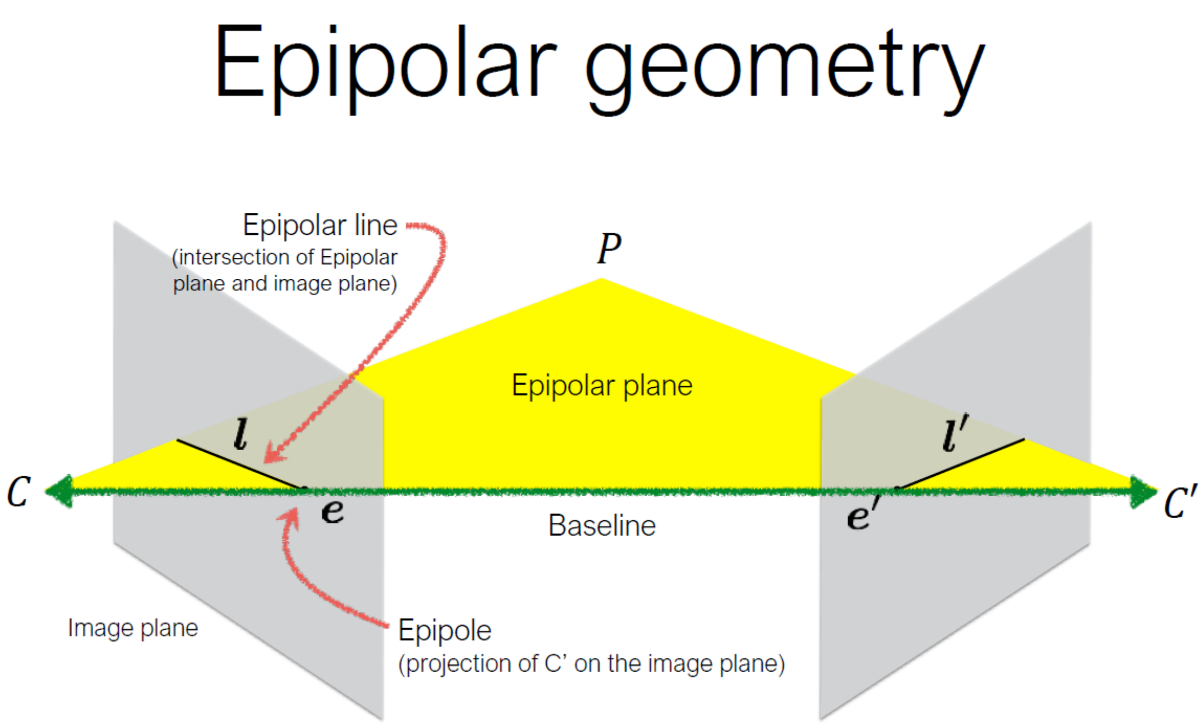
- Since $R_\theta R_\theta = R_{2\theta}$, we indeed get:

$$H^2_{1\to 2} = K R_{2\theta} K^{-1}$$

Which is a homography that corresponds to a rotation of $2\theta$.

# Epipolar Geometry

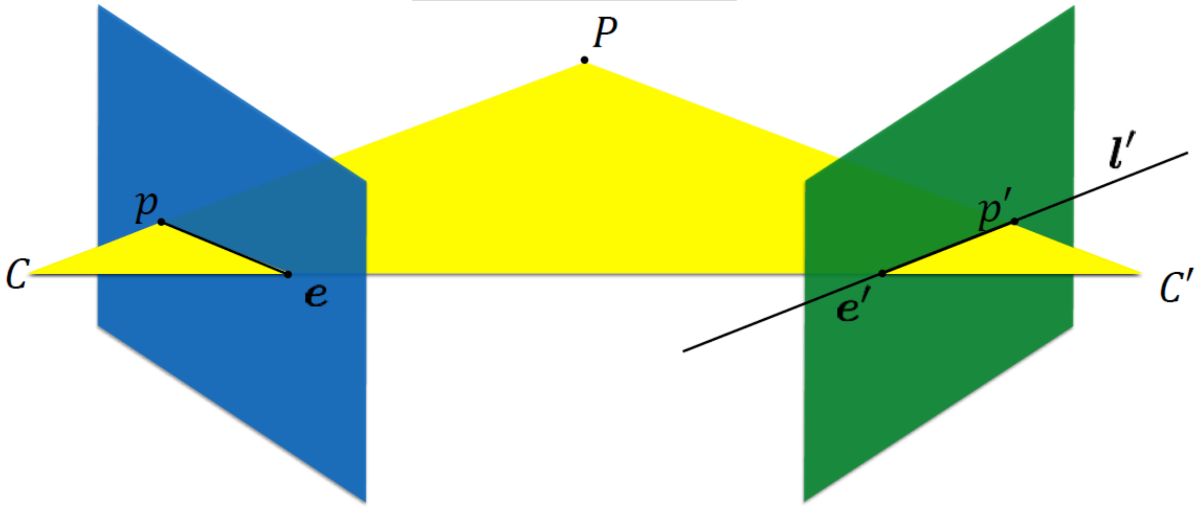
## Epipolar Lingo




## Essential Matrix

Given a point in one image,
multiplying by the **essential matrix** will tell us
the **epipolar line** in the second view.

$$\boxed{Ep = l'}$$






# Essential Matrix vs Homography

*What's the difference between the essential matrix and a homography?*

They are both 3 x 3 matrices but …

$$Ep = l'$$

Essential matrix maps a **point** to a **line**

$$Hp = p'$$

Homography maps a **point** to a **point**

*When can we use a homography? And when only an essential matrix?*

Homography applies only for planer scenes

# The essential matrix

$$\boxed{(p')^T E p = 0}$$

$$E = R[dC_\times]$$

Can also show $E = [t_\times]R$

with $t = -RdC$

**Essential Matrix Properties**

# properties of the E matrix

| | | |
|---|---|---|
| Longuet-Higgins equation | $p'^T E p = 0$ | |
| Epipolar lines | $p^T l = 0$<br>$l' = Ep$ | $p'^T l' = 0$<br>$l = E^T p'$ |
| Epipoles | $e'^T E = 0$ | $Ee = 0$ |

(Since for every p: $e'^T l' = e'^T E p = 0 \Rightarrow e'^T E = 0$)

**Fundamental Matrix**

$$\widehat{p'}^{T} E \hat{p} = 0$$

The Essential matrix operates on image points expressed in
**normalized coordinates**
(points have been aligned (normalized) to camera coordinates)

$$\widehat{p'} = K'^{-1} p' \qquad \underset{\text{camera point}}{\hat{p}} = \underset{\text{image point}}{K^{-1} p}$$

Writing out the epipolar constraint in terms of image coordinates

$$p'^{T} K'^{-T} E K^{-1} p = 0$$

$$p'^{T} (K'^{-T} E K^{-1}) p = 0$$

$$p'^{T} F p = 0$$

## Fundamental Geometric Interpretation

Breaking down the fundamental matrix

$$\mathbf{F} = \mathbf{K}'^{-\top} \mathbf{E} \mathbf{K}^{-1}$$

$$\mathbf{F} = \mathbf{K}'^{-\top} [\mathbf{t}_{\times}] \mathbf{R} \mathbf{K}^{-1}$$

Depends on both intrinsic and extrinsic parameters

## Fundamental Matrix Properties

# properties of the ~~E~~ F matrix

| | | |
|---|---|---|
| Longuet-Higgins equation | $p'^T F p = 0$ | |
| Epipolar lines | $p^T l = 0$ | $p'^T l' = 0$ |
| | $l' = F\ p$ | $l = F^T p'$ |
| Epipoles | $e'^T F\ = 0$ | $F\ e = 0$ |

## Fundamental Matrix Estimation

- Assume we are given 2D to 2D M matched image points:

$$\{p_i, p_i'\}_{i=1}^{M}$$

- Each cosspondence should satisfy:

$$p_i^T F p_i' = 0 \leftrightarrow \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^T \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = 0$$

- How to solve?
  - The 8-point algorithm $\leftrightarrow$ arrange into homogeneous linear equations and SVD..

## Fundamental Matrix Estimation

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_Mx'_M & x_My'_M & x_M & y_Mx'_M & y_My'_M & y_M & x'_M & y'_M & 1 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \\ f_8 \\ f_9 \end{bmatrix} = \mathbf{0}$$

- How much matches needed to solve?
- How much did we need for Homography?

## Fundamental Matrix Demo

In [6]:

```
# start by detecting features and matching them with SIFT
img1 = cv2.imread('./assets/left.jpg',0)  #queryimage # left image
img2 = cv2.imread('./assets/right.jpg',0) #trainimage # right image
sift = cv2.SIFT_create()
# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1,None)
kp2, des2 = sift.detectAndCompute(img2,None)
# FLANN parameters
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params,search_params)
matches = flann.knnMatch(des1,des2,k=2)
pts1 = []
pts2 = []
# ratio test as per Lowe's paper
for i,(m,n) in enumerate(matches):
    if m.distance < 0.8*n.distance:
        pts2.append(kp2[m.trainIdx].pt)
        pts1.append(kp1[m.queryIdx].pt)
```

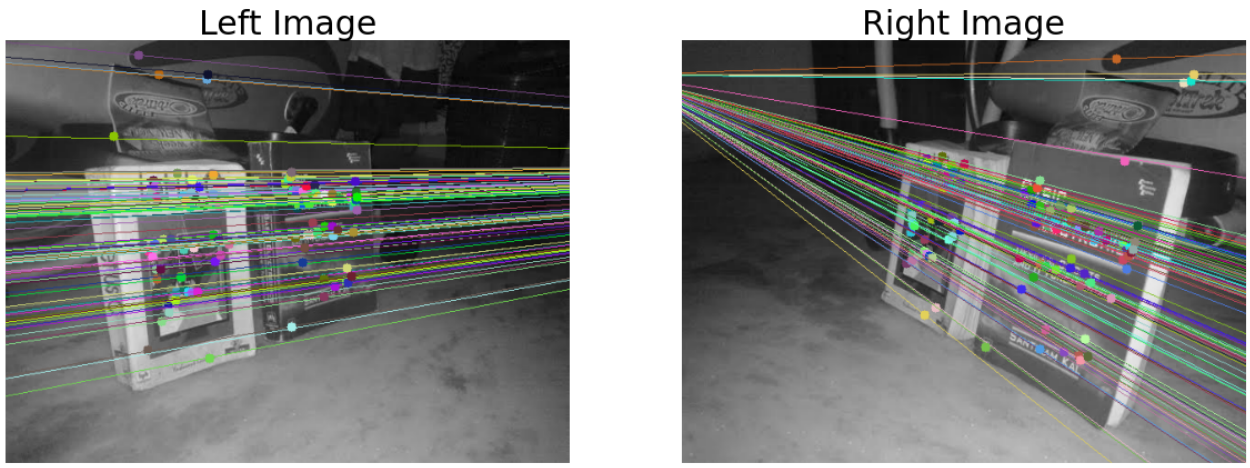
In [7]:

```
# estimating the fundamental matrix
pts1 = np.int32(pts1)
pts2 = np.int32(pts2)
F, mask = cv2.findFundamentalMat(pts1,pts2,cv2.FM_LMEDS)
# We select only inlier points
pts1 = pts1[mask.ravel()==1]
pts2 = pts2[mask.ravel()==1]
```

In [8]:

```
# drawing epilines
def drawlines(img1,img2,lines,pts1,pts2):
    ''' img1 - image on which we draw the epilines for the points in img2
        lines - corresponding epilines '''
    r,c = img1.shape
    img1 = cv2.cvtColor(img1,cv2.COLOR_GRAY2BGR)
    img2 = cv2.cvtColor(img2,cv2.COLOR_GRAY2BGR)
    for r,pt1,pt2 in zip(lines,pts1,pts2):
        color = tuple(np.random.randint(0,255,3).tolist())
        x0,y0 = map(int, [0, -r[2]/r[1] ])
        x1,y1 = map(int, [c, -(r[2]+r[0]*c)/r[1] ])
        img1 = cv2.line(img1, (x0,y0), (x1,y1), color,1)
        img1 = cv2.circle(img1,tuple(pt1),5,color,-1)
        img2 = cv2.circle(img2,tuple(pt2),5,color,-1)
    return img1,img2
```

In [9]:
```python
# Find epilines corresponding to points in right image (second image) and
# drawing its lines on left image
lines1 = cv2.computeCorrespondEpilines(pts2.reshape(-1,1,2), 2,F)
lines1 = lines1.reshape(-1,3)
img5,img6 = drawlines(img1,img2,lines1,pts1,pts2)
# Find epilines corresponding to points in left image (first image) and
# drawing its lines on right image
lines2 = cv2.computeCorrespondEpilines(pts1.reshape(-1,1,2), 1,F)
lines2 = lines2.reshape(-1,3)
img3,img4 = drawlines(img2,img1,lines2,pts2,pts1)
plt.figure(figsize=(20,10))
plt.subplot(121)
plt.imshow(img5)
plt.axis('off')
plt.title('Left Image', fontsize=30)
plt.subplot(122)
plt.imshow(img3)
plt.axis('off')
plt.title('Right Image', fontsize=30)
plt.show()
```




## Recommended Videos

---

### Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

### Video By Subject

- Epipolar and Essential matrix - William Hoff
- Fundamental Matrix - William Hoff
- The Fundamental Matrix Song - Daniel Wedge

## Credits

---

- EE 046746 Spring 2020 - Dahlia Urbach
- Slides - Elad Osherov (Technion), Simon Lucey (CMU)
- Multiple View Geometry in Computer Vision - Hartley and Zisserman - Chapter 6
- Least–squares Solution of Homogeneous Equations - Center for Machine Perception - Tomas Svoboda
- Computer vision: models, learning and inference , Simon J.D. Prince - Chapter 15
- Computer Vision: Algorithms and Applications - Richard Szeliski - Sections 2.1.5, 6.2. , 7.1, 7.2, 11.1

- Icons from Icon8.com - https://icons8.com