CS294-137
Lecture 6: Fundamentals of Computer Vision

Allen Y. Yang

Fall, 2017

Berkeley
UNIVERSITY OF CALIFORNIA

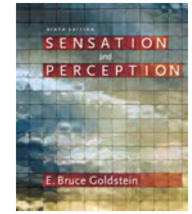
Course Schedule Update

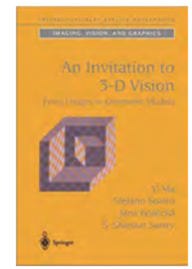
Week 1 (8-23): Introduction and Capstone Options
Week 2 (8-30): Human Perception in the Context of VR
Week 3 (9-6): Basic Unity3D/VR Programming Workshop
Week 4 (9-13): Course project proposal presentation
Week 5 (9-20): Optics and Display technologies
Week 6 (9-27): Vision Accommodation and Vergence
Week 7 (10-4): Computer Vision related topics
Week 8 (10-11): Computer Graphics related topics
****
Week 9: (10-18) Telemedicine (Ruzena Bajcsy/Gregorij Korillo)
Week 10 (10-25): Gaming (Jack McCauley)
Week 11 (11-1): VR Film Making (Richard Hernandez)
Week 12 (11-8): AR/VR in Arts & Design (Ted Selker)
Week 13 (11-15): Computational Imaging for VR (Ren Ng)
Week 14 (11-22): No class
Week 15 (11-29): Final project presentation
Week 16 (12-6): Final project presentation

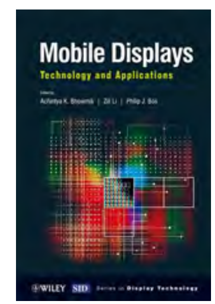
Berkeley
UNIVERSITY OF CALIFORNIA

Recommended Reading Material

- **Perception:** Sensation and Perception
  by Bruce Goldstein
- **Virtual Reality:** Virtual Reality
  By Steven LaValle (and checkout his YouTube lectures)
- **Computer Graphics:** Fundamentals of CG
  by Peter Shirley
- **Computer Vision**: An Invitation to 3-D Vision
  by Yi Ma, et al.
- **Display: Mobile Displays**
  by Achin Bhowmik, et al.
- **AR/VR Market Research:** Virtual & Augmented Reality, understanding the race for the next computing platform
  by Goldman Sachs

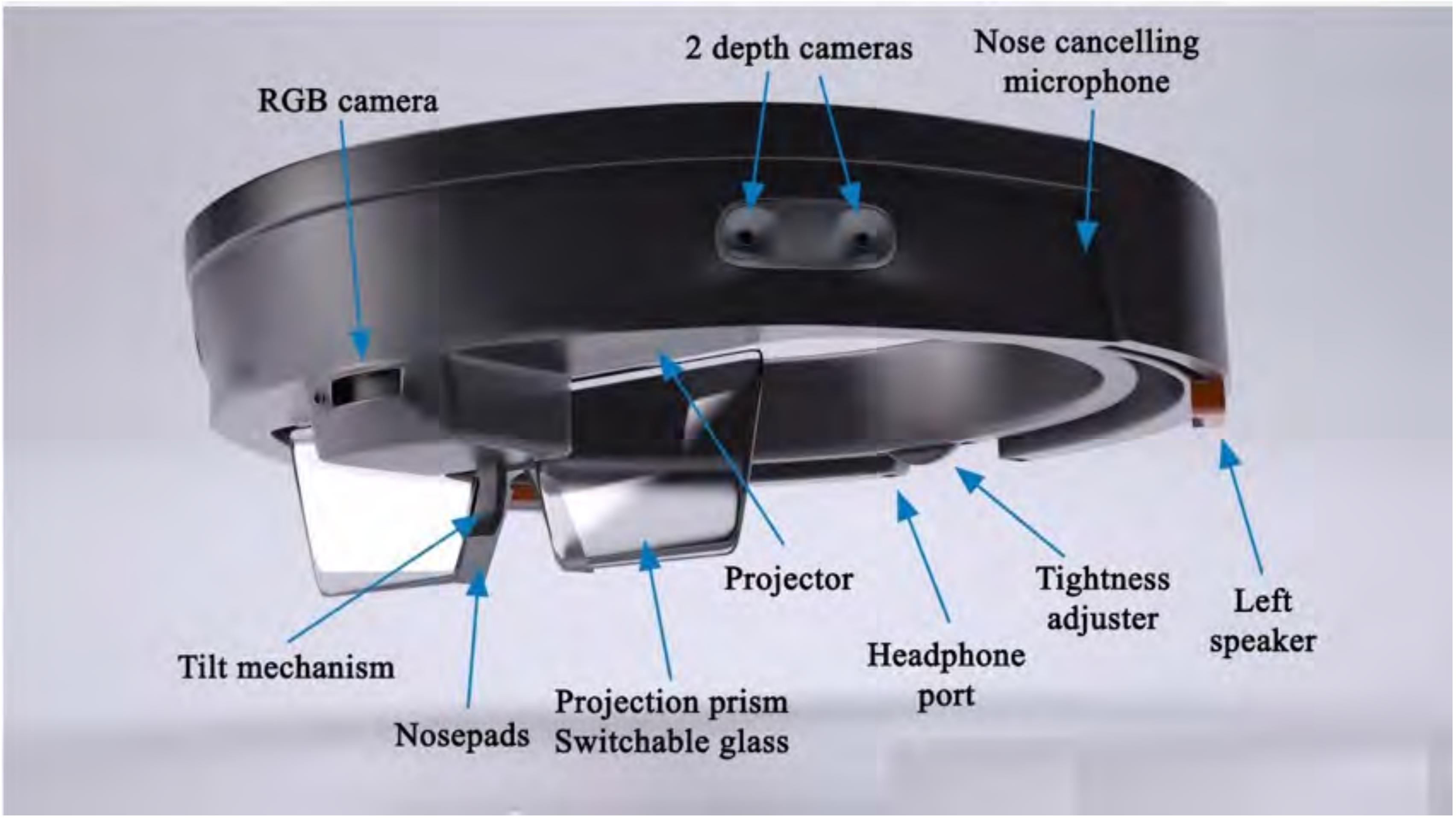






Berkeley
UNIVERSITY OF CALIFORNIA

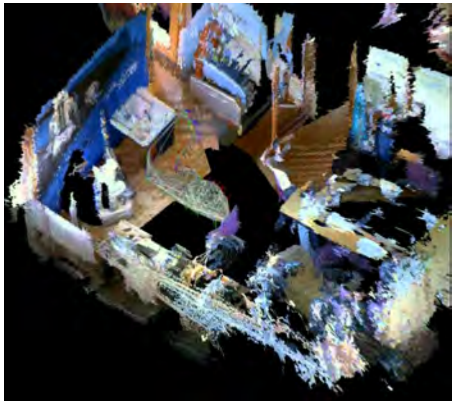
Anatomy of an AR Device: HoloLens



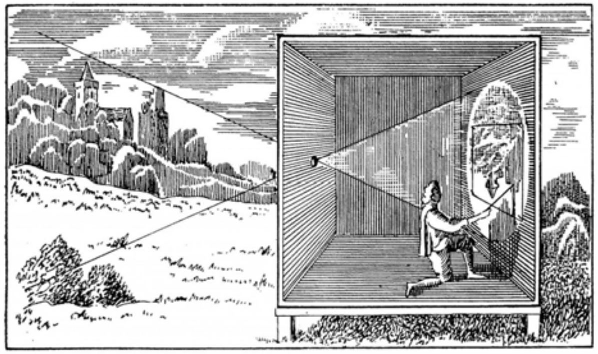
Including perception & display, end-to-end latency not exceeding 16ms (60 fps)

Berkeley
UNIVERSITY OF CALIFORNIA

What can Computer Vision do?



Fundamental Problems of Computer Vision



Camera Obscura, circa 400BC

Holmes stereoscope, 1861

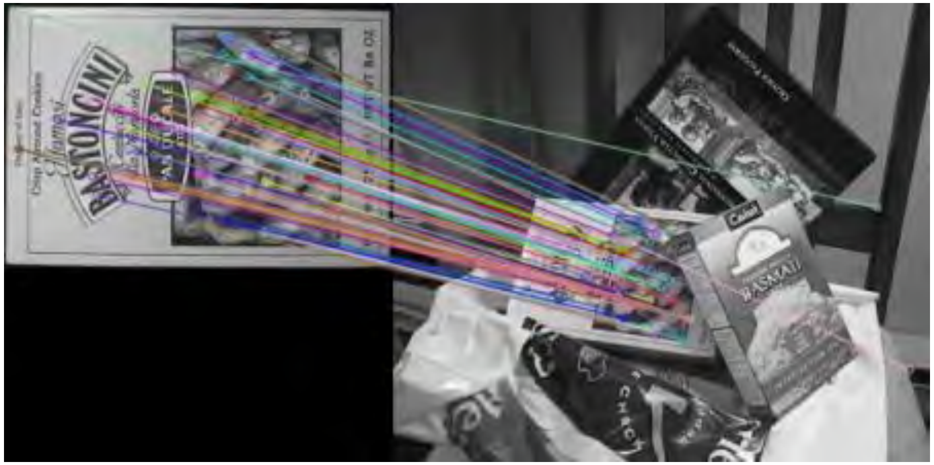
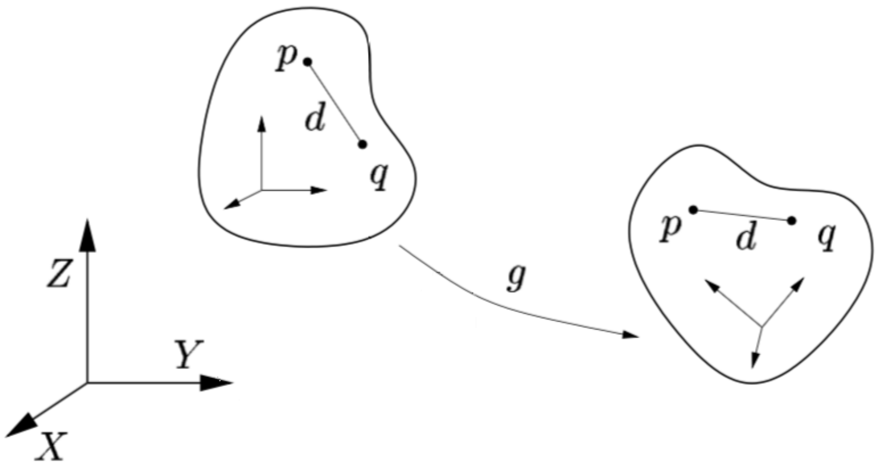


Image Matching using Robust Features

Berkeley
UNIVERSITY OF CALIFORNIA

Part I: Basic Linear Algebra

Rigid Body Motion



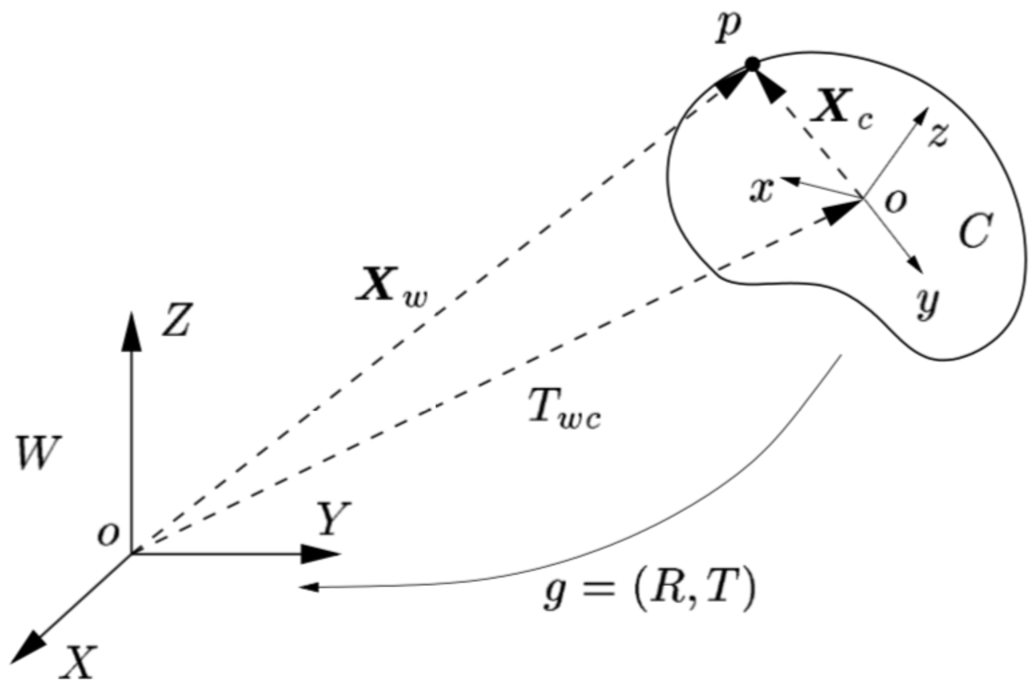
Thus, if $\boldsymbol{X}(t)$ and $\boldsymbol{Y}(t)$ are the coordinates of any two points p and q on the object, respectively, the distance between them is constant:

$$\|\boldsymbol{X}(t) - \boldsymbol{Y}(t)\| \equiv \text{constant}, \quad \forall\, t \in \mathbb{R}. \tag{2.3}$$

A *rigid-body motion* (or rigid-body transformation) is then a family of maps that describe how the coordinates of every point on a rigid object change in time while satisfying (2.3). We denote such a map by

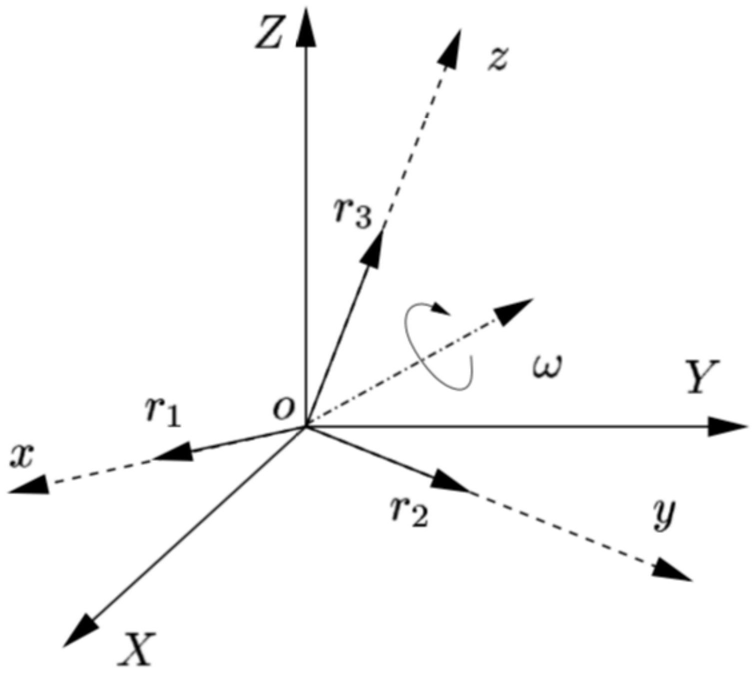
$$g(t): \ \mathbb{R}^3 \to \mathbb{R}^3; \quad \boldsymbol{X} \mapsto g(t)(\boldsymbol{X}).$$

Change of Coordinate Systems



$$\boldsymbol{X}_w = R_{wc}\boldsymbol{X}_c + T_{wc}.$$

Special Orthogonal Group



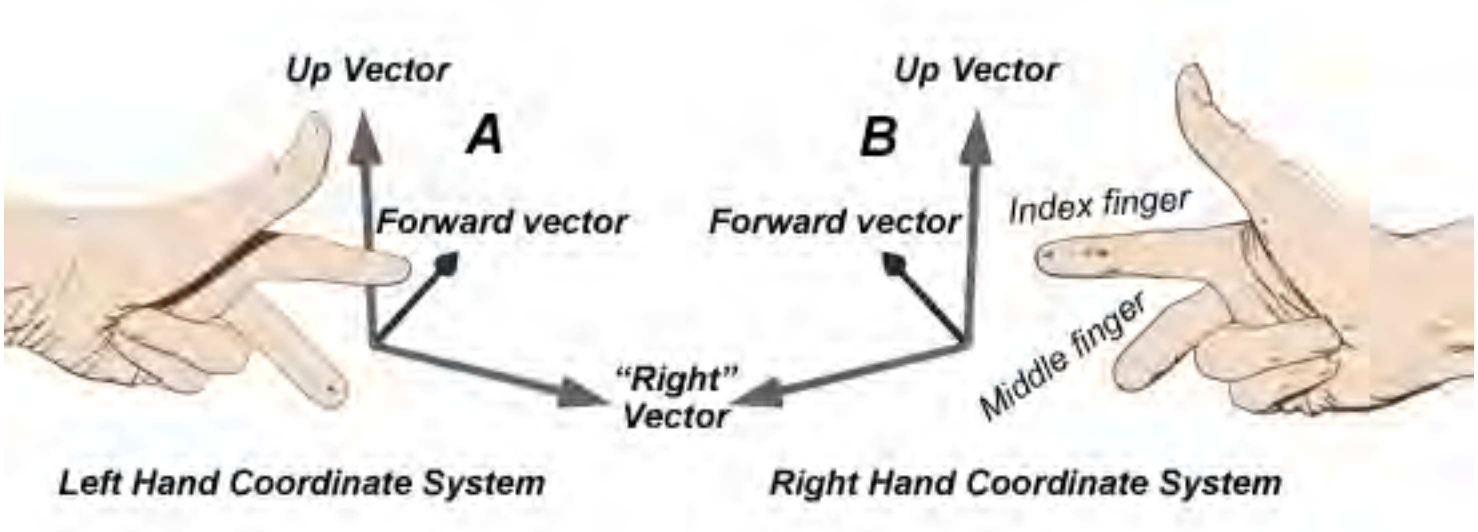
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{SO(3) \doteq \left\{ R \in \mathbb{R}^{3\times 3} \mid R^T R = I, \det(R) = +1 \right\}.}$$

Be Aware of Left-Handed or Right-Handed Coordinate Systems



Homogeneous Coordinates & Special Euclidean Group SE(3)

$$\bar{\boldsymbol{X}}_w = \begin{bmatrix} \boldsymbol{X}_w \\ 1 \end{bmatrix} = \begin{bmatrix} R_{wc} & T_{wc} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_c \\ 1 \end{bmatrix} \doteq \bar{g}_{wc} \bar{\boldsymbol{X}}_c,$$

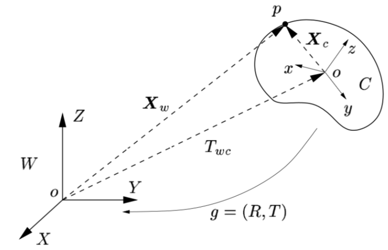
$$\boxed{SE(3) \doteq \left\{ \bar{g} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \middle| R \in SO(3), T \in \mathbb{R}^3 \right\} \quad \subset \mathbb{R}^{4\times 4}.}$$

Concatenation: $\bar{g}_1 \bar{g}_2 = \begin{bmatrix} R_1 & T_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2 & T_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1 R_2 & R_1 T_2 + T_1 \\ 0 & 1 \end{bmatrix} \quad \in SE(3)$

Inverse: $\bar{g}^{-1} = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} R^T & -R^T T \\ 0 & 1 \end{bmatrix} \quad \in SE(3).$

Berkeley
UNIVERSITY OF CALIFORNIA

Estimation of (*R*, *T*)



Translation Only: $$Y_i = X_i + T \Leftrightarrow Y_{centroid} \doteq \frac{\sum Y_i}{n} = X_{centroid} + T$$

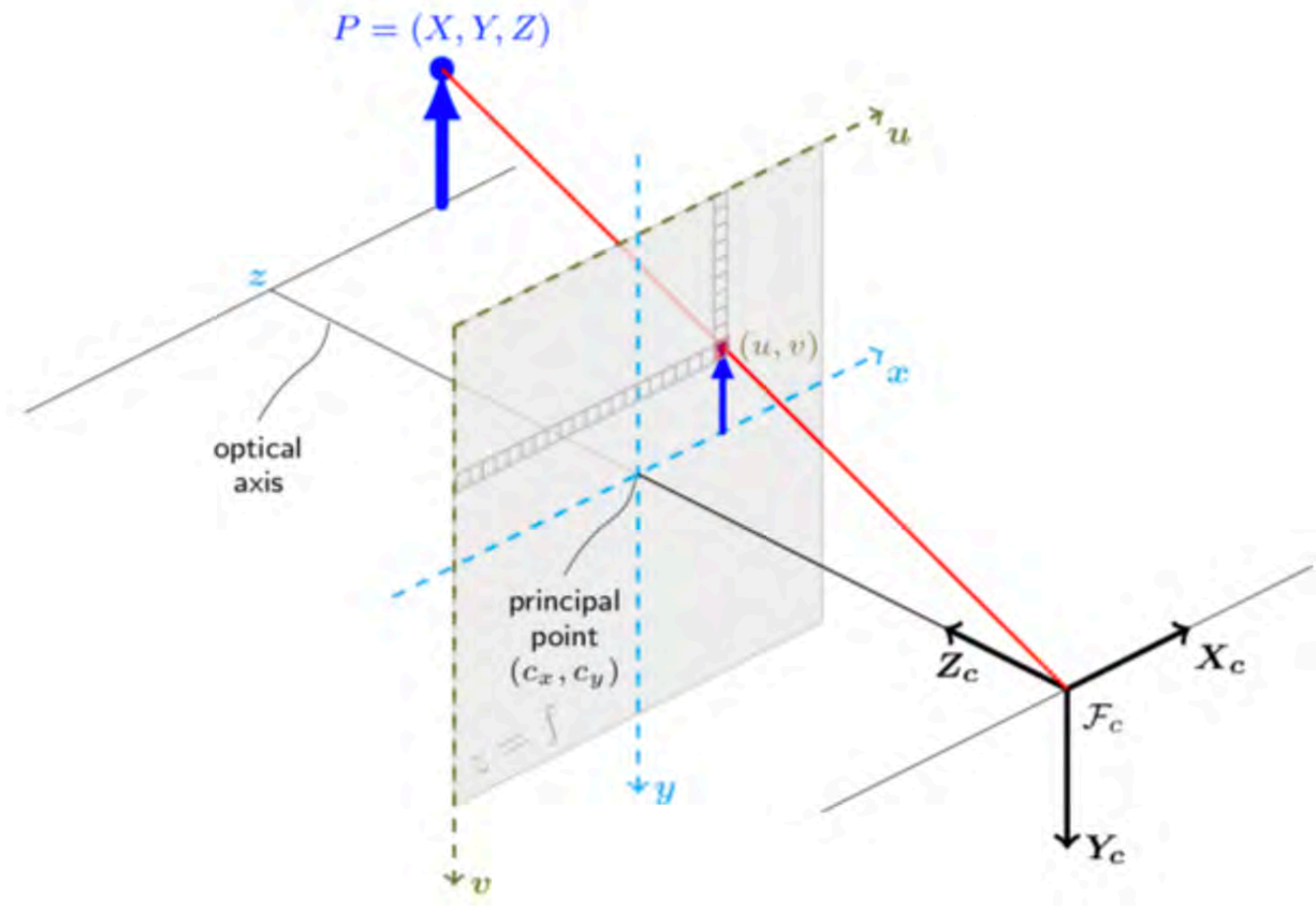
Rotation Only:

$$
\begin{aligned}
& Y_i = RX_i \\
& \Rightarrow R^* = \arg \min_{R \in SO(3)} \sum_i \|RX_i - Y_i\|^2 \\
& \text{since} \quad \sum_i \|RX_i - Y_i\|^2 = \sum_i (X_i^T X_i - 2Y_i^T RX_i + Y_i^T Y_i) \\
& \Rightarrow R^* = \arg \min_{R \in SO(3)} \sum_i (-Y_i^T RX_i) = \arg \max_{R \in SO(3)} \sum_i (Y_i^T RX_i) \\
& \quad \text{further} \quad \sum_i (Y_i^T RX_i) = \operatorname{tr}\left(Y^T RX\right) = \operatorname{tr}\left(RXY^T\right) \\
& \text{let} \quad \mathrm{SVD}(XY^T) = U\Sigma V^T \\
& \Rightarrow R^* = \arg\max \operatorname{tr}\left(RU\Sigma V^T\right) = \arg\max \operatorname{tr}\left(\Sigma V^T RU\right) \\
& \text{since} \quad \Sigma \text{ is diagnal with non-negative values and } V^T RU \in \mathrm{SO}(3) \\
& \Rightarrow \max \operatorname{tr}\left(\Sigma V^T RU\right) = \operatorname{tr}(\Sigma) \text{ iff } V^T RU = I \\
& \Rightarrow R^* = VU^T.
\end{aligned}
$$

Don’t forget to check right-handedness!

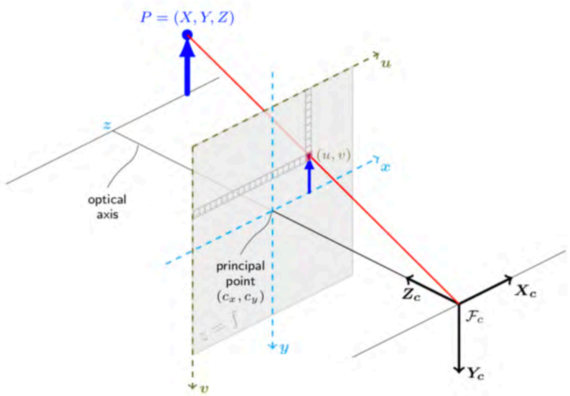
Berkeley
UNIVERSITY OF CALIFORNIA

Part II: Geometry of Pinhole Camera



* OpenCV Online Documentation

Pinhole Camera Parameters



$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t$$

$$x' = x/z$$
$$y' = y/z$$
$$u = f_x * x' + c_x$$
$$v = f_y * y' + c_y$$

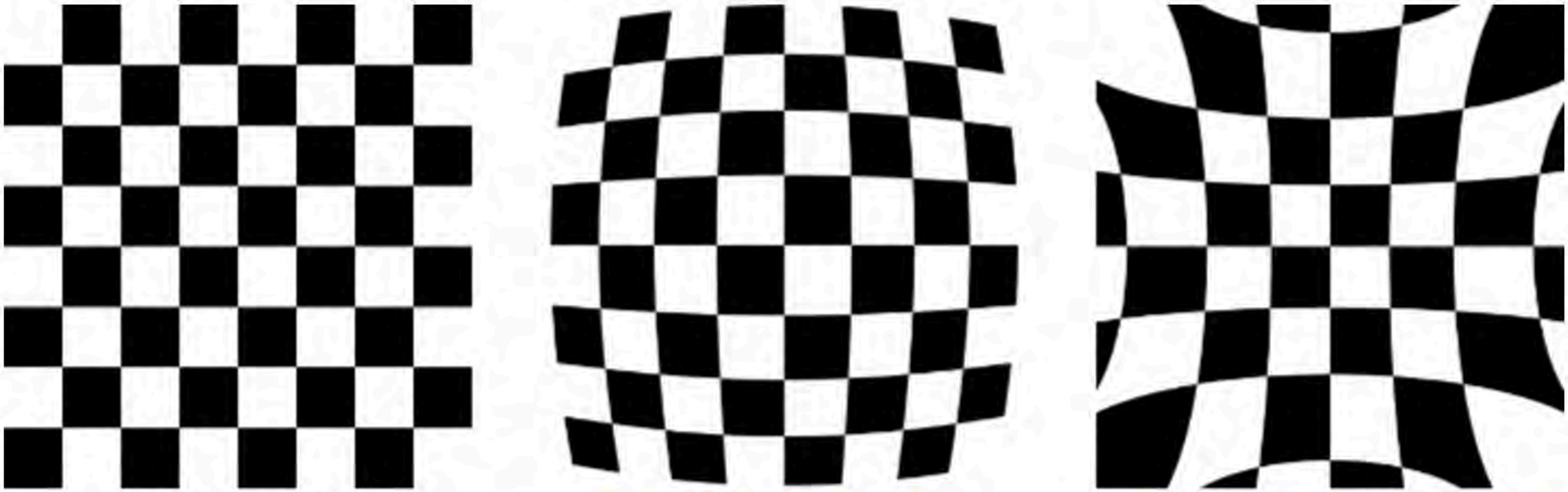
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Intrinsic parameters Extrinsic parameters

where:

- (X, Y, Z) are the coordinates of a 3D point in the world coordinate space
- (u, v) are the coordinates of the projection point in pixels
- A is a camera matrix, or a matrix of intrinsic parameters
- (cx, cy) is a principal point that is usually at the image center
- fx, fy are the focal lengths expressed in pixel units.

Camera Calibration using OpenCV



No distortion

Positive radial distortion (Barrel distortion)

Negative radial distortion (Pincushion distortion)

calibrateCamera

Finds the camera intrinsic and extrinsic parameters from several views of a calibration pattern.

C++: double `calibrateCamera`(InputArrayOfArrays **objectPoints**, InputArrayOfArrays **imagePoints**, Size **imageSize**, InputOutputArray **cameraMatrix**, InputOutputArray **distCoeffs**, OutputArrayOfArrays **rvecs**, OutputArrayOfArrays **tvecs**, int **flags**=0, TermCriteria **criteria**=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, DBL_EPSILON))

Berkeley
UNIVERSITY OF CALIFORNIA

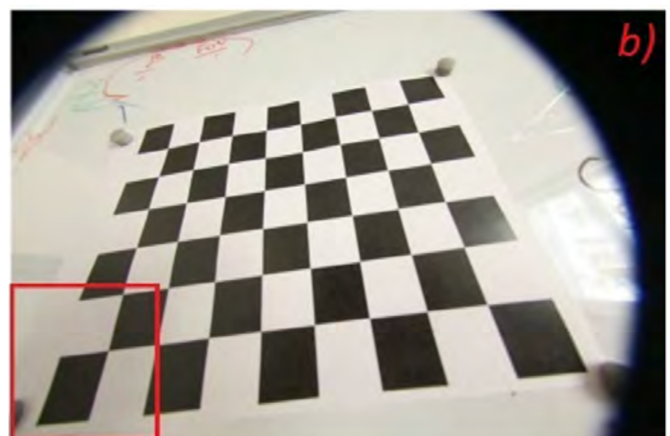
Camera Distortion Rectification

fisheye::undistortImage

Transforms an image to compensate for fisheye lens distortion.

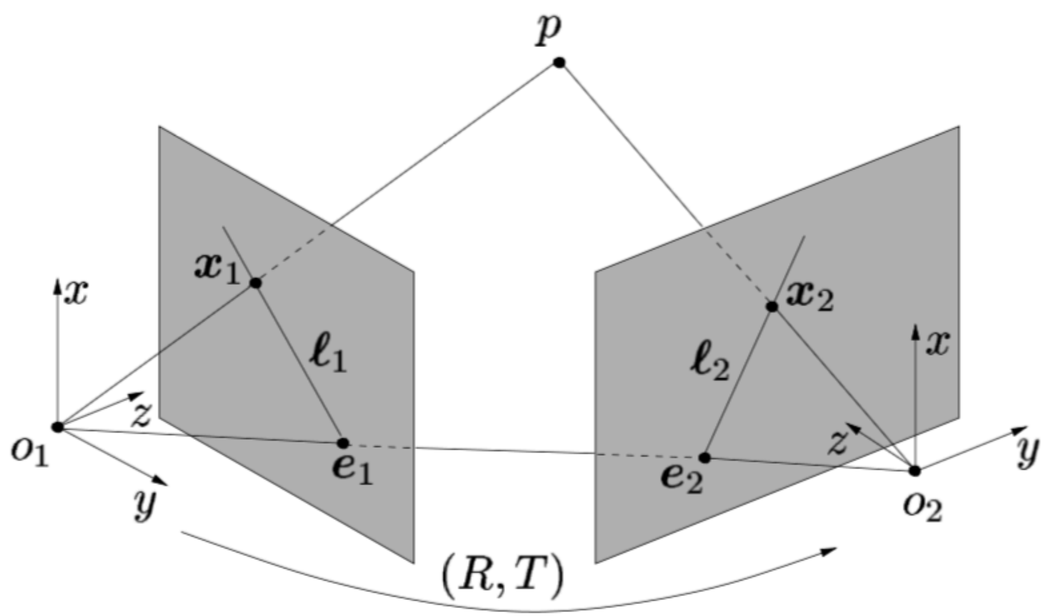
C++: void `fisheye::undistortImage`(InputArray **distorted**, OutputArray **undistorted**, InputArray **K**, InputArray **D**, InputArray **Knew**=cv::noArray(), const Size& **new_size**=Size())

Parameters:
- **distorted** – image with fisheye lens distortion.
- **K** – Camera matrix $K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$.
- **D** – Input vector of distortion coefficients (k_1, k_2, k_3, k_4).
- **Knew** – Camera matrix of the distorted image. By default, it is the identity matrix but you may additionally scale and shift the result by using a different matrix.
- **undistorted** – Output image with compensated fisheye lens distortion.



Berkeley
UNIVERSITY OF CALIFORNIA

Part III: Structure from Motion

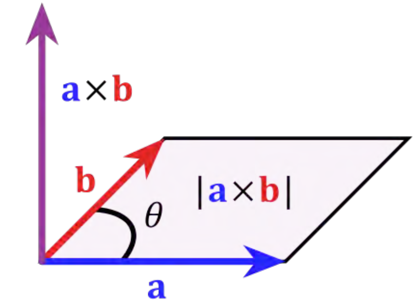


SfM Problem

Assume multiple 2D images of 3D points and their correspondence are known, estimate their 3D locations and the transformations (R, T).

Berkeley
UNIVERSITY OF CALIFORNIA

Definition: hat operator



$$\mathbf{u}\times\mathbf{v} = (u_1\mathbf{i}+u_2\mathbf{j}+u_3\mathbf{k})\times(v_1\mathbf{i}+v_2\mathbf{j}+v_3\mathbf{k}) = \begin{pmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

$$\Rightarrow \hat{\mathbf{u}} \doteq \begin{pmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{pmatrix} \in \Re^{3\times 3}$$

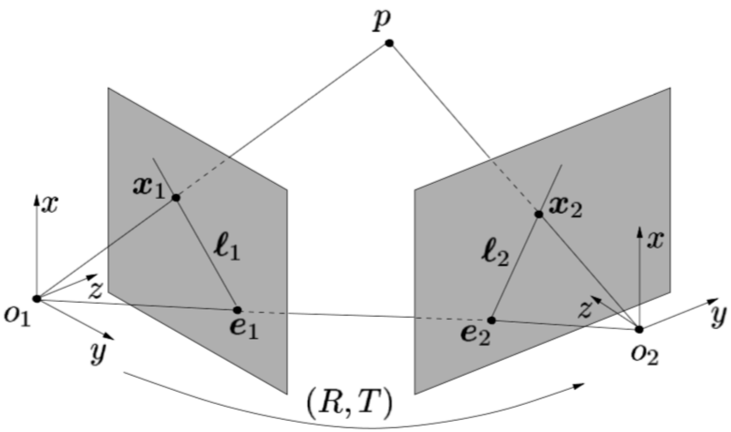
$$\Rightarrow \mathbf{u}\times\mathbf{v} = \hat{\mathbf{u}}\mathbf{v}$$

Quick Facts:

1. $a\mathbf{u}\times\mathbf{u} = 0$
2. $\mathbf{u}\times\mathbf{v} = -\mathbf{v}\times\mathbf{u}$
3. $\mathbf{u}\times\mathbf{v}\cdot\mathbf{u} = \mathbf{u}\times\mathbf{v}\cdot\mathbf{v} = 0$
4. $\mathbf{u}\cdot(\mathbf{v}\times\mathbf{w}) = \mathbf{v}\cdot(\mathbf{w}\times\mathbf{u}) = \mathbf{w}\cdot(\mathbf{u}\times\mathbf{v})$

Berkeley
UNIVERSITY OF CALIFORNIA

Epipolar Constraint

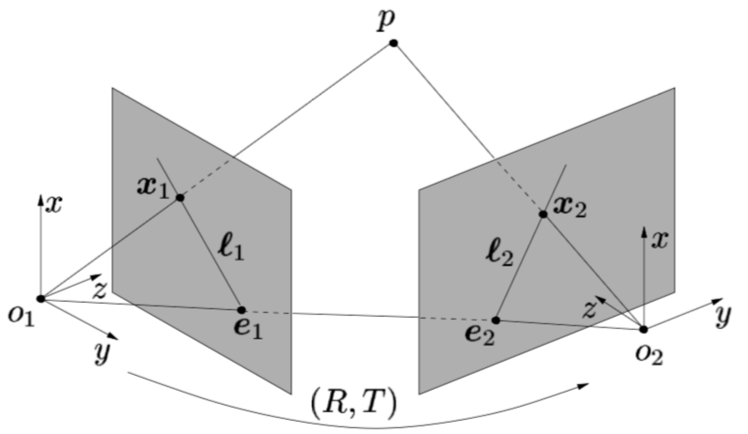


$$
\begin{aligned}
& X_2 = RX_1 + T \\
\Rightarrow\ & \lambda_2 \mathbf{x}_2 = \lambda_1 R \mathbf{x}_1 + T \\
\Rightarrow\ & \lambda_2 \hat{T} \mathbf{x}_2 = \lambda_2 \hat{T} R \mathbf{x}_1 \\
\Rightarrow\ & \mathbf{x}_2^T \hat{T} \mathbf{x}_2 = 0 = \mathbf{x}_2^T \hat{T} R \mathbf{x}_1
\end{aligned}
$$

The matrix is called the *essential matrix*.

$$E \doteq \widehat{T} R \quad \in \mathbb{R}^{3\times 3}$$

Properties of Epipolar Constraint



Conditions on the epipoles

$$\boldsymbol{e}_2 \sim T \; and \; \boldsymbol{e}_1 \sim R^T T.$$

$$\boldsymbol{e}_2^T E = 0, \quad E\boldsymbol{e}_1 = 0.$$

Conditions on the epipolar lines (by co-images)

In each image, both the image point and the epipole lie on the epipolar line

$$\boldsymbol{\ell}_i^T \boldsymbol{e}_i = 0, \quad \boldsymbol{\ell}_i^T \boldsymbol{x}_i = 0, \quad i = 1, 2. \tag{5.5}$$

Berkeley
UNIVERSITY OF CALIFORNIA

Estimation of Essential Matrix

Let $E = \widehat{T}R$ be the essential matrix associated with the epipolar constraint (5.2). The entries of this 3×3 matrix are denoted by

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \quad \in \mathbb{R}^{3\times 3} \tag{5.10}$$

and stacked into a vector $E^s \in \mathbb{R}^9$, which is typically referred to as the *stacked version* of the matrix E (Appendix A.1.3):

$$E^s \doteq [e_{11}, e_{21}, e_{31}, e_{12}, e_{22}, e_{32}, e_{13}, e_{23}, e_{33}]^T \quad \in \mathbb{R}^9.$$

The inverse operation from E^s to its matrix version is then called *unstacking*. We further denote the *Kronecker product* $\otimes$ (also see Appendix A.1.3) of two vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ by

$$\boldsymbol{a} \doteq \boldsymbol{x}_1 \otimes \boldsymbol{x}_2. \tag{5.11}$$

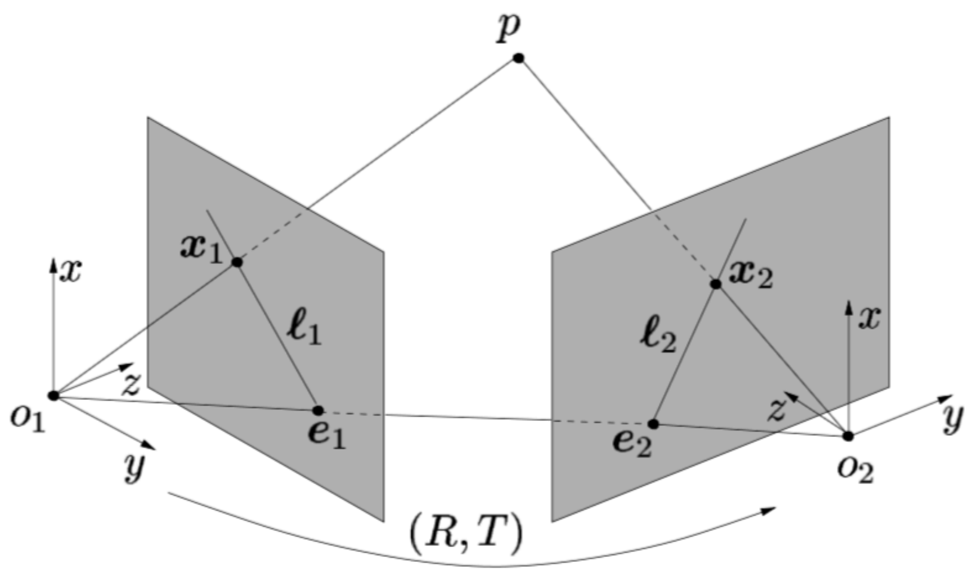
Or, more specifically, if $\boldsymbol{x}_1 = [x_1, y_1, z_1]^T \in \mathbb{R}^3$ and $\boldsymbol{x}_2 = [x_2, y_2, z_2]^T \in \mathbb{R}^3$, then

$$\boldsymbol{a} = [x_1x_2, x_1y_2, x_1z_2, y_1x_2, y_1y_2, y_1z_2, z_1x_2, z_1y_2, z_1z_2]^T \quad \in \mathbb{R}^9. \tag{5.12}$$

Since the epipolar constraint $\boldsymbol{x}_2^T E \boldsymbol{x}_1 = 0$ is linear in the entries of E, using the above notation we can rewrite it as the inner product of $\boldsymbol{a}$ and E^s:

$$\boxed{\boldsymbol{a}^T E^s = 0.}$$

Enforcing Essential Matrix



Theorem 5.5 (Characterization of the essential matrix). *A nonzero matrix* $E \in \mathbb{R}^{3\times 3}$ *is an essential matrix* if and only if E *has a singular value decomposition (SVD)* $E = U\Sigma V^T$ *with*

$$\Sigma = diag\{\sigma, \sigma, 0\}$$

for some $\sigma \in \mathbb{R}_+$ *and* $U, V \in SO(3)$.

8-Point or 7-Point Algorithm

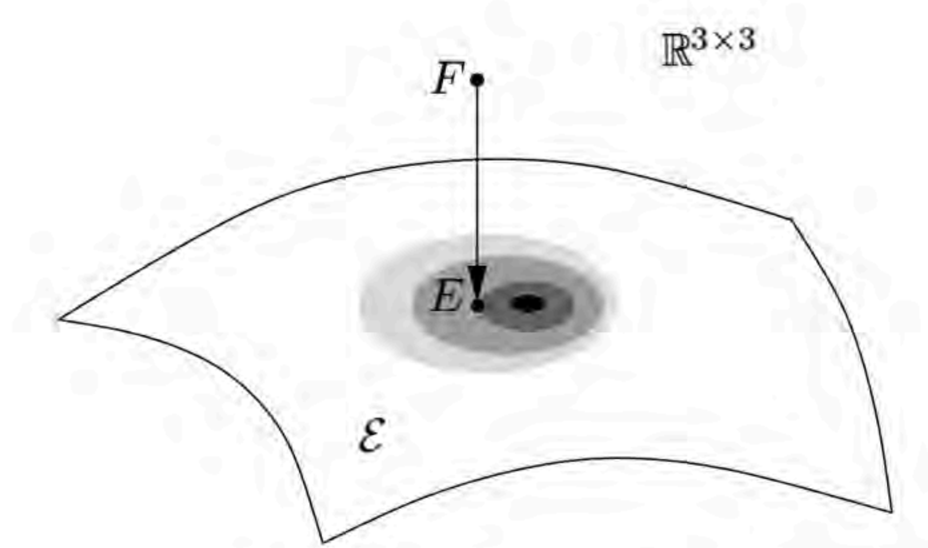


Figure 5.5. Among all points in the essential space $\mathcal{E} \subset \mathbb{R}^{3\times 3}$, E has the shortest Frobenius distance to F. However, the least-square error may not be the smallest for so-obtained E among all points in $\mathcal{E}$.

Theorem 5.9 (Projection onto the essential space). *Given a real matrix $F \in \mathbb{R}^{3\times 3}$ with SVD $F = U\mathrm{diag}\{\lambda_1, \lambda_2, \lambda_3\}V^T$ with $U, V \in SO(3)$, $\lambda_1 \geq \lambda_2 \geq \lambda_3$, then the essential matrix $E \in \mathcal{E}$ that minimizes the error $\|E - F\|_f^2$ is given by $E = U\mathrm{diag}\{\sigma, \sigma, 0\}V^T$ with $\sigma = (\lambda_1 + \lambda_2)/2$. The subscript "$f$" indicates the Frobenius norm of a matrix. This is the square norm of the sum of the squares of all the entries of the matrix (see Appendix A).*

What if correspondence has error: Random Sample Consensus (RANSAC)

C++: Mat `findEssentialMat`(InputArray **points1**, InputArray **points2**, double **focal**=1.0, Point2d **pp**=Point2d(0, 0), int **method**=RANSAC, double **prob**=0.999, double **threshold**=1.0, OutputArray **mask**=noArray())

Parameters:

- **points1** – Array of `N (N >= 5)` 2D points from the first image. The point coordinates should be floating-point (single or double precision).
- **points2** – Array of the second image points of the same size and format as `points1` .
- **focal** – focal length of the camera. Note that this function assumes that `points1` and `points2` are feature points from cameras with same focal length and principle point.
- **pp** – principle point of the camera.
- **method** –

  Method for computing a fundamental matrix.

  - **RANSAC** for the RANSAC algorithm.
  - **MEDS** for the LMedS algorithm.
- **threshold** – Parameter used for RANSAC. It is the maximum distance from a point to an epipolar line in pixels, beyond which the point is considered an outlier and is not used for computing the final fundamental matrix. It can be set to something like 1-3, depending on the accuracy of the point localization, image resolution, and the image noise.
- **prob** – Parameter used for the RANSAC or LMedS methods only. It specifies a desirable level of confidence (probability) that the estimated matrix is correct.
- **mask** – Output array of N elements, every element of which is set to 0 for outliers and to 1 for the other points. The array is computed only in the RANSAC and LMedS methods.

Decomposition of E Matrix

Theorem 5.7 (Pose recovery from the essential matrix). *There exist exactly two relative poses* (R, T) *with* $R \in SO(3)$ *and* $T \in \mathbb{R}^3$ *corresponding to a nonzero essential matrix* $E \in \mathcal{E}$.

Proof. Assume that $(R_1, T_1) \in SE(3)$ and $(R_2, T_2) \in SE(3)$ are both solutions for the equation $\widehat{T}R = E$. Then we have $\widehat{T}_1 R_1 = \widehat{T}_2 R_2$. This yields $\widehat{T}_1 = \widehat{T}_2 R_2 R_1^T$. Since $\widehat{T}_1, \widehat{T}_2$ are both skew-symmetric matrices and $R_2 R_1^T$ is a rotation matrix, from the preceding lemma, we have that either $(R_2, T_2) = (R_1, T_1)$ or $(R_2, T_2) = (e^{\widehat{u}_1 \pi} R_1, -T_1)$ with $u_1 = T_1/\|T_1\|$. Therefore, given an essential matrix E there are exactly *two* pairs of (R, T) such that $\widehat{T}R = E$. Further, if E has the SVD: $E = U\Sigma V^T$ with $U, V \in SO(3)$, the following formulae give the two distinct solutions (recall that $R_Z(\theta) \doteq e^{\widehat{e}_3 \theta}$ with $e_3 = [0, 0, 1]^T \in \mathbb{R}^3$)

$$\boxed{\begin{array}{rcl} (\widehat{T}_1, R_1) & = & \left(UR_Z(+\frac{\pi}{2})\Sigma U^T, UR_Z^T(+\frac{\pi}{2})V^T\right), \\ (\widehat{T}_2, R_2) & = & \left(UR_Z(-\frac{\pi}{2})\Sigma U^T, UR_Z^T(-\frac{\pi}{2})V^T\right). \end{array}} \tag{5.9}$$

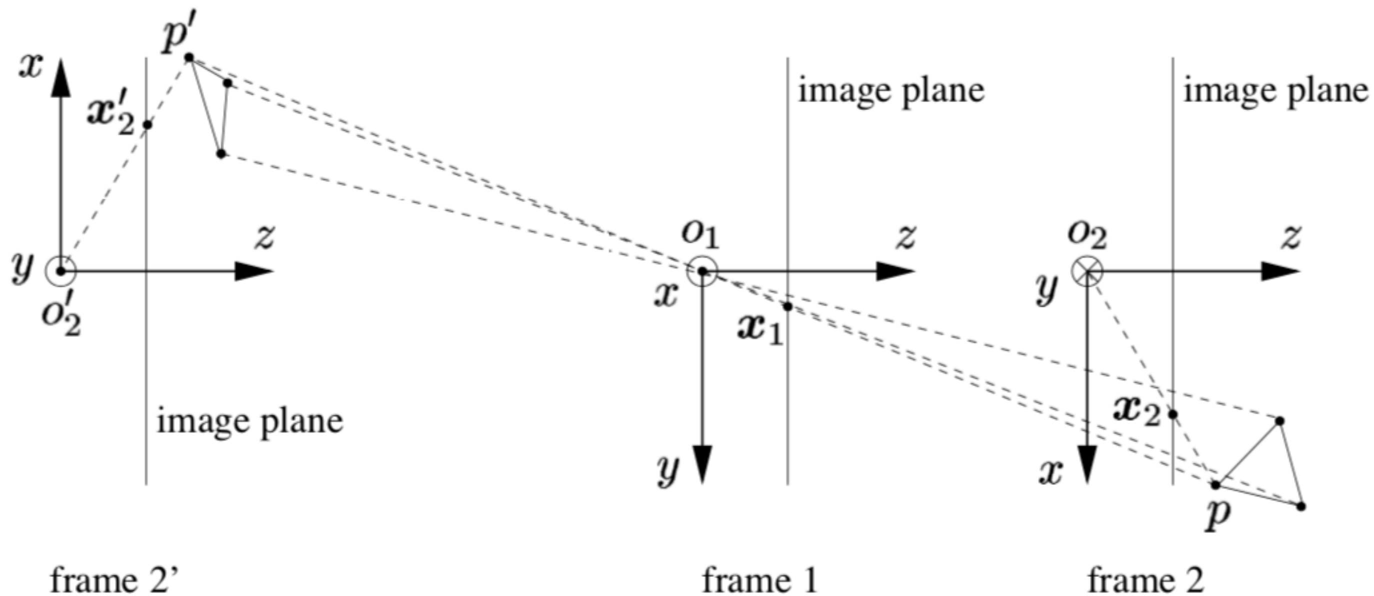
Berkeley
UNIVERSITY OF CALIFORNIA

Decomposition of E Matrix

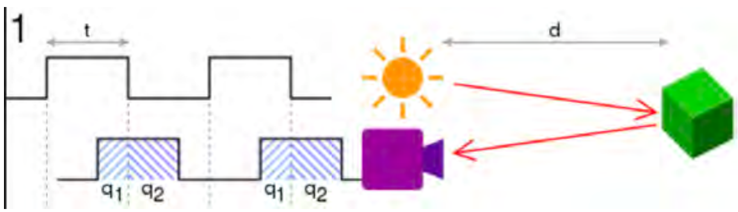
Example 5.8 (Two solutions to an essential matrix). It is immediate to verify that $\widehat{e_3} R_Z\left(+\frac{\pi}{2}\right) = \widehat{-e_3} R_Z\left(-\frac{\pi}{2}\right)$, since

$$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

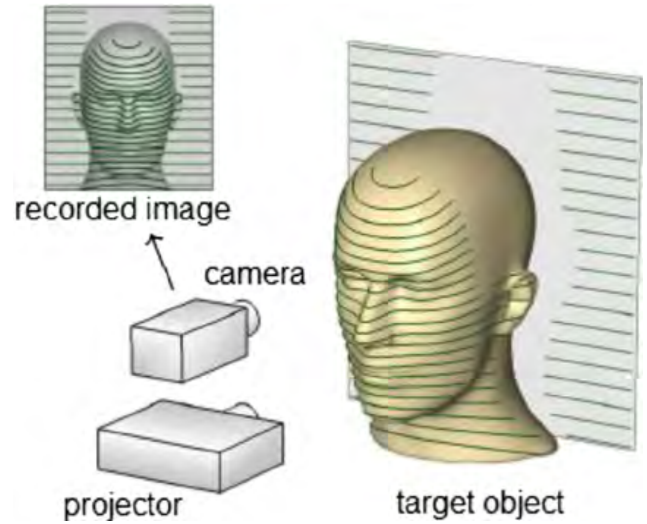
These two solutions together are usually referred to as a "twisted pair," due to the manner in which the two solutions are related geometrically, as illustrated in Figure 5.3. A physically correct solution can be chosen by enforcing that the reconstructed points be visible, i.e. they have positive depth. We discuss this issue further in Exercise 5.11. ■



About Depth Cameras

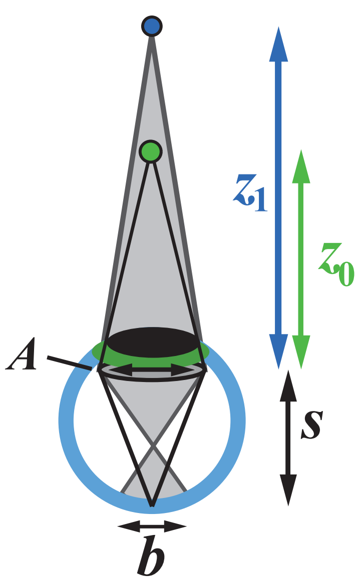


Time of Flight



Structured Light

Blur:



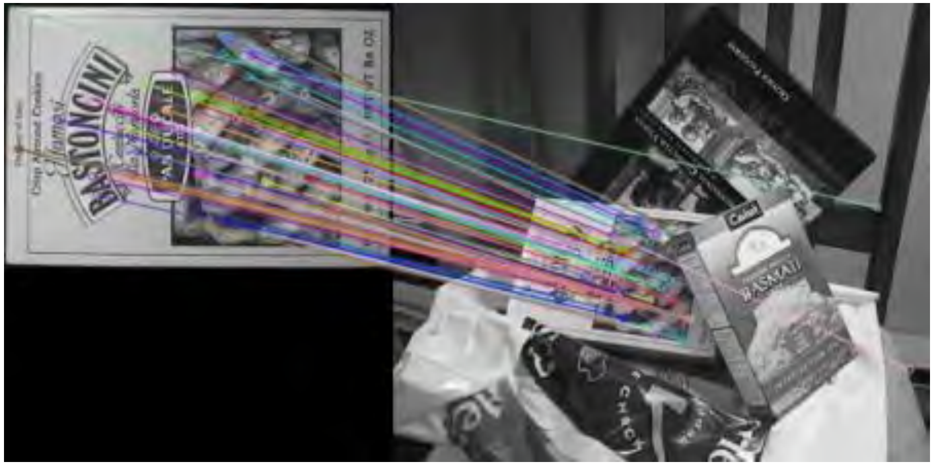
Depth from Defocus

Light Field Camera

Berkeley
UNIVERSITY OF CALIFORNIA

Part IV: Feature Matching

Features in images are not just 0-dim abstract points, their local appearance can be used to improve matching across images



SIFT (Scale-Invariant Feature Transform)
Step 1: Feature Detector

What is a corner point?

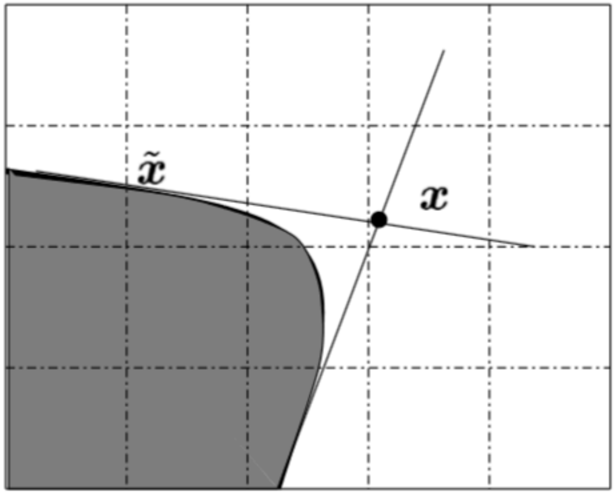


Figure 4.5. A corner feature $\boldsymbol{x}$ is the virtual intersection of local edges (within a window).

Berkeley
UNIVERSITY OF CALIFORNIA

SIFT (Scale-Invariant Feature Transform)
Step 1: Feature Detector

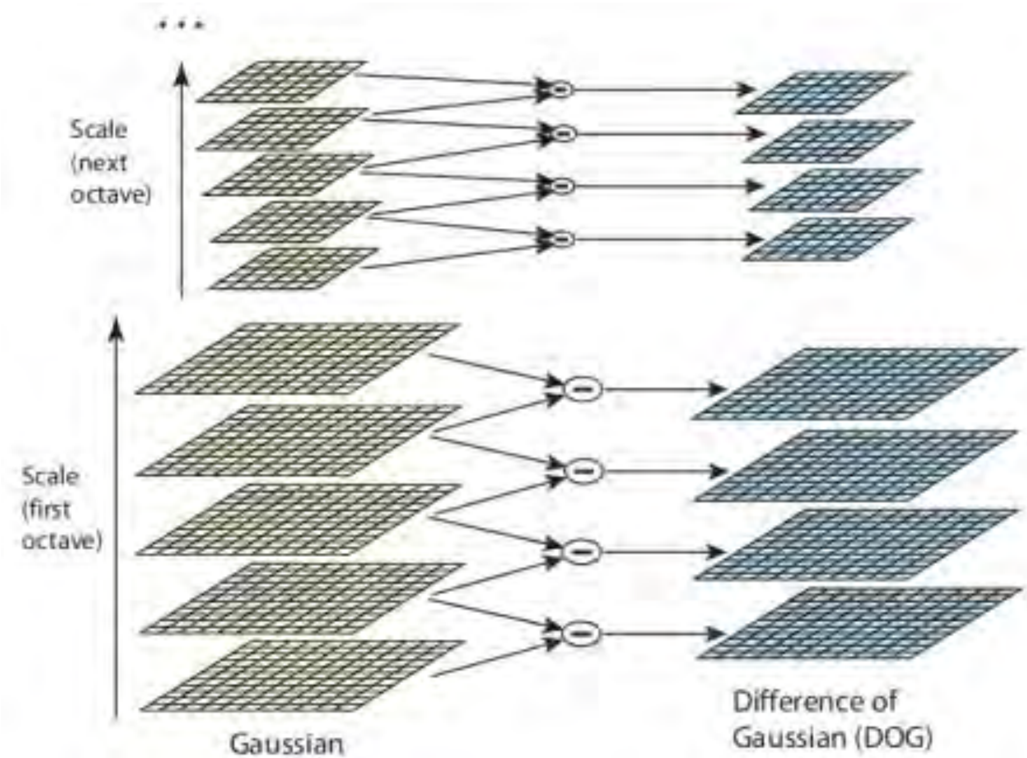
Algorithm 4.2 (Corner detector).

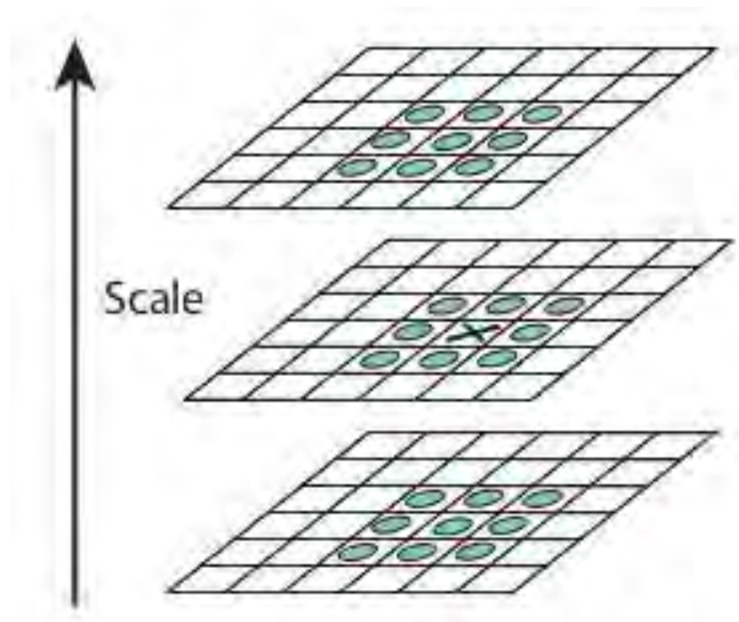
Given an image $I(x, y)$, follow the steps to detect whether a given pixel (x, y) is a corner feature:

- set a threshold $\tau \in \mathbb{R}$ and a window W of fixed size, and compute the image gradient (I_x, I_y) using the filters given in Appendix 4.A;
- at all pixels in the window W around (x, y) compute the matrix
$$G = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}; \tag{4.30}$$
- if the smallest singular value $\sigma_2(G)$ is bigger than the prefixed threshold τ, then mark the pixel as a feature (or corner) point.

Berkeley
UNIVERSITY OF CALIFORNIA

David Lowe's Solution: Difference of Gaussian





Typically, detection of SIFT combines both corner detection and DoG detection

Berkeley
UNIVERSITY OF CALIFORNIA

* David Lowe, Distinctive image features from scale-invariant keypoints, IJCV 2004

SIFT (Scale-Invariant Feature Transform) Step 2: Feature Descriptor

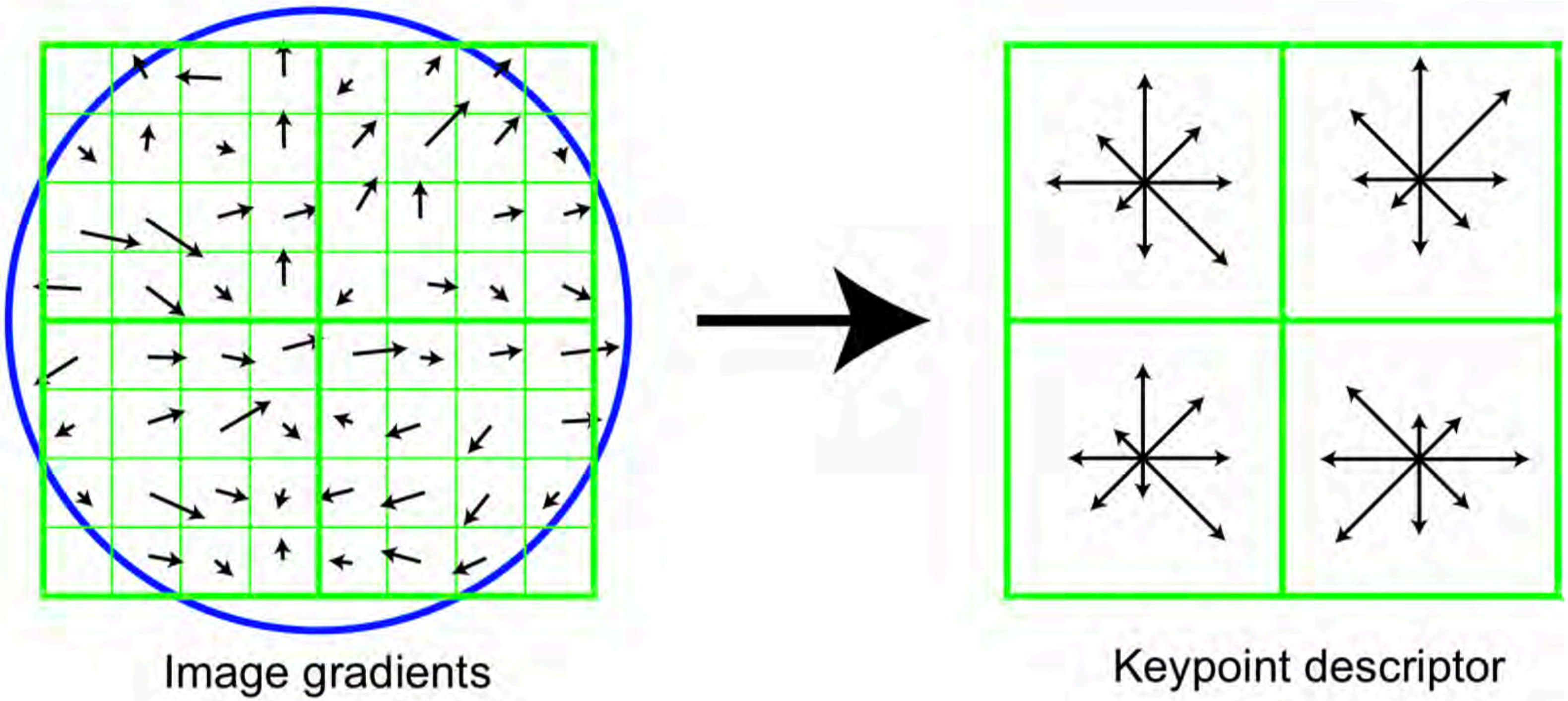
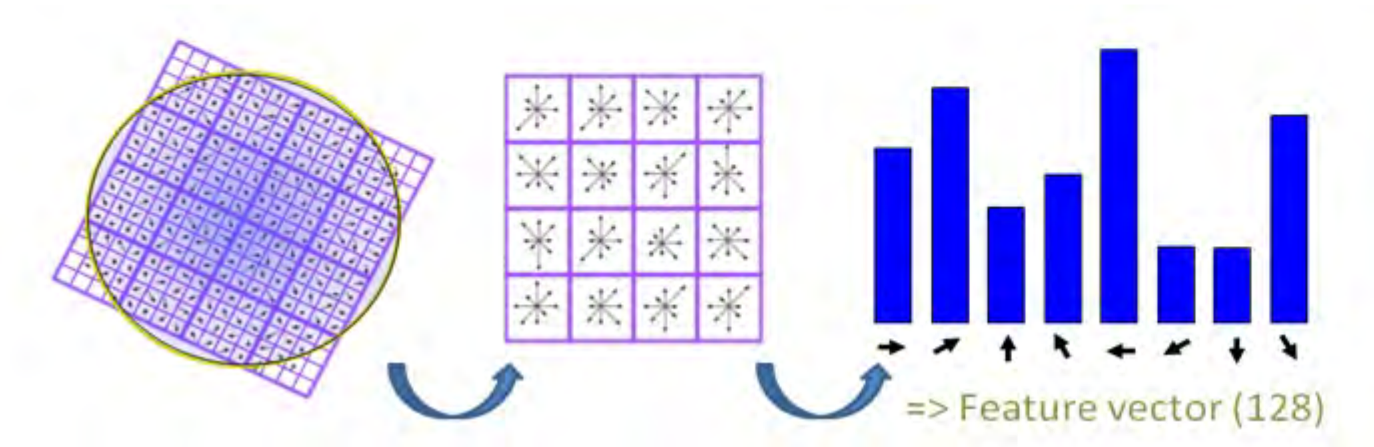


Figure 7: A keypoint descriptor is created by first computing the gradient magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a 2x2 descriptor array computed from an 8x8 set of samples, whereas the experiments in this paper use 4x4 descriptors computed from a 16x16 sample array.

SIFT (Scale-Invariant Feature Transform)
Step 3: Histogram Matching



Berkeley
UNIVERSITY OF CALIFORNIA

SIFT (Scale-Invariant Feature Transform)
Step 3: Histogram Matching

- OpenCV implements the function compareHist to perform a comparison. It also offers 4 different metrics to compute the matching:

  a. **Correlation (CV_COMP_CORREL)**

  $$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

  where

  $$\bar{H_k} = \frac{1}{N} \sum_J H_k(J)$$

  and N is the total number of histogram bins.

  b. **Chi-Square (CV_COMP_CHISQR)**

  $$d(H_1, H_2) = \sum_I \frac{\left(H_1(I) - H_2(I)\right)^2}{H_1(I)}$$

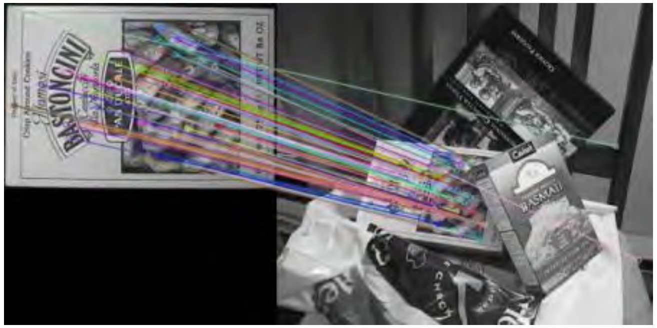
  c. **Intersection (method=CV_COMP_INTERSECT)**

  $$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

  d. **Bhattacharyya distance (CV_COMP_BHATTACHARYYA)**

  $$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\bar{H_1} \bar{H_2} N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

OpenCV Sample Code



```
// detecting keypoints
SurfFeatureDetector detector(400);
vector<KeyPoint> keypoints1, keypoints2;
detector.detect(img1, keypoints1);
detector.detect(img2, keypoints2);

// computing descriptors
SurfDescriptorExtractor extractor;
Mat descriptors1, descriptors2;
extractor.compute(img1, keypoints1, descriptors1);
extractor.compute(img2, keypoints2, descriptors2);

// matching descriptors
BFMatcher matcher(NORM_L2);
vector<DMatch> matches;
matcher.match(descriptors1, descriptors2, matches);

// drawing the results
namedWindow("matches", 1);
Mat img_matches;
drawMatches(img1, keypoints1, img2, keypoints2, matches, img_matches);
imshow("matches", img_matches);
waitKey(0);
```