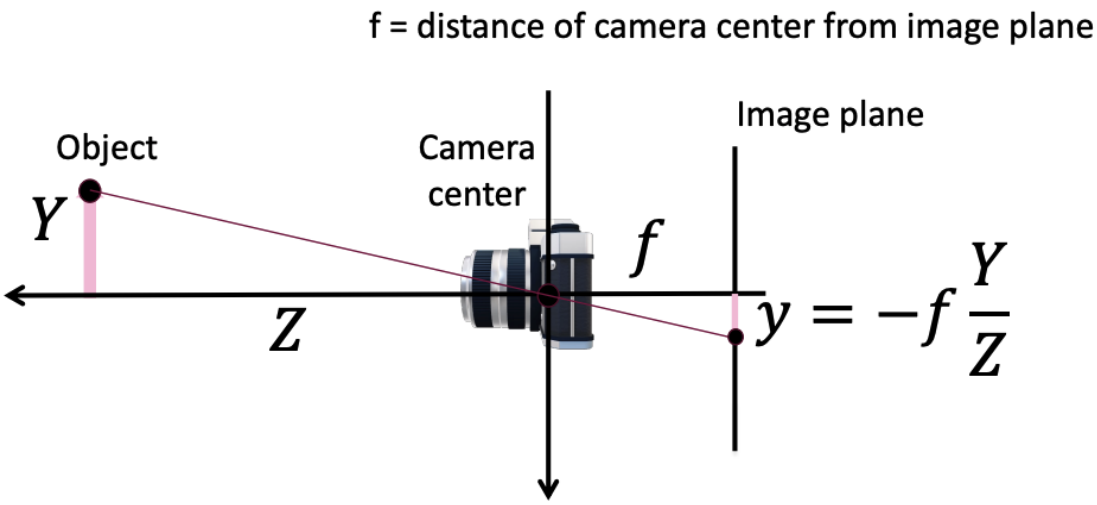
Notes on Geometric Computer Vision

Lecture 01 - Pinhole Camera

- Can model a camera as pinhole camera
  - Good model as long as the aperture is small



  - Can move image plane in front of camera to remove negative sign
- Perspective Projective Equations

$$x = f\frac{X}{Z}$$

$$y = f\frac{Y}{Z}$$

- Properties preserved by Perspective Projections
  - Only straightness/collinearity
  - Not shape, parallelness, lengths, ratio of lengths
  - Perspective effects are strongest when the object is close to the camera
- Linear Algrebra Review
  - Dot Product

$$x \cdot y = x^T y = \sum_i x_i y_i = ||x|| * ||y|| cos\theta_{xy}$$

  - Output is scalar
  - Measures angles if applied to unit vectors
  - Measures projection of one vector onto another

- Orthogonality: $x^T y = 0$
- Cross Product $a \times b$
- Outputs vector with magnitude: $||a|| * ||b|| sin\theta_{xy}$ and direction perpendicular to both vectors
- Computed as the determinant of $\begin{bmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{bmatrix}$
- $a \times b = -(b \times a)$

Lecture 02 - Projective Geometry

- Points and Lines in Euclidean 2D
  - Point: $x = (x, y)^T \in \mathbb{R}^2$
  - Line: Collection of all points that satisfy $ax + by + c = 0$
    - Can be rewritten as vector: $l = [a, b, c]^T$
    - Equation of a line:

$$l^T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0$$

- Points and Lines in Euclidean 3D
  - Point: $x = (x, y, z)^T \in \mathbb{R}^3$
  - Plane in 3D: Collection of all points that satisfy $ax + by + cz + d = 0$
    - Can be rewritten as vector: $\pi = [a, b, c, d]^T$
    - Equation of a plane:

$$\pi^T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = 0$$

  - Lines in 3D
    - Interpolate between two points $(x_1, y_1, z_1)^T$ and $(x_2, y_2, z_2)^T$
    - Set of all points that satisfy $(x, y, z)^T = (x_1, y_1, z_1)^T + \lambda(x_2 - x_1, y_2 - y_1, z_2 - z_1)^T$
    - Also can be thought of as intersection between 2 3D planes:

$$\begin{bmatrix} \pi_1^T \\ \pi_2^T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

    - Special Case Through Origin: $(x, y, z)^T = \lambda(x_2, y_2, z_2)^T$
        - We are interested in rays *from* the origin ($\lambda >= 0$) in the case of a pinhole camera
- Transformations in 3D:
    - Shift/Translation (3 DOF)
        - $(x, y, z)^T \rightarrow (x, y, z)^T + (x_0, y_0, z_0)^T$
    - Rotation (3 DOF)
        - $(x, y, z)^T \rightarrow R_{3\times3}(x, y, z)^T$
        - Rotation Matrices
            - $R_{3\times3} = (r1, r2, r3)$
            - $R^T R = I$
            - $r_i^T r_i = ||r_i||_2 = 1$ and $r_i^T r_j = 0$ for all i,j
            - 3 DOF despite 9 elements
    - **Euclidean Transformations**: Compositions of rotation and translations (6 DOF)
    - Scaling (1DOF)
        - $(x, y, z)^T \rightarrow \lambda(x, y, z)^T$
    - **Similarity Transformations**: Combinations of rotations, translations, scaling (7 DOF)
    - **Affine Transformations** (12 DOF)
        - Remove orthonormality constraint on R
        - $(x, y, z)^T \rightarrow A_{3\times3}(x, y, z)^T + t$
        - 9 DOF for A (absorbs scaling) and 3DOF for t
- Vanishing Points
    - Points at ∞ in $\mathbb{R}^n$
    - Finite in projective space $\mathbb{P}^n$
    - Solely a function of the direction of the line, not it's position
- Projective Geometry
    - Extension of Euclidean geometry using homogeneous coordinates
    - Two lines always meet in a point (even if the point is at infinity)
    - $\mathbb{R}^2 \rightarrow \mathbb{P}^2 : \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
    - w often denotes homogeneous vector element
    - Key Property: $(x, y, 1) = (wx, wy, w)$
    - What if $w = 0$? $\rightarrow$ This is a point at infinity! Can also be thought of as rays through origin that are parallel to the projective image plane
- Projective Lines

- What does a line in the image correspond to in projective space?
- A line is a plane of rays through the origin
- All rays (x, y, w) satisfying $ax + by + cw = 0$
- $$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0 \Rightarrow l^T x = 0$$
- Can also be represented as:

  $$l = p_1 \times p_2$$

  where p_1 and p_2 are 2 points the line passes through
- Intersection point of 2 lines:

  $$p = l_1 \times l_2$$

Lecture 03 - Projective Geometry pt. 2

- Projective Geometry $\longleftrightarrow$ Euclidean Interpretation
  - In the Euclidean interpretation, we treat w as the third spatial coordinate
  - w is a scaled version of the principal axis Z
  - Image plane is $w = 1$
  - $w = 0$ is the same as $Z = 0$ and is parallel to the image plane, passing through the camera center
- The Line at Infinity
  - $$l_\infty = (0, 0, 1)^T$$
  - Rewritten as $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix} = 0$
  - Line passing through all ideal points (points at infinity)
- Camera Coordinate System with Principal Point Offset
  - Recall: $\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix}$
  - With an offset from the image center:
  - $$u = f\frac{X_c}{Z_c} + u_0$$

- $$v = f\frac{Y_c}{Z_c} + v_0$$
  - This becomes...

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & & u_0 \\ & f & v_0 \\ & & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

- Camera to World: Euclidean Transformation
  - $$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = R_{3\times3} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} + t = \begin{bmatrix} R_{3\times3} & t \\ 0 & 1 \end{bmatrix} X_w$$
  - Putting it all together...

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & & u_0 \\ & f & v_0 \\ & & 1 \end{bmatrix} \begin{bmatrix} R_{3\times3} & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = K\,[R|t]X_w$$

  - In summary
    - Rotation and translation convert from world coordinates to camera-centric coordinates
    - K projects and converts to pixel coordinates
    - $$x \sim K[R|t]X_w$$
    - Camera Projection Matrix: $P_{3\times4} = K_{3\times3}[R_{3\times3}|t_{3\times1}]$
      - K represents camera *intrinsics* and R and t represent *extrinsics*
      - P has 9 DOF (3 for rotation + 3 for translation + 3 for intrinsics)

Lecture 04 - Homographies

- Projective Transformations/Collineations/Homographies
  - Projective transformation is any invertible matrix transformation $\mathbb{P}^2 \rightarrow \mathbb{P}^2$
  - Represents the perspective projection from a world **plane** to the image plane
  - A projective transformation A maps $p' \sim Ap$, or rewritten as $\lambda p' = Ap$
  - Projective transformation matrix must be invertible $\therefore det(A) \neq 0$
  - Scale invariant $\mu\lambda p' = \mu Ap$, therefore only 8DOF

    - Less restrictive transformation type than affine
- Representing a Perspective Projection P as a Homography H
    - $$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \sim K \begin{bmatrix} r_1 & r_2 & r_3 & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$
    - Let's map the ground plane $Z = 0$ to the image plane, this becomes
    $$\begin{bmatrix} u \\ v \\ w \end{bmatrix} \sim K \begin{bmatrix} r_1 & r_2 & T \end{bmatrix} \begin{bmatrix} X \\ Y \\ W \end{bmatrix}$$
- Computing Homographies from 4-Point Correspondences (*4-Point Collineation*)
    - 3 points allow us to recover
    $$\begin{bmatrix} a & b & c \end{bmatrix} = \begin{bmatrix} \alpha a & \beta b & \gamma c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
    where a, b, c are vectors
    - This only has 3DOF and is therefore not enough
    - With a 4th point where A maps d to d'
    $$\lambda d' = \begin{bmatrix} \alpha a & \beta b & \gamma c \end{bmatrix} d$$
    - This will allow us to recover the full homography matrix up to a scale (8DOF)
    - No 3 of the 4 point correspondences can be collinear
    - Often times choose: $(1,0,0), (0,1,0), (0,0,1), (1,1,1)$ as the correspondences (x vanishing point, y vanishing point, origin, and one other point)
- Notes on the Homography
    - The first 2 columns are **vanishing points**
    - This allows us to compute the horizon by taking the cross product of the first 2 columns
    - The transformation from 1 image plane to another can be computed by 2 homographies, H^{-1} to go to world coordinates and then another homography H' to go to the second image plane:
    $$a' = H' H^{-1} a$$

Lecture 05 - Homographies pt. 2

- Recap
  - Homographies are linear maps from $x \in \mathbb{P}^2$ to $y \in \mathbb{P}^2$
  - In computer vision, are a direct outcome of applying a perspective projection to world points lying on a single plane
- Computing Homographies with 4 Correspondences
  - $$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
  - Produces system of 3 linear equations
    - $\lambda x' = h_{11}x + h_{12}y + h_{13}$
    - $\lambda y' = h_{21}x + h22y + h_{23}$
    - $\lambda = h_{31}x + h_{32}y + h_{33}$
  - Input λ in top 2 equations and get, equations that are linear in h_{ij}
    - $a_x = [-x \quad y \quad -1 \quad 0 \quad 0 \quad 0 \quad xx' \quad yx' \quad x']$
    - $a_y = [0 \quad 0 \quad 0 \quad -x \quad -y \quad -1 \quad xy' \quad yy' \quad y']$
    - $h = [h_{11} \quad h_{12} \quad h_{13} \quad h_{21} \quad h_{22} \quad h_{23} \quad h_{31} \quad h_{32} \quad h_{33}]$
    - $\begin{bmatrix} a_x \\ a_y \end{bmatrix} h = 0$
    - A has 8 total DOF, each point nails down 2 so we need 4 point correspondences total
  - Inaccuracies of Measurements
    - There may not be an h that produces exactly $Ah = 0$, therefore we use the SVD of A and set h to be the smallest right singular vector
- Equation of the Horizon
  - Horizon is the line connecting vanishing points (first 2 columns of a homography!)
  - $(v_1 \times v_2)^T \begin{bmatrix} x \\ y \\ z \end{bmatrix} = 0$
  - Points at infinity in the world plane are of the form $(X, Y, W = 0)^T$
  - The "line" connecting them is $l = (0, 0, W = 1)^T \rightarrow$ the line at ∞. The image of this line is the horizon
  - If we can find the projection of this line, can compute the horizon...
- Projective Transformation of Lines
  - If A maps a point to Ap, where does line l map?
  - Equation of a line in original plane $\rightarrow l^T p = 0$
  - Equation of a line in image plane ($p' \sim Ap$) $\rightarrow l^T A^{-1} p' = 0$
  - Implies that: $l' = A^{-T} l$

- Back to the Horizon
  - Projection of the horizon is therefore: $H^{-T}(0,0,1)^T$
  - This becomes: $h_1 \times h_2$, the same equation we had derived previously!
- Horizon gives complete info about how the ground plane is oriented (ONLY assuming standard camera intrinsics K, i.e. no cropping, etc.)

Lecture 06 - Intrinsics from Vanishing Points

- Computing Camera Intrinsics from Vanishing Points
  - Given a scene with 3 orthogonal sets of parallel lines, we can get 3 orthogonal vanishing points
  - Can compute the horizon with 2 of these on the same plane, but cannot compute the 3rd vanishing point from this horizon unless assuming camera intrinsics

Bonus proof: Perpendicular from camera center is the orthocenter of VPs

THEOREM: The image center (u_0, v_0) is the orthocenter of the triangle formed by the projections of three orthogonal vanishing points.

PROOF: See figure 2. Let $C = (u_0, v_0, 1)$ denote the homogeneous coordinates of the image center: it is defined as the intersection of image plane $V_1V_2V_3$ with the optical axis, which is the line through O and perpendicular to $V_1V_2V_3$.

$OC \perp V_1V_2V_3 \Rightarrow OCV_1 \perp V_1V_2V_3$

$\Rightarrow OCV_1 \perp$ any line contained in $V_1V_2V_3$

By B: every plane containing OC is ⊥ to V1V2V3

By C: V2V3 ⊥ CV1, so OCV1 ⊥ V2V3

In particular $\boxed{OCV_1 \perp V_2V_3}$. Moreover, $OV_1 \perp OV_2$, therefore $\boxed{OCV_1 \perp OV_2V_3}$.

When two planes are perpendicular, their intersections with a third plane are also perpendicular. Therefore, the intersection of OCV_1 with $V_1V_2V_3$ is perpendicular to intersection of OV_2V_3 with $V_1V_2V_3$.

In other words $V_1C \perp V_2V_3$. A similar reasoning leads to $V_2C \perp V_3V_1$ and $V_3C \perp V_1V_2$, therefore C is the orthocenter of $V_1V_2V_3$.

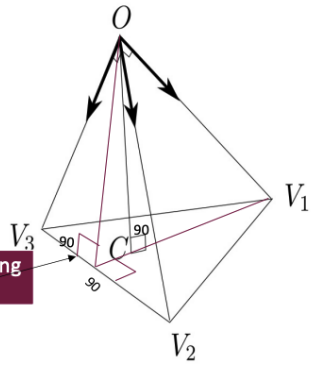
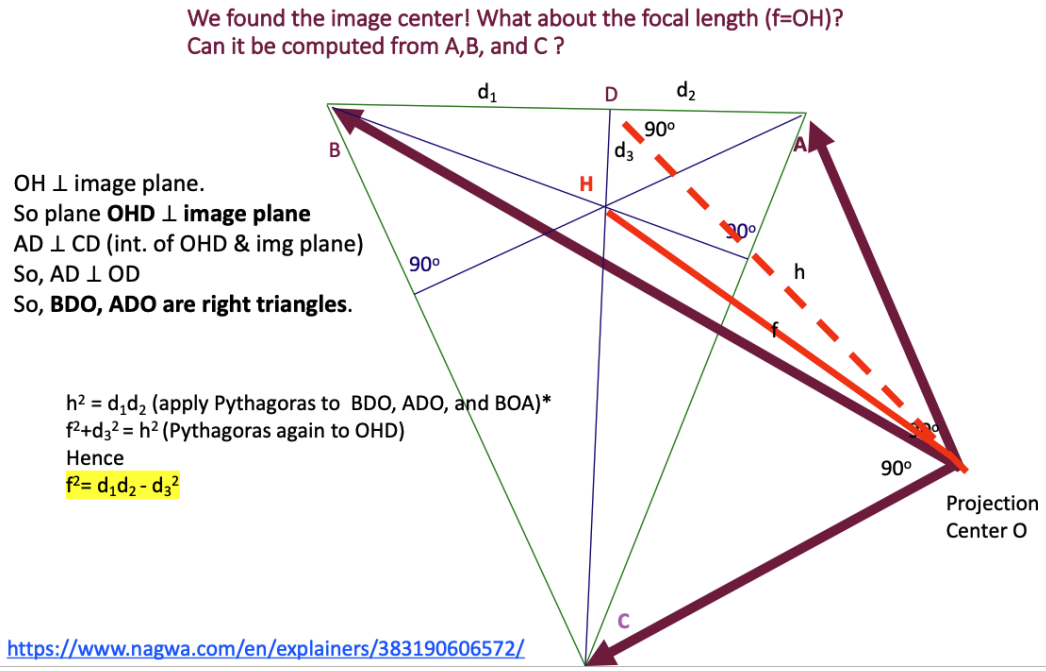


Figure 2: Three orthogonal vanishing points and image center.
We used the fact that the two non-parallel lines OV_2 and V_2V_3 contained in the plane OV_2V_3 are perpendicular to OCV_1, therefore $OV_2V_3 \perp OCV_1$.

- Through some crazy geometry, we get the image center is the orthocenter of the triangle formed by the projections of the 3 orthogonal vanishing points
- To compute the focal length...



- * https://www.nagwa.com/en/explainers/383190606572/

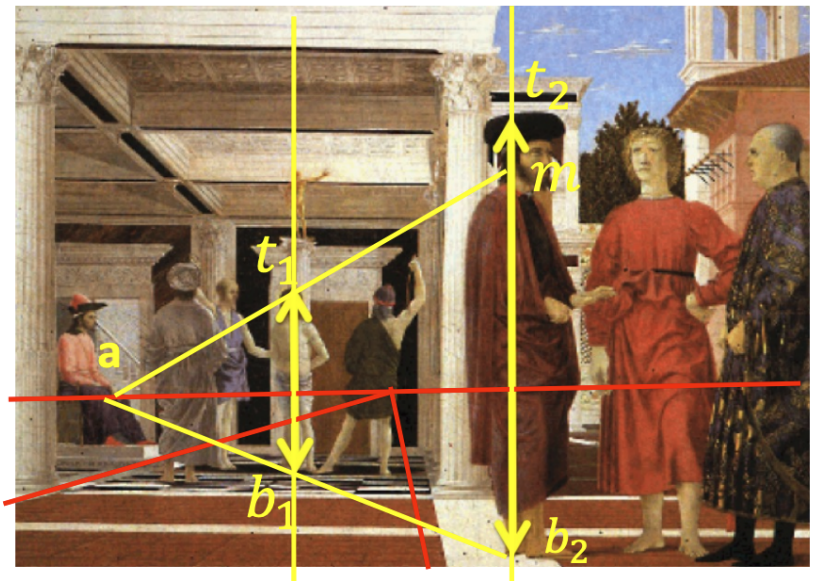
- In Summary
  - (u_0, v_0) is the orthocenter of the 3 orthogonal vanishing points
  - $f = \sqrt{d_1 d_2 - d_3^2}$
  - Can recover full intrinsics K from 3 orthogonal VPs and extrinsics R but not the translation because translations do not affect vanishing points

Lecture 07 - Cross Ratios

- SVD Primer to the Primer
  - Any matrix $A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$
  - Columns u_i of U are an orthonormal basis for the column space (left singular vectors)
  - Columns v_i of V (not V^T!) are an orthonormal basis for the row space (right singular vectors)
  - Σ is a diagional matrix containing square roots $\sigma_i = \sqrt{\lambda_i}$ of eigenvectors of A^TA (the eigenvalues)
  - In relation to the homography
    - $AV = U\Sigma$ or $Av_i = \sigma_i u_i$ so σ_i is the length of Av_i
    - The closest approximation to the null vector h for finding homographies is therefore the smallest right singular vector, corresponding to the smallest singular value
- What is preserved under a Homography?
  - Not lengths, not length ratios

    - Ratio of ratios of distances of 2 points from 2 other collinear points is however
- Cross Ratio
    - $\frac{AC}{AD} : \frac{BC}{BD}$ is the cross ratio of the 4 collinear pts A, B, C, D
    - Can also be written $\frac{AC \cdot BD}{AD \cdot BC}$
    - Remains invariant under projective transformations
    - Different ordering of $CR(A, B, C, D)$ will produce different values but are still preserved under a projective transformation
- Cross Ratio of a Pt at ∞
    - Given pt D is at ∞, the cross ratio becomes $\frac{AC}{\infty} : \frac{BC}{\infty} = \frac{AC}{BC} \rightarrow$ just a ratio!
    - Vanishing pts allow us to leverage this, only need measurement of 1 distance in the world
    - Can also be used the other way: given 2 known lengths in the world plane and the A', B', C' in pixels we can compute the vanishing point
- Computing Distances *from* a World Plane
    1. Find the horizon (from H or by vanishing points)
    2. Connect horizon with bottom of known world length b_1 and unknown world length b_2 , denote this a
    3. Connect a with top of known length t_1 and continue to connect to object of unknown length, denote this m. mb_2 is the same length as b_1t_1
    4. Compute the unknown length via cross ratios and the vanishing point where t_1 and t_2 converge (could be at ∞ or not)

4. Then cross ratio of {v, t2, m, b2} = ratio $\boldsymbol{B_2T_2 : B_2M}$



(and $\boldsymbol{B_2M}$ is statue height.)

Horizon

- Camera 6DOF Pose Estimation with Respect to World Plane
    - Compute extrinsics given homography H and camera intrinsics K
    - Naive Solution:

        - Given all points in the world lie on the ground plane $Z = 0$

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \sim K \begin{pmatrix} r_1 & r_2 & T \end{pmatrix} \begin{pmatrix} X \\ Y \\ W \end{pmatrix}$$

        - We then should be able to recover: $K^{-1}H = (r_1, r_2, T)$ and $r_3 = r_1 \times r_2$ BUT we are not guaranteed to recover a valid rotation matrix in this manner; there is no constraint guaranteeing r_1 and r_2 will be orthonormal
    - Procrustes
        - Goal: Given camera intrinsics and homography, let us find the closest valid rotation matrix
        - Given $K^{-1}H = \begin{pmatrix} h_1' & h_2' & h_3' \end{pmatrix}$
        - We find the orthogonal matrix R that is the closest to $\begin{pmatrix} h_1' & h_2' & h_1' \times h_2' \end{pmatrix}$
        - $$argmin_{R \in SO(3)} ||R - \begin{pmatrix} h_1' & h_2' & h_1' \times h2' \end{pmatrix}||_F^2$$
    - Kabsch Algorithm for Procrustes
        - If the SVD of

$$\begin{pmatrix} h_1' & h_2' & h_1' \times h_2' \end{pmatrix} = U \Sigma V^T$$

        - Then

$$R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(UV^T) \end{pmatrix} V^T$$

        - $$T = h_3' / ||h_1'||$$
        - Diagonal matrix is inserted to guarantee $det(R) = 1$

Lecture 08 - P3P

- Aside: Orthographic Projections
    - Used in engineering drawings
    - Are images formed by rays orthogonal to the image plane, therefore lengths are preserved
- World to Camera Frame Euclidean Transformation
    - What do R and t mean in $\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R | t \end{bmatrix} X_w$

    - R is the rotation of the world axes with respect to the camera axis: R_w^c
    - T is performed *after* the perspective projection, so if a is the direct translation from the camera origin to the world origin, then $T = Ra$
- Pose from 3 Point Correspondences (P3P)
    - Given 3 known world 3D coordinates $P_i^w \in \mathbb{R}^3$ and it's camera-centric 3D coordinates $P_i^c \in \mathbb{R}^3$...
    - The P3P problem is finding camera pose R, T such that

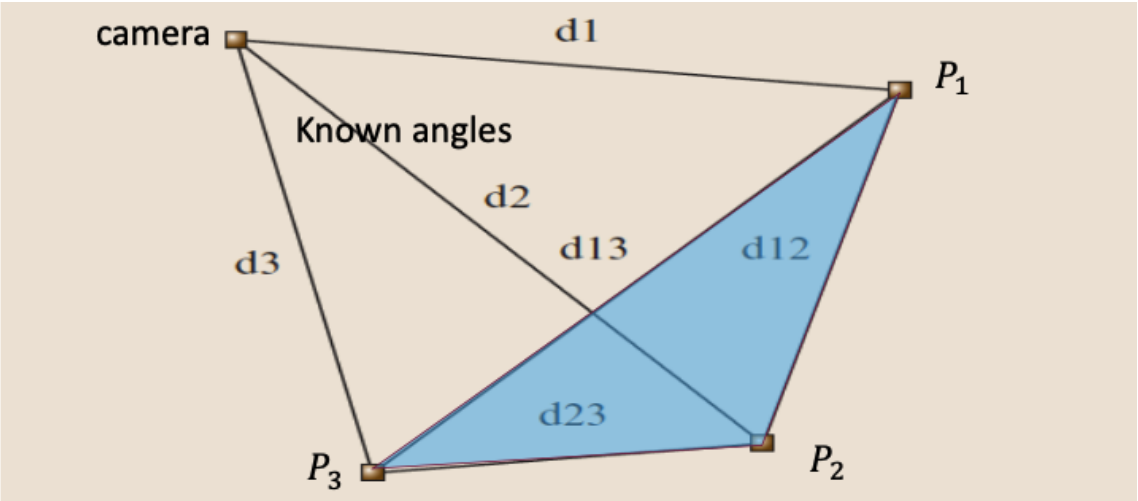
$$P_i^c = RP_i^w + T$$

    1. Convert pixels to calibrated coordinates: $p_i \sim K^{-1}(u_i \quad v_i \quad 1)^T$
        - Essentially image plane coordinates with principal point as origin and focal length of 1
        - Euclidean interpretation: p_i is the vector in camera coordinates pointing from the camera center in the direction of the 3D point we are imaging
        - Now just need to find depths to get camera-centric 3D coordinates
    2. Find λ_i, R, T such that

$$\lambda_i p_i = RP_i + T$$

        - λ_i's are not guaranteed depths unless p_i's are unit vectors
        - Therefore the depths are: $\frac{d_i}{||p_i||_2} p_i = RP_i + T$
    3. Use law of cosines to get a quadratic equation for each point

$$d_i^2 + d_j^2 - 2d_i d_j cos\delta_{ij} = d_{ij}^2$$

        - Law of Cosines: $c^2 = a^2 + b^2 - 2ab \cdot cos(C)$ for any triangle
        - 

    4. Now insane algebra (not all pictured here)

P3P Step 1: The algebraic drudgery

Reduces to two quadratic equations in u and v.

$$d_{13}^2(u^2+v^2-2uv\cos\delta_{23}) = d_{23}^2(1+v^2-2v\cos\delta_{13}) \quad (1)$$
$$d_{12}^2(1+v^2-2v\cos\delta_{13}) = d_{13}^2(u^2+1-2u\cos\delta_{12}) \quad (2)$$

a) Solve Eqn (1) for u^2 in terms of u, v, v^2 (and constants).
b) Plug this solution into Eqn (2) RHS, so that it has no u^2 term. Solve for u in terms of v, v^2, and constants.
c) Plug this solution for u back into Eqn (1), so that it has no more u or u^2. Instead, it is a 4th degree equation in v. *Get the 4 real solutions analytically. (NOTE)*
d) Then plug back into the solution found in step b) above, to get u.
e) Then get d_1 from the quadratic equations on the last page.
f) Then plug back into $d_2 = ud_1$ and $d_3 = vd_1$ from the last page to get d_2, d_3.

- Grunert 1841
- This will give us multiple valid solutions (Roots of a 4th degree polynomial)
- To find best:
  - Only check solutions with positive, real roots
  - Use Kabsch algorithm to solve Procrustes problem for each valid solution and compare computed R, T against 4th known world point (if known)

Lecture 09 - P3P pt. 2 and PnP

- Continuing P3P Problem: How do we solve for R, T from frame B to A, given n point correspondences with known depths
  - $\frac{d_i}{||p_i||_2} = RP_i + T \rightarrow A_i = RB_i + T$
  - Becomes a minimization problem: $min_{R,T} \sum_i^N ||A_i - RB_i - T||^2$
  - Differentiate with respect to T and we find: $T = \frac{1}{N}\sum_i^N A_i - R\frac{1}{n}\sum_i^N B_i = \bar{A} - R\bar{B}$
    - $\bar{A}$ and $\bar{B}$ are the centroids of matrix A and B respectively
    - This gives us 2 important observations
      - If we find the correct rotation matrix R, we can solve for T afterwards
      - If the centroids are both 0, we can set $T = 0$
  - Rotation matrix is unaffected by translating the point sets, so we can subtract the centroids from the point sets before finding the rotation
  - We now have $min_R||\hat{A} - R\hat{B}||_F^2$
    - $\hat{A} = \begin{pmatrix} A_1 - \bar{A} & \dots & A_N - \bar{A} \end{pmatrix}$
    - $\hat{B} = \begin{pmatrix} B_1 - \bar{B} & \dots & B_N - \bar{B} \end{pmatrix}$
  - We can now reuse our solution to the Procrustes problem by slightly rewriting the minimization problem:

$$argmin_{R \in SO(3)} ||\hat{A} - R\hat{B}||_F^2 = argmin_{R \in SO(3)} ||R - \hat{A}\hat{B}^T||^2$$

- Summary of P3P Step 2 (Already solved for depths)
  1. Compute centroids $\bar{A}$ and $\bar{B}$ of the 2 sets of 3D points
  2. Create matrices $\hat{A}_{3\times n}$ and $\hat{B}_{3\times n}$ after subtracting $\bar{A}$ and $\bar{B}$ from all points in the 2 sets (Note that each point is a column in A and B as opposed to the usual row)
  3. To find R solve $argmin_{R \in SO(3)} ||\hat{A} - R\hat{B}||_F$
     1. Set $\hat{R} = (\hat{A}\hat{B}^T)_{3x3}$
     2. Decompose via SVD: $\hat{R} = U\Sigma V^T$
     3. Set $R = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & det(VU^T) \end{pmatrix} V^T$
     4. Set $T = \bar{A} - R\bar{B}$
  4. NOTE: There will most likely be multiple solutions due to the roots of the 4th degree polynomial when solving for depths. If a 4th point correspondence is known, the solved R and T's should be compared with it to select the most accurate solution
- PnP Problem
  - Gives unique solution by having at least 6 point correspondences and don't have to do ridiculous algebra like in P3P
  - Steps
    1. Convert to calibrated coordinates: $\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \sim K^{-1} \begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix}$
    2. Need to solve: $\lambda_i \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = R \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} + T$
       - Rewrite equation by dividing by lambda for x_i and y_i equations to make them linear in the elements of R and T
       - Produces $A_{2n\times 12} x_{12\times 1} = 0$ so need at least 11 equations or 6 point correspondences
    3. Kabsch Algorithm for Procrustes Again
       1. Solve $A_{2n\times 12} x_{12\times 1} = 0$ from $n \geq 6$ correspondences
       2. Assemble rotation matrix $\hat{R}$ from the solution for x
       3. $\hat{R} = U\Sigma V^T$, set determinant to 1 by doing same diagonal matrix computation
       4. Adjust translation to be consistent with new R by resolving $A_{2n\times 12} x_{12\times 1} = 0$ for only t_1, t_2, t_3 holding R fixed
- Calibrating Cameras Under more General Intrinsics
  - If pixels are not square, can have different focal lengths for x and y

$$\begin{bmatrix} f & & u_0 \\ & f & v_0 \\ & & 1 \end{bmatrix} \rightarrow \begin{bmatrix} s_x & & u_0 \\ & s_y & v_0 \\ & & 1 \end{bmatrix}$$

- If radial distortions are present, u_0, v_0 are no longer constants and are now functions of distance r from camera center

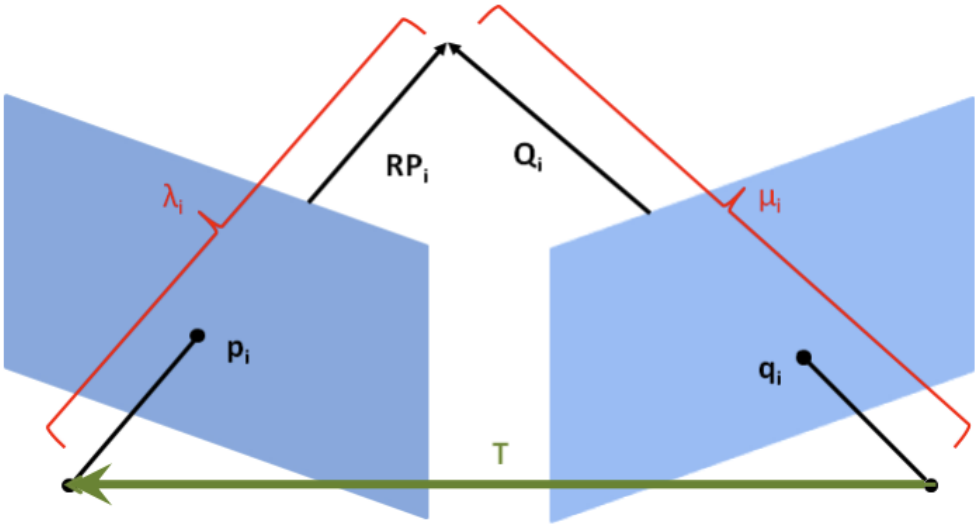
$$\begin{bmatrix} f & & u_0 \\ & f & v_0 \\ & & 1 \end{bmatrix} \rightarrow \begin{bmatrix} f & & u(r) \\ & f & v(r) \\ & & 1 \end{bmatrix}$$

  - Often parameterized as
    - $u(r) = u(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots)$
    - $v(r) = v(1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + \dots)$
    - Greater degree results in greater accuracy but requires more images to fit and is more computationally expensive

Lecture 10 - SfM

- Calibrating Camera Coordinates in Practice
  - Use OpenCV and multiple views of a checkerboard
  End of Single View Geometry

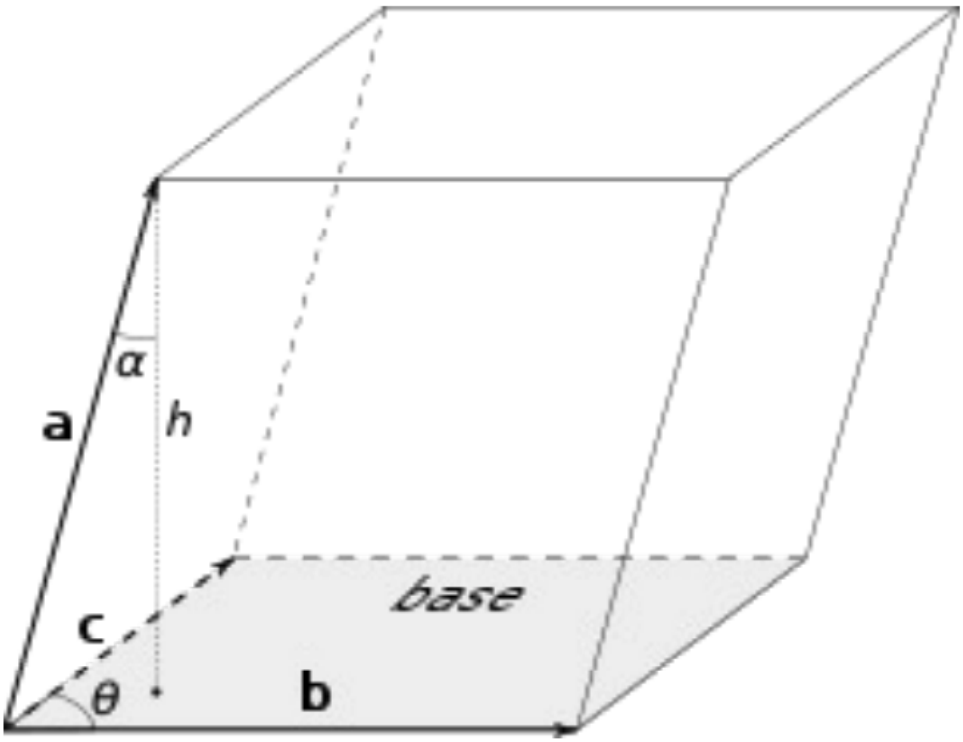
- 3D Motion from 2 Views: Structure from Motion Problem (SfM)
  - Problem Overview
    - Input: 2 calibrated views of the same rigid 3D scene
    - Output: Find R, T between the two views and depths for point correspondences from each view
    - Mathematical Formation
      1. $X^{C1} \rightarrow [R_{C1}^{C2}[X]^{C1} + T_{C1}^{C2}]^{C2}$
      2. $[\lambda p]^{C1} \rightarrow [R(\lambda p) + T]^{C2} = [\mu q]^{C2}$



$$R(\lambda p) + T = \mu q$$

- Epipolar Constraints between 2 Views of a Scene
  - $$q^T(T \times Rp) = 0$$
  - Allow us to eliminate depths from the equation
  - How?
    - In our previous equation: $\lambda Rp + T = \mu q$; q, Rp, T are the edges of a triangle, which is by definition planar

- Volume of a parallelpiped:



$$\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = \mathbf{b} \cdot (\mathbf{c} \times \mathbf{a}) = \mathbf{c} \cdot (\mathbf{a} \times \mathbf{b})$$

- The vectors q, T, and Rp form a triangle which has volume 0 (because it's planar)
- Don't overthink it, very weird problem reformation but it's actually pretty simple

Lecture 11 - SfM pt. 2

- Recap: Trying to solve

  $$R(\lambda_i p_i) + T = \mu_i q_i$$

  and rewrote this as

  $$q_i^T (T \times R p_i) = 0$$

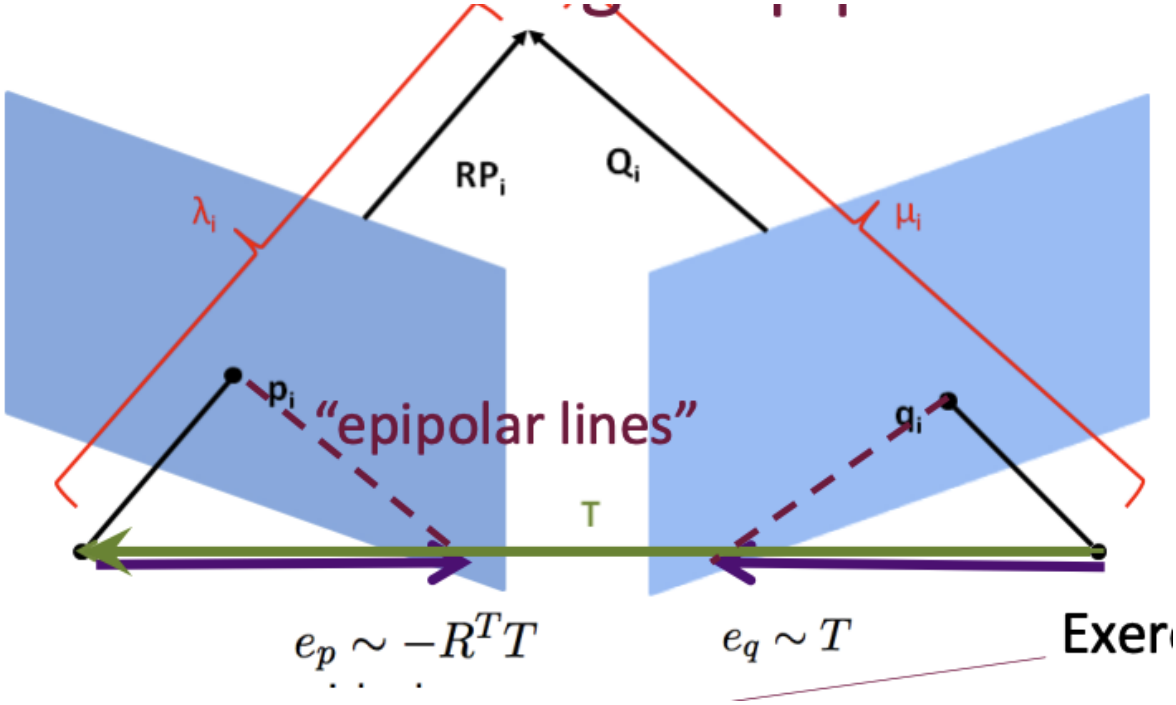
  - R, λ, T, μ are all unknowns
- Epipolar Planes, Lines, and Epipoles
  - An epipolar plane is any plane that contains the baseline. Any 3D point will produce a corresponding epipolar plane
  - The intersection of an epipolar plane with the image plane is the epipolar line for that 3D point

- Epipoles are the images of the other camera center on each plane
  - Also can be thought of as the intersection of the baseline T with the two planes or the vanishing point of the translation direction in each plane
  - All epipolar lines in each image plane pass through its epipole
  - Epipole for from view q

$$e_q \sim T$$

  - Epipole from view p

$$e_p \sim -R^T T$$



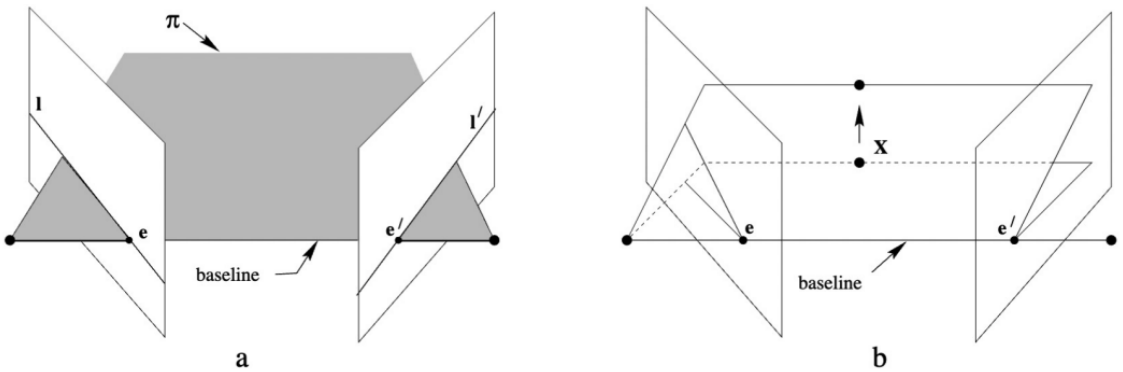
- Linear Algebra Review: Cross Products through Skew-Symmetric Matrices
  - $$a \times b = [a]_x b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

  - Skew-symmetric matrix form of a vector a is often denoted $\hat{a}$
- The Essential Matrix E
  - $$q_i^T (T \times R p_i) = 0 \rightarrow q_i^T (\hat{T} R) p_i$$

  - $$E = (\hat{T} R) \therefore q_i^T E p_i = 0$$

  - This is now linear in the unknowns of $E_{3\times3}$, we will need to recover $T_{3\times1}$ and $R_{3\times3}$ later
  - Verifying Linearity

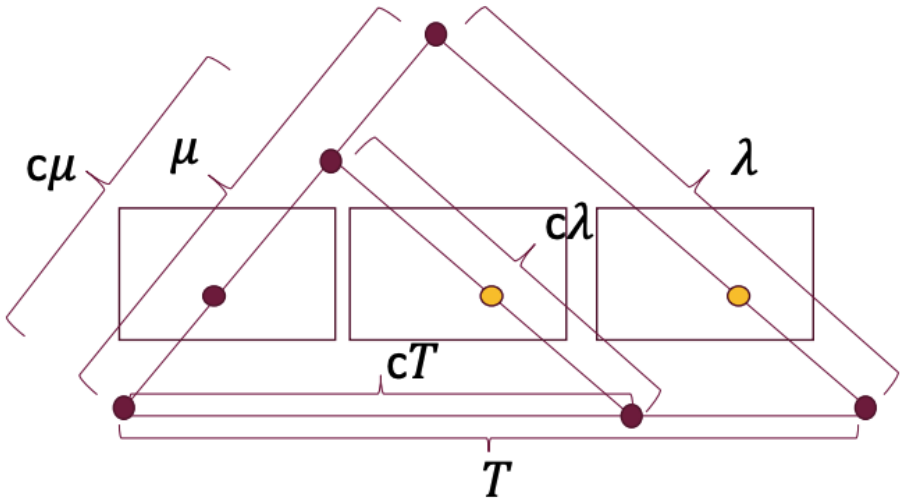
    - $E = (e_1 \quad e_2 \quad e_3)$
    - $q^T \begin{pmatrix} e_1 & e_2 & e_3 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = q^T (p_x e_1 + p_y e_2 + p_z e_3) = \begin{pmatrix} p_x q^T & p_y q^T & p_z q^T \end{pmatrix} \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix} 0$
    - Equation is therefore linear in homogeneous in $E' = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix}$
- Slight Aside: Fundamental Matrix F
    - Essential matrix E connects image plane coordinates (calibrated coordinates)
    - Fundamental matrix F assumes still pixel coordinates
    - Can rewrite $q^T E p = 0$ as $(K_q q)^T F (K_p p) = 0$
- Epipolar Lines in terms of E
    - Recall equation of a line is projective space is $l^T p = 0$
    - Epipolar line is $q^T E p = 0$ with coefficients (l) $E^T q$
- Epipolar Lines and Point Correspondences
    - Given a point p is the left view's image of a world point, the right image of that point is constrained to line on the corresponding right epipolar line
    - This makes finding point correspondences much easier as we can reduce it to a 1D search
- Special Case: Frontoparallel/Parallel Stereo Cameras
    - If two cameras have no rotation differences and lie on the same axis, finding point correspondences via the epipolar line constraint allows us to search on the same row of the image
    - In this case the epipole (image of other camera center in each image) is at infinity and the epipolar line is horizontal
- Minor Aside: Epipolar "Pencils"



    - The epipolar lines in each image plane form a "pencil" whose tip is the epipole
    - Allow us to visually identify camera orientations
- Computing E from Point Correspondences
    - We know $p_i' E q_i = 0$ for all corresponding points p_i and q_i

- Longuet-Higgins' 8-Point Algorithm
  - Each correspondence gives us one linear equation in the unknowns E
  - Let $a = \begin{pmatrix} p_x q^T & p_y q^T & p_z q^T \end{pmatrix}_{1\times 9}$
  - $$\begin{pmatrix} a_1^T \\ a_2^T \\ \cdots \\ a_n^T \end{pmatrix} E' = 0$$
  - Solution to E' is the null space of $A = \begin{pmatrix} a_1^T \\ a_2^T \\ \cdots \\ a_n^T \end{pmatrix}$, therefore set E' to the last right singular vector of $A_{n\times 9}$
  - Rank of A
    - If we have greater than 8 points, it is not a problem. The last singular vector of A already gives us the closest thing to finding a null vector for a non-singular matrix
    - If A is too low rank, we can't do anything besides restart and find better point correspondences. This can occur if points lie on any quadric surfaces (planes, cylinders, ellipsoids, etc)
  - After solving for E...
    - $E = \hat{T}R$ has fewer than 8DOF (T has 3, R has 3, and E is scale invariant so there are only 5 in total
    - We technically don't need 8 points, there are methods with as few as 5 points but are even more complicated than the P3P solution and is generally just less convenient than getting a few more point correspondences
    - Scale invariance: There is actually no way of recovering the scale in this problem formulation without some measure of distance in the real world. Most often this is overcome through some previous knowledge of the world or using gyroscopic measurements of a robot moving, where the multiple views are from the same cameras at different position in space and time

So, suppose one solution to SfM is R, T, λ, μ.
Then, an equally valid solution is $R, cT, c\lambda, c\mu$!



Lecture 12 - SfM pt. 3

- Recap: Strategy for 2-View SfM
  - Solve for $E = \hat{T}R$
  - Then from E, get R, T
  - Finally recover λ_i, μ_i
  - Problem: How to ensure E is valid
- Valid Essential Matrices
  - E is essential if and only if $\sigma_1(E) = \sigma_2(E) \neq 0$ and $\sigma_3(E) = 0$
  - Proof:
    - E is a singular matrix because $E = \hat{T}R$ and $\det(\hat{T}) = 0$
    - Can find the other 2 singular values of E from the eigenvalues of EE^T
    - Skipping over the math, we get $\sigma_1(E) = \sigma_2(E) = ||T||$
- To get the closest valid essential matrix,
  1. $$E = SVD(E) = U \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \sigma_3 \end{bmatrix} V^T$$
  2. $$E_{new} = U \begin{bmatrix} (\sigma_1 + \sigma_2)/2 & & \\ & (\sigma_1 + \sigma_2)/2 & \\ & & 0 \end{bmatrix} V^T$$

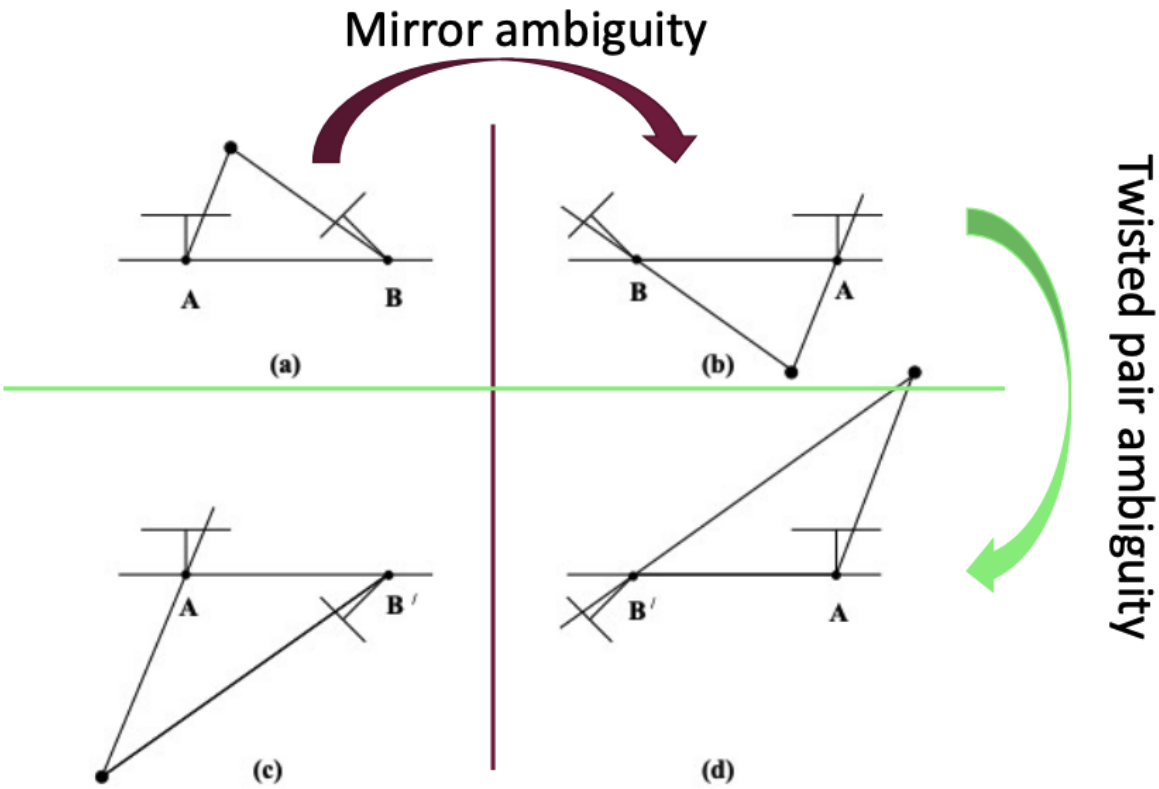
3. Alternatively can also just set $\Sigma = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix}$ because E is scale-invairant

- Construction of E as $\hat{T}R$
  - Proof:
    1. - Can consider the simplest possible valid essential matrix: $E = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
       - This can be decomposed into $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \hat{T}R$ which gives us $T = T_{-z} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$ and $R = R_{z,\pi/2}$, a 90° rotation about the z axis
    2. - If Q is orthogonal, $Q^TQ = I$, then $\hat{Qa} = Q\hat{a}Q^T$
    - Putting 1 and 2 together
      - $E = U \begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T = \sigma U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$
      - $$E = \sigma U \hat{T_{-z}} R_{z,\pi/2} V^T = \sigma U \hat{T_{-z}} U^T U R_{z,\pi/2} V^T = \sigma (U\hat{T}_{-z})(U R_{z,\pi/2} V^T)$$
        - We can see T is the last left singular vector of E with a flipped sign (kinda, will revisit in a sec) and R is orthogonal because U and V are orthogonal as well
    - This solution is not unique! There are 4 possible solutions (also can remove scale σ technically, so we shall do so below). $E = \ldots$
      1. $(U\hat{T}_{-z})(UR_{z,+\pi/2}V^T)$
      2. $(U\hat{T}_{+z})(UR_{z,+\pi/2}V^T)$
      3. $(U\hat{T}_{+z})(UR_{z,-\pi/2}V^T)$
      4. $(U\hat{T}_{-z})(UR_{z,-\pi/2}V^T)$
    - Can disambiguate these solutions by enforcing all points to have positive depths (as they are in front of the camera)
    - Additionally, 2 of the rotation matrices will be in a left-hand coordinate system (having $\det(R) = -1$) so we can remove solutions that way as well
  - Computing depths λ_i, μ_i through triangulation
    - Can compute depths up to a scale factor after computing R and T
    - Set $||T|| = 1$ since scale is irrelevant
    - $$(q_i \quad -Rp_i)_{3\times 2} \begin{pmatrix} \mu_i \\ \lambda_i \end{pmatrix}_{2\times 1} = T_{3\times 1}$$

- Have 3 equations and 2 unknown, so can solve each point independently using the pseudoinverse

Lecture 13 - SfM pt. 4 and Mosaicing

- Types of ambiguity in 2-View SfM Solutions
  - Mirror Ambiguity: If T is a solution, then $-T$ is also a solution. There is no way to disambiguate the epipolar constraint from $q^T(-T \times Rp) = 0$
  - Twisted Pair Ambiguity: If R is a solution, then also $R_{T,\pi}R$ is a solution. The first camera is twisted around the baseline 180°



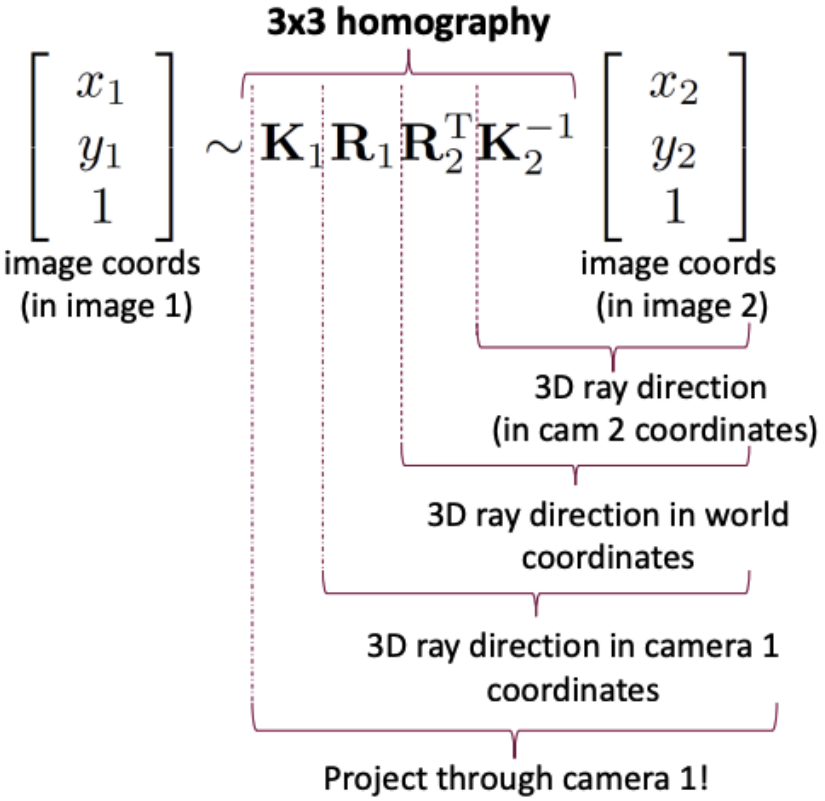
- In 2-View SfM, what if there is no translation?
  - Epipolar constraint becomes $0 = 0$ so can't continue with SfM solution
  - Can however go back to $\lambda q = \mu Rp + T(= 0) = \mu Rp \therefore q \sim Rp$
    - Going back to pixel coordinates from calibrated:

$$K^{-1}q_{px} \sim RK^{-1}p_{\text{px}} \therefore q_{\text{px}} \sim KRK^{-1}p_{\text{px}}$$

    - This is a homography! We are mapping one image plane to another of the same rigid world-view. Mapping of one plane to another is exactly a homography

$$H = KRK^{-1}$$

        - Intuitively in the equation, we are converting to calibrated coordinates, rotating to get to the second view's frame, and then projecting via the intrinsics
- Checking for no translation during SfM
    - Set up n point correspondences as $A_{2n\times 9}h_{9\times 1} = 0$ (same as when solving for homography)
    - If $\text{rank}(A) \approx 8$, can approximately compute H, meaning there is no significant translation
- Image Stitching/Mosaicing from Rotated Views
    - Can use the above to stitch images together (like in a panorama)



    - Can't use to create a greater than 180° panorma, as you start projecting backwards-facing views onto the same image plane
    - To do this, we need to instead project onto a sphere
- Hough and RANSAC
    - Not covered on Midterm 1, return to later

Lecture 15 - Hough Transform and RANSAC

Fitting a line to data

- Given data (x_i, y_i) belonging to a line $x\ cos(\theta) + y\ sin(\theta) = d$

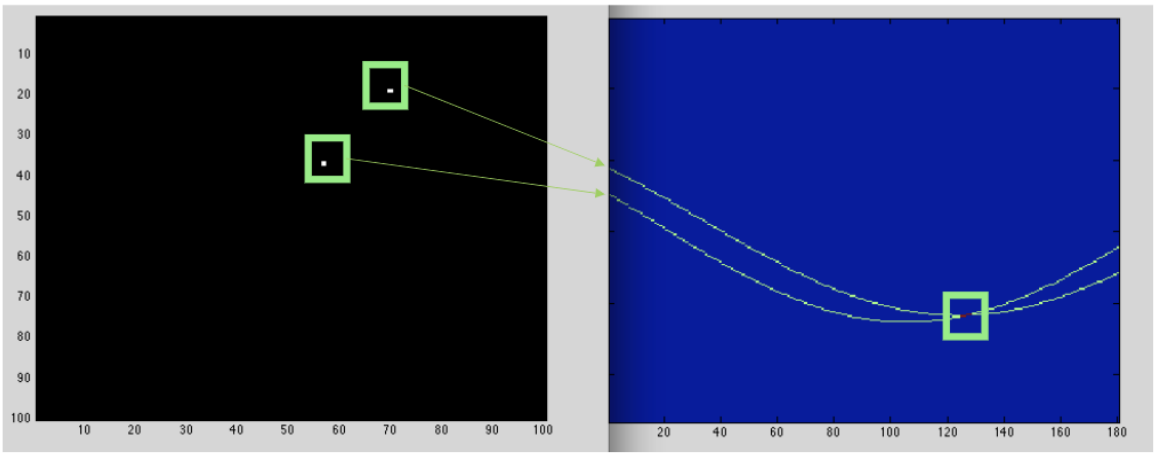
- We can solve for the line by optimization (minimizing the mean squared error)

$$\text{argmin}_{\theta\in[0,2\pi],\, d\geq 0}\sum_{i=1}^{N}(x_i cos(\theta)+y_i sin(\theta)-d)^2$$

- Solve by setting $\frac{\partial f}{\partial d}=0$ and get $d=\frac{1}{N}\sum_i(x_i cos(\theta)+y_i sin(\theta))$
- Eliminate d and get the solution is the eigenvector to the minimal eigenvalue of C (not tested)
- This corresponds to maximizing (maximum likelihood)

$$e^{\frac{-1}{2\sigma_i^2}}(cos(\theta)x_i+sin(\theta)y_i-d)^2$$

Hough Transform

- Idea: What if there are multiple lines? How do we choose which points belong
- For a line in the form $x\ cos(\theta)+y\ sin(\theta)=d$, the parameter space of the line is in (θ,d)
  - Each point in the parameter space corresponds to a line in 2D
  - For a set of points (x_i,y_i), if we plot θ vs d, the intersection points are where multiple points exist on the same line
  - 

- Hough Voting Procedure: For every possible line (d,θ), count the number of points that support it

$$\sum_{\text{points } x,y}1(x\ cos(\theta)+y\ sin(\theta)=d)$$

  - The (d,θ) with the maximum votes wins
- Issue 1: Points shouldn't need to lie perfectly on the line to vote
  - Solution:

$$\sum_{\text{points } x,y}1(|x\ cos(\theta)+y\ sin(\theta)-d|<\delta_{\text{threshold}})$$

- Issue 2: *For every possible line* (d, θ)
  - Solution: Discretize the parameter space
    - For each of some enumerated discrete parameter combinations $\{(d_j, \theta_j)\}_{j=1}^{J}$, count supporting points among the data
- Hough Circle Detection (3 parameters)
  - Fixing (x, y) in the circle equation produces

$$(x-a)^2 + (y-b)^2 = r$$

  - We get the equation of a cone in (a, b, r) Hough space
- Hough Ellipse Detection (5 parameters)
  - Hough transform needs the computation of $V(c_x, c_y, \alpha, \beta, \theta)$
  - Becomes intractable beyond lines and circles
- Application: Automatically grouping vanishing points
  - Each pair of detected line segments votes for their point of intersection
  - Use larger weights for longer segments and non-collinear segments
- Disadvantages of Hough Transforms
  - Needs a bounded parameter space to allow a finite discrete set of hypotheses
  - Poor scaling: Alternative solution that scales much better is RANSAC

RANSAC

- Random Sample Consensus
- Restrict to finding one among a group (one line, circle, etc)
- Algorithm
  - Set maximum inlier count $M = 0$
  - For k iterations
    - Find the corresponding line l, that goes through both (for general case, choose number of points in minimal sample set)
    - Check how many other points approximately lie on this line (inliers)
    - If $n_{\text{inliers}} > M$, set $M = n_{\text{inliers}}$ and set best candidate to l^*
- Minimal Sample Sets
  - Plane: 3 points
  - Circle: 3 points
  - Ellipsoid: 5 points
  - Line: 2 points
  - Homography: 4 2D $\rightarrow$ 3D point correspondes on a 3D plane

    - P3P: 3 2D $\rightarrow$ 3D point correspondences in 3D
- Probabilistic Analysis of RANSAC
    - Suppose the probability of any given sample being a true inlier is ϵ
    - Suppose the minimal set has M points (listed above)
    - The probability that the minimal set is all inliers is ϵ^M
    - In k iterations, the probability of never finding an all-inlier set (RANSAC failing) is $(1-\epsilon^M)^k$
    - The probability of success is then:

$$1-(1-\epsilon^M)^k$$

    - The number of iterations is then defined as

$$k=\frac{\log(1-p)}{\log(1-\epsilon^M)}$$

        - Where p is the desired probability of success, ϵ is the probability of a sample being an inlier, and M is the number of samples in a minimal set
- RANSAC vs Hough
    - RANSAC can only deal with 1 model, Hough can detect multiple
    - RANSAC is more efficient when fraction of outliers is low
    - RANSAC requires the solution of a minimal set problem
    - Hough needs a bounded parameter space
    - Hough is intractable for large numbers of unknowns

Lecture 16 - Optical Flow

- What Methods for Each Type of Correspondence?

Correspondence	Method
3D -> 3D	Procrustes to solve R, T
3D -> 2D	PnP/P3P
2D -> 2D	SfM/Image Stitching

Optical Flow

- The pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene
- Problem:

    - Given two images $I_t(x, y)$ and $I_{t+1}(x, y)$, can we compute the "optical flow" to tell us which pixel in the second image matches with the first
    - Solution: If video frames are close enough, we can look in the neighborhood of $I_t(x, y) \approx I_{t+1}(x + \delta x, y + \delta y)$
- Local Search Objective Function
    - Parameters:
        - (x_0, y_0) Pixel we are computing the optical flow from
        - N Neighborhood patch around a pixel
        - $(\delta x, \delta y)$ Offset we are optimizing for - the optical flow
    - We want to optimize (in discrete form)

$$\text{argmin}_{\delta x, \delta y} \sum_{(x,y) \in N(x_0, y_0)} (I_t(x, y) - I_{t+1}(x + \delta x, y + \delta y))^2$$

    - To make this tractable, we can make the assumption that the change is small and use a first order taylor expansion:

$$I_t(x, y) - I_{t+1}(x + \delta x, y + \delta y) \approx \Delta I_t(x, y) - \nabla I_{t+1}(x, y)^T \begin{pmatrix} \delta x \\ \delta y \end{pmatrix}$$

    - Difference $\Delta I_t(x, y)$: $I_t(x_i, y_i) - I_{t+1}(x_i, y_i)$ for each respective pixel
    - Spatial Gradient $\nabla I_{t+1}(x, y)^T$: Compute through convolution in x and y direction
- Lukas-Kanade Optical Flow
    - Solution to above overdetermined linear systems of equations
        - Why overdetermined: Cannot use patch size of 1, would result in 1 equation with 2 unknowns
    -

$$\begin{bmatrix} \nabla_x I|_{p_0} & \nabla_y I|_{p_0} \\ \cdots & \cdots \\ \nabla_x I|_{p_i} & \nabla_y I|_{p_i} \\ \cdots & \cdots \\ \nabla_x I|_{p_n} & \nabla_y I|_{p_n} \end{bmatrix} \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = \begin{bmatrix} \Delta I|_{p_0} \\ \cdots \\ \Delta I|_{p_i} \\ \cdots \\ \Delta I|_{p_n} \end{bmatrix}$$

    - Solve with pseudo-inverse $Ax = b \rightarrow x = (A^T A)^{-1} A^T b$
- Aside: Horn-Schunck Optical Flow
    - Instead of assuming shared flow over a local pixel neighborhood, Horn-Schunck assumes smoothness over the global flow field rather than local consistency
    - No closed-form solution, use iterative gradient descent updates
- Assumptions We've Made
    - Brightness Constancy: Color doesn't change between different viewpoints
    - No Occlusions: Things stay in sight

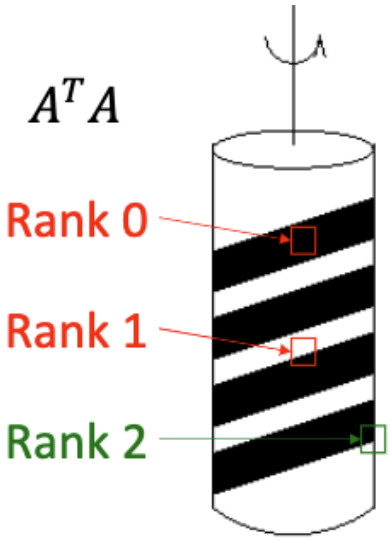
- Small Motions: Minimal patch displacement (because of Taylor expansion)
- Locally Uniform Motion: Constancy of pixel motion within a small neighborhood (ex: rotation/scaling breaks this assumption)
- **Invertibility of** $(A^TA)^{-1}$

Lecture 17 - Optical Flow pt. 2

Invertibility of A^TA

- Optical flow does not always reflect the physical motion field
  - I.e. barber pole! We'll see this example a lot coming up
- $\text{rank}(A^TA) = 0$
  - White wall problem
  - $\nabla I_{t+1}(x, y) = 0$
  - No gradient/approximately zero gradient, pixels are indistinguishable from one another
  - Results in both eigenvalues of A^TA being small
  - I.e. uniform areas of barber pole (within red, blue, or white stripe)
- $\text{rank}(A^TA) = 1$
  - Constant/Aligned gradients
  - $\nabla I_{t+1}(x, y) = \vec{c}, \forall x, y \in N$
  - In barber pole these are the edges of the stripes. Gradient is indistinguishable along the direction of the stripe
- $\text{rank}(A^TA) = 2$
  - Best case scenario: Often times are corners in practice/highly patterned areas
  - In barber pole, these are the corner of the stripe
  - What part of the barber pole we focus on affects the optical flow we see!
  - In this case the optical flow we perceive is downward because the only rank 2 patches

(corners) are on the vertical axis



Barber's pole

- Optical Flow Confidence
  - Check determinant of $A^T A$ for invertibility
  - Can rank various pixels according to determinant or to smallest singular value of A (square root of smallest eigenvalue)
  - If b in $Ax = b$ is not in the range of A, the MSE will be high
    - Occurs when other assumptions are violated like occlusion, light refraction, etc.

Inferring 3D Motion from Optical Flow

- Problem Statement: Given optical flow, can we compute the 3D motion of the camera?
  - Haven't we already done this with SfM? Yes but with some additional reasonable assumptions we can compute it much more efficiently with this method
- Pixel Motion in terms of 3D Object Motion (1)
  - For $p = (x, y, [1]) \in \mathbb{P}^2$
    - $x = \frac{X}{Z}$
    - $y = \frac{Y}{Z}$
    - $p = \frac{1}{Z}P$
    - $$\dot{p} = \frac{dp}{dt} = \frac{\dot{P}}{Z} - \frac{\dot{Z}}{Z}p$$
    - Where...
      - $\frac{\dot{P}}{Z}$ is the projection of the 3D velocity

        - $\frac{\dot{Z}}{Z}$ is the inverse time to collision
- 3D Motion in terms of Camera Motion (2)
    - For a camera moving at velocity V and spinning at angular velocity Ω

$$\dot{P} = -\Omega \times P - V$$

- Combining 2 Key Equations
    - Plug $\dot{P}$ from equation (2) into equation (1)
    - $$\dot{p} = \frac{1}{Z}\begin{bmatrix} xV_z - V_x \\ yV_z - V_y \end{bmatrix} + \begin{bmatrix} xy & -(1+x^2) & y \\ (1+y^2) & -xy & -x \end{bmatrix}\Omega$$
        - First part of the equation is translational flow ($\dot{p}_{\text{trans}}$)
        - Second part is rotational flow ($\dot{p}_{\text{rot}}$) which is independent of depth

Lecture 18 - 3D Motion from Optical Flow

3D Motion from Optical Flow

$$\dot{p} = \frac{1}{Z}\begin{bmatrix} xV_z - V_x \\ yV_z - V_y \end{bmatrix} + \begin{bmatrix} xy & -(1+x^2) & y \\ (1+y^2) & -xy & -x \end{bmatrix}\Omega$$

Translational Flow

$$\dot{p} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} xV_z - V_x \\ yV_z - V_y \end{bmatrix} = \frac{V_z}{Z}\begin{bmatrix} x - \frac{V_x}{V_z} \\ y - \frac{V_y}{V_z} \end{bmatrix}$$

- $\begin{bmatrix} \frac{V_x}{V_z} \\ \frac{V_y}{V_z} \end{bmatrix}$ is the **Focus of Expansion** (FOE)
    - Change in x or y coordinate is proportional to how far that coordinate is from the focus of expansion
    - FOE is the same as the epipole! (Image of the second camera from the first camera's frame)
        - If camera center moves from $(0,0,0)$ to (V_x, V_y, V_z) at $t = 1$, the image of the $t = 1$ camera at $t = 0$ is $\frac{V_x}{V_z}, \frac{V_y}{V_z}$
        - FOE does not depend on the scene, just the motion
        - Can also be thought of as the intersection of the flow vectors (same as all epipolar lines connecting at epipole)
- $\frac{V_z}{Z}$ is the **Inverse Time-To-Collision**
    - Change in x or y coordinate is inversely proportional to the time to collision of the camera with the Z plane of thfe object in the camera coordinate system

Rotational Flow

$$\dot{p} = \begin{bmatrix} xy & -(1+x^2) & y \\ (1+y^2) & -xy & -x \end{bmatrix} \Omega$$

- If we know angular velocity Ω (usually from IMU) can...
  - Compute optical flow $\dot{p}$ from the images with LK
  - Estimate rotational flow at each pixel $\dot{p}_{\text{rot}}$
  - Get translational flow $\dot{p}_{\text{trans}} = \dot{p} - \dot{p}_{\text{rot}}$

Finding FOE and TTC

- Finding FOE $\sim V$ up to scale ambiguity
  - The FOE can be written as $V \sim [V_x, V_y, V_z]$
  - For a point with known translational flow, its flow line is

$$p_1 \times (p_1 + \dot{p}_1) = p_1 \times \dot{p}_1$$

    - Recall: Equation of a line in projective space is $l = p_1 \times p_2$
    - Reduction comes from $p_1 \times p_1 = 0$
  - FOE is the intersection of all flow lines so

$$(p_1 \times \dot{p}_1)^T V = 0$$

  - Given $n \geq 2$ points and flows

$$\begin{pmatrix} (p_1 \times \dot{p_1})^T \\ (p_2 \times \dot{p_2})^T \\ \cdots \\ (p_n \times \dot{p_n})^T \end{pmatrix} V = 0$$

    - Solved equations like this many times before, take SVD, last right singular vector of V
    - NOTE: SVD is for solving the (left or right) null space of an over or underdetermined system $Ax = \vec{0}$ (or $xA = \vec{0}$ for left null space respectively) . Least squares/pseudo-inverse is for overdetermined systems $Ax = \vec{b}$
- Finding Time-To-Collision (TTC)
  - After computing FOE

$$\text{Inverse TTC} = \frac{V_z}{Z} = \frac{||\dot{p}_{\text{trans}}||}{||p - FOE||}$$

3D Motion from Depth Cameras (Known Z)

- If Z is known, $\dot{p}$ is linear in V and Ω
- We have 2 equations per point correspondence (the flow)
- If we have at least 3 optical flow vectors not on collinear points and their corresponding depths we can solve

$$A_{2n\times 6}\begin{bmatrix} V_{3\times 1} \\ \Omega_{3\times 1} \end{bmatrix} = \dot{p}$$

  - Typically do RANSAC over many sets of 3 flow vectors since $\dot{p}$ can be very noisy

---

Summary of Methods for Motion from Optical Flow

- Given optical flow and angular velocity Ω, can solve for $FOE \sim V$ with 2 2D-2D correspondences
- Given optical flow and depths Z, we can solve for V and Ω with 3 2D-2D correspondences
- Alone, we can solve for V and Ω with 5 correspondences (SfM)
- With a known 3D scene, we can solve for single-frame camera pose with 3 2D-3D correspondences (PnP)

Lecture 19 - Visual Odometry and ORB-Slam

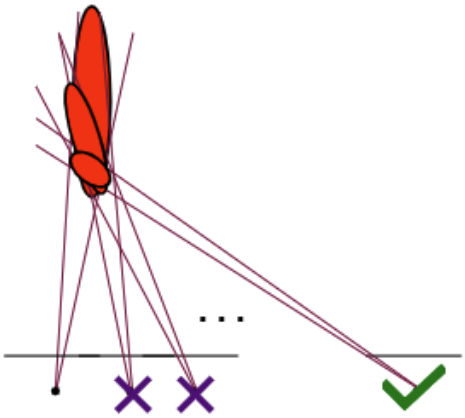
Visual Odometry

- Types
  - Odometry: Measuring how far you go by counting wheel rotations or steps (also known as path integration)
  - Inertial Odometry: Integration of velocity or acceleration measurements
  - Visual Odometry: Process of incrementally estimating your position and orientation with respect to an initial reference frame by tracking visual features in an unknown environment
- Why Visual Odometry?
  - Error from inertial approaches continues to grow larger as time goes on (double integration of acceleration), known as drift
- Why not just adapt SfM to more than 2 frames
  - Would also lead to drift
  - Need a new system that maintains some kind of consistency over longer durations

Visual Odometry Basic Pipeline

1. Do 2-View SfM between first 2 frames from video
    - Outputs:
        1. R_2, T_2 of camera 2 with respect to camera 1
        2. A 3D map with the 3D positions X_i of points seen in both frames
2. For next frame k, identify 2D-2D correspondences with the previous frame
    - These produce 2D-3D correspondences between this frame and the current map
    - Compute R_k, T_k with PnP
3. Improve triangulated 3D map X_i and expand it using correspondences that aren't yet in the map
    - Go back to Step 2

- NOTE: Larger baselines ($T_k \rightarrow T_{k+1}$) result in lower triangulation error. Pixels are discrete and the true point could lie anywhere in the discretization of the pixel. A wider baseline reduces this as seen below



ORB-SLAM

- Definitions

- **Map Points**: A list of 3D points that represent the map of the environment reconstructed from the key frames
- **Key Frames**: A subset of video frames that contain cues for localization and tracking. The current key frame should have some overlap with the previous key frame but also be sufficiently different to gain additional information

Map Initialization

1. Wait for a good keyframe
    1. Keep comparing between 1st frame and k^{th} until none of the following happens
        - Homographic frames (rank deficient system for estimation of essential matrix E)
            - Means scene is planar or there is no translation
        - Insufficient inliers for an estimate E after performing RANSAC
    2. Declare this as the next keyframe
2. Store keyframe features and poses computed with 2-view SfM
3. Triangulate and initialize a 3D map with these localized points

Tracking

1. Extract ORB features with previous keyframe that are already in the 3D map
2. Use PnP with RANSAC to estimate the camera pose with respect to the last keyframe
3. Given the camera pose, project the unmatched map points from the previous key frame into the current frame and search for more features correspondences
4. Refine PnP with "pose-only bundle adjustment" (out of scope)
5. Define this frame as a key frame if both:
    1. At least 20 frames have passed since the last key frame or the current frame tracks fewer than 100 map points
    2. Fewer than 90% of the map points tracked by the current frame are also tracked by the reference key frame

Local Mapping

1. Project local map points visible in several neighboring keyframes into the current frame to search for more features correspondences
    1. Refine for camera pose further
    2. Triangulate new correspondences to grow the map further
2. Local bundle adjustment to refine these and achieve optimal reconstruction in the neighborhood of the current camera pose

- Overall: Lots of quality checking to avoid redundant keyframes and low-quality correspondences

Loop Closure

1. How to know whether I've returned to a place I've visited?
    - Deep learning or bag of visual words to find past keyframes that look similar
    - If visual similarity candidate is found, run a geometric consistency check: Align 3D points from current frame and this keyframe and check that they follow a similarity transformation
2. If loop closure declared, run bundle adjustment to update all poses in essential graph

Bundle Adjustment

- Used to refine camera poses and 3D points starting from some pretty good initialization
- Minimize reprojection error: The sum of errors between 2D observations and the reprojected 2D points

$$\text{argmin}_{\{X_n\}_{n=1}^N, \{R_f, T_f\}_{f=1}^F} \sum_{n,f} d(x_{nf}, K_f[R_f|T_f]X_n)$$

    - x_{nf} is the observed n^{th} point in frame f
    - R_f, T_f are the extrinsics for frame f
    - X_n is the 3D estimation of the observed point
    - We are solving for R_f, T_f, X_n for all frames f and points n

Lecture 20 - Optimization and Bundle Adjustment

- Reprojection Error:

$$\text{argmin}\{X_n\}_{n=1}^N, \{R_f, T_f\}_{f=1}^F \sum_{n,f} d(x_{nf}, K_f[R_f|T_f]X_n)$$

    - $d(\dots)$ is a distance function and is usually just 2D mean squared error after normalizing homogeneous coordinate to 1. This produces

$$\text{argmin}_{P,X} \sum_{n,f} ||P_f(X_n) - x_{nf}||^2$$

        - Where $P_f(X_n)$ is the reprojection of point n in frame f
        - x_{nf} is the observed point n in frame f
- Why can't we do bundle adjustment directly

    - Non-linear minimization: Requires good initializations and is very computationally expensive
- Framing the Problem
    - Set reference frame to be 1st keyframe
        - $R_1 = I$ and $T_1 = \vec{0}$
    - Number of unknowns

$$6(F-1) + 3N - 1$$

        - 3 DOF for rotation and 3 for translation in each frame f
        - X, Y, Z for each point n
        - -1 due to scale invariance
    - If every point is visible in every frame (the best case scenario) we have $2NF$ equations which are:
        - $$x_p^f = \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$
        - $$y_p^f = \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}$$
        - If equations are independent we need

$$2NF \geq 6F + 3N - 7$$

            - For 2 frames (SfM), we already know we need 5 correspondences
            - In practical cases, N and F are HUGE - on the order of thousands to hundreds of thousands for each
- Expanding Reprojection Error
    - $$\text{argmin}_{u=\{\{X_n\}_{n=1}^N, \{R_f, T_f\}_{f=1}^F\}} ||\epsilon(u)||_2^2$$
    - $$\epsilon^T = (\dots, x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}, y_p^f - \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}, \dots$$
    - Non-linear least squares problem

Optimization Crash Course

- First Order Optimization: Gradient Descent

- To minimize a differentiable function $f(x)$ over parameters x
1. Start from some initialization x
2. Compute the gradient $g = \frac{df}{dx}(x)$
3. Descent along that direction by some step-length $\alpha \rightarrow \delta x = -\alpha g$
4. Set $x \leftarrow x + \delta x$

- Second Order Optimization: Bird's Eye View
  - Minimize a twice-differentiable function $f(x)$ over parameters x
  1. Start with an initial estimate for x
  2. Locally approximate $f(x)$ using a second-order Taylor expansion

$$f(x + \delta x) \approx f(x) + g^T \delta x + \frac{1}{2} \delta x^T H \delta x$$

     - $g = \frac{df}{dx}(x)$
     - $H = \frac{d^2 f}{dx^2}(x)$ (the Hessian)
  3. Find a displacement that locally minimizes the quadratic local approximation of $f(x)$
- Newton/Newton-Raphson's Method
  - Same as above
  - Assume Hessian is positive, semi-definite (if not i.e. from poor initialization, quadratic function could be concave down and have an unbounded minimum). Then can find the minimum with

$$\frac{df}{dx}(x + \delta x) \approx H \delta x + g = 0$$

$$\delta x = -H^{-1} g$$

  - Iterating over the above update is called Newton's method
- Gauss-Newton Method for Least Squares
  - $g = \nabla_u f = 2 \sum_i \epsilon_i(u) \nabla_u \epsilon_i(u) = 2 J(u)^T \epsilon(u)$
    - $J_{ij} = \frac{\partial \epsilon_i}{\partial u_j}$
  - $H(u) = 2 \sum_i \nabla_u \epsilon_i(u) \nabla_u \epsilon_i(u) + \epsilon_i(u) \frac{\partial^2 \epsilon_i}{\partial^2 u^2}$ $\therefore$
  - $H(u) = 2 J(u)^T J(u) + 2 \sum_i \epsilon_i(u) \frac{\partial^2 \epsilon_i}{\partial u^2}$
  - We will just ignore the hard-to-compute quadratic terms so

$$H(u) \approx 2 J(u)^T J(u)$$

    - Holds true when error ϵ_i is small or the function is approximately linear
  - Applying above to Newton's update $\delta u = -H^{-1} g \rightarrow$ Gauss-Newton update/Normal equation

$$\delta u = -(J(u)^T J(u))^{-1} J(u)^T \epsilon(u)$$

- Basically pseudoinverse of jacobian with error! ($||Ju - \epsilon||_2^2$)

Applying to Bundle Adjustment

- Problems right out the gate
  - $J_{i,j} = \frac{\partial \epsilon_i}{\partial u_j}$
  - Dimension of ϵ is $2NF$
  - Dimension of unknown parameters u is $M = 6F + 3N - 7$
  - Size of J is $2NF$! Computing the inverse of that ($O(N^3)$ operation) is impossible at scale
- Workaround: Linear algebra implementation tricks
  - Block diagonal matrices can be inverted block by block
  - Try to set up $Ax = b$ with a triangular matrix A (can solve element by element)
  - Avoid matrix multiplications of large general matrices, sparse are much better

---

Lecture 21 - Bundle Adjustment pt. 2

Recap: Minimization Problem

$$\text{argmin}_{u=\{\{X_n\}_{n=1}^N, \{R_f, T_f\}_{f=1}^F\}} ||\epsilon(u)||_2^2$$

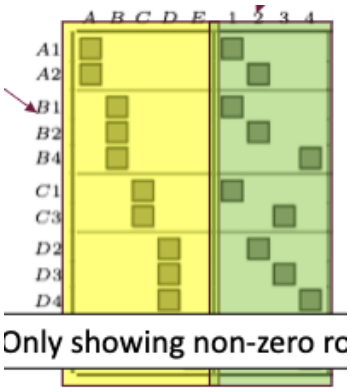
$$\epsilon^T = (\dots, x_p^f - \frac{R_{11}^f X_p + R_{12}^f Y_p + R_{13}^f Z_p + T_x}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}, y_p^f - \frac{R_{21}^f X_p + R_{22}^f Y_p + R_{23}^f Z_p + T_y}{R_{31}^f X_p + R_{32}^f Y_p + R_{33}^f Z_p + T_z}, \dots)$$

Use Newton's method for least squares can be implemented as the Gauss-Newton method

$$\delta u = -(J^T J)^{-1} J^T \epsilon$$

Creating Structure

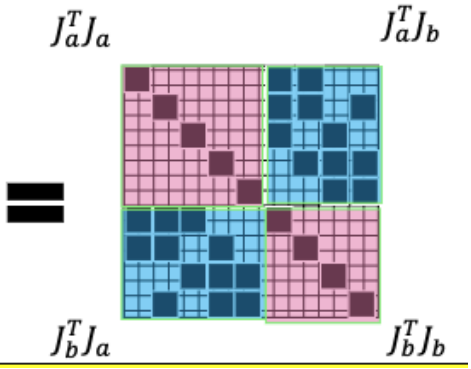
- Separate the parameters u into
  - Structure parameters $a_{3n \times 1}$
  - Camera parameters $b_{6f \times 1}$
  - So $u = \begin{bmatrix} a \\ b \end{bmatrix}$
- Structure of the Jacobian
  - Recall $J_{ij} = \frac{\partial \epsilon_i (\text{errors})}{\partial u_j (\text{parameters})}$
  - The Jacobian looks like the following



- where
- J_a is in yellow
  - $A1, A$ corresponds to the reprojection error of pt A in camera 1
  - All entries are zero except the reprojection of a pt visible in a camera view to that camera's view
  - NOTE: These are block matrices, each pt $A, B, \ldots$ has 3 DOF and camera $1, 2, \ldots$ has 6
  - NOTE: If every point were visible in every camera, there would be no non-zero rows (zero-rows omitted in image above)
- J_b is in green
  - $A_1, 1$ corresponds to ???

- $J^T J$
  - $J^T J$ can be written as a block matrix of the components defined above

$$J^T J = \begin{bmatrix} J_a^T J_a & J_a^T J_b \\ J_b^T J_a & J_b^T J_b \end{bmatrix} = \begin{bmatrix} U & W \\ W^T & V \end{bmatrix}$$

- 

  - $J^T J_a$ and $J^T J_b$ are block diagonal matrices
  - $J_a^T J_b = (J_b^T J_a)^T$
    - This matrix would be full if every point were visible in every camera
  - Adding this into our least squares update equation, we get

$$\begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = - \begin{bmatrix} J_a^T J_a & J_a^T J_b \\ J_b^T J_a & J_b^T J_b \end{bmatrix}^{-1} J^T \epsilon$$

- $$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = -J^T \epsilon$$

- Continuing to exploit structure
  - We can multiple by a carefully chosen matrix to make the problem easier
  - $$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = -J^T \epsilon = \begin{bmatrix} \epsilon'_a \\ \epsilon'_b \end{bmatrix}$$
  - $$\begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = \begin{bmatrix} I & -WV^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \epsilon'_a \\ \epsilon'_b \end{bmatrix}$$
  - $$\begin{bmatrix} U - WV^{-1}W^T & 0 \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta a \\ \delta b \end{bmatrix} = \begin{bmatrix} \epsilon'_a - WV^{-1}\epsilon'_b \\ \epsilon'_b \end{bmatrix}$$
  - This allows us to first solve for δa by

    $$(U - WV^{-1}W^T)\delta a = \epsilon'_a - WV^{-1}\epsilon'_b$$

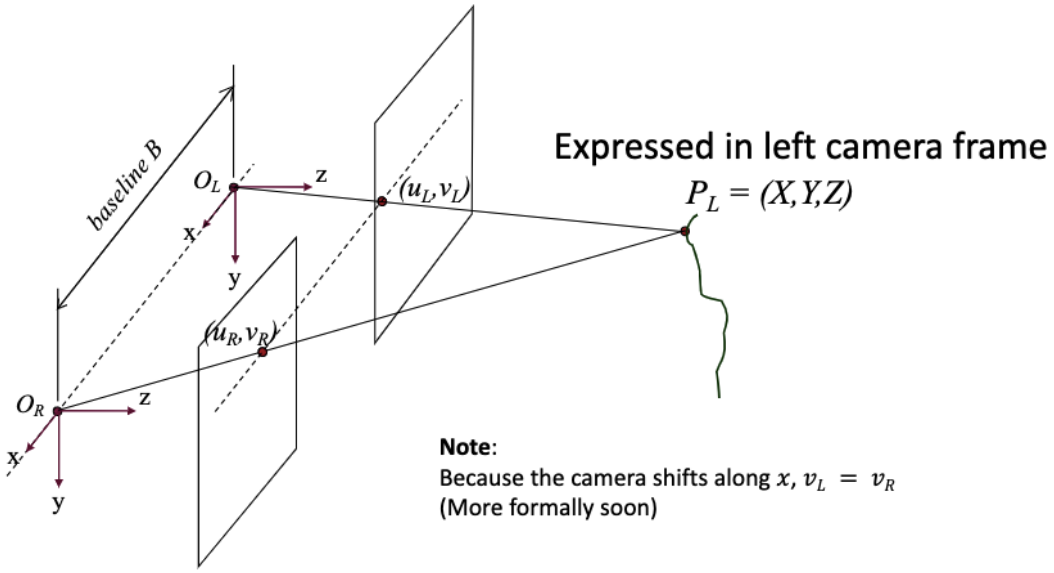
  - And then solve for δb with our calculation of δa

    $$V\delta b = \epsilon'_b - W^T \delta a$$

  - The only inverse we have to calculate is V^{-1} (remember V is block diagonal so computing its inverse is relatively trivial)

Multi-View Stereo

- Goal: Produce dense 3D reconstruction of the scene (not just a sparse set of points in it)
- Given:
  - Images of the same scene from multiple cameras
  - K, R, T for all cameras
- Frontoparallel Cameras



- $(u_l, v_l) = (f\frac{X}{Z}, f\frac{Y}{Z})$
- $(u_r, v_r) = (f\frac{X-B}{Z}, f\frac{Y}{Z})$
- Disparity d: Difference in projections between the two cameras
  - $d = u_l - u_r = f\frac{B}{Z}$
  - $Z = f\frac{B}{d}$
- We can use this to find correspondences! Even when the pixel movements are large (does not permit optical flow due to first order taylor expansion)
  - Correspondences lie on the same horizontal line (epipole is horizontal point at ∞, point correspondence must lie on epipolar line)

Lecture 22 - Multi-View Stereo

Recap

- With fronttoparallel cameras, can triangulate with...
  - $Z = f\frac{B}{d}$
  - B is the baseline between the two cameras
  - d is the disparity between the point correspondences ($u_l - u_r$)
- Point correspondences for fronttoparallel cameras displaced along the x-axis lie on the same row

Point Correspondences

- Setting Search Range of Disparity
  - Assume some minimum "depth", $Z_{\min}$
  - Set $d_{\max} = \frac{1}{Z_{\min}}$

    - Quantize the interval $[-d_{\max}, d_{\max}]$
- Components of Stereo Correspondence Matching
    - **Matching Criterion** (error function)
        - Quantify similarity of a pixel pair
        - Options
            - Direct RGB intensity difference
            - Correlation
    - **Aggregation Method**
        - How the error function is accumulated
        - Options
            - Pixelwise
            - Edgewise
            - Window-wise
    - **Optimization and Winner Selection**
        - How the final correspondences are determined
        - Options
            - Winner-take-all
            - Dynamic Programming
            - Graph Cuts
- A Simple Process
    - Matching Windows: For a given left window, we will select from various right windows along the scanline as candidate correspondences
    - Matching Windows with SSD Over the Window
        - w_L, w_R are corresponding $m \times m$ windows of pixels
            - Define the window function

$$W_m(x, y) = \{u, v | x - \frac{m}{2} \leq u \leq x + \frac{m}{2}, y - \frac{m}{2} \leq v \leq y + \frac{m}{2}\}$$

            - SSD Cost is then

$$C_r(x, y, d) = \sum_{(u,v) \in W_m(x,y)} [I_L(u, v) - I_R(u - d, v)]^2$$

    - Matching Windows with SSD + Winner-Take-All
        - Assign lowest SSD as match for each window

Stereo Rectification

- What to do if cameras aren't frontoparallel to start with $\rightarrow$ "rectify" them!
- Works best when cameras are roughly aligned
- Key Idea: Reproject image planes onto a common plane parallel to the line between camera centers. We need two homographies (which produce a single homography)
- Simplified Process
  1. Estimate E using 8-point algorithm
  2. Decompose E into R and $T \approx e$
  3. Build R_{rect} from e
  4. Set $R_1 = R_{\text{rect}}$ and $R_2 = RR_{\text{rect}}$
  5. On left and right camera pixel planes, apply the homographies KR_1 and KR_2 respectively
- Calculating R_{rect}
  - Done by setting the epipole to ∞
  - Can use the idea of $r_1 = e_1 = \frac{T}{||T||}$ and r_2, r_3 are orthogonal to produce the homography
  - Proof
    - Let

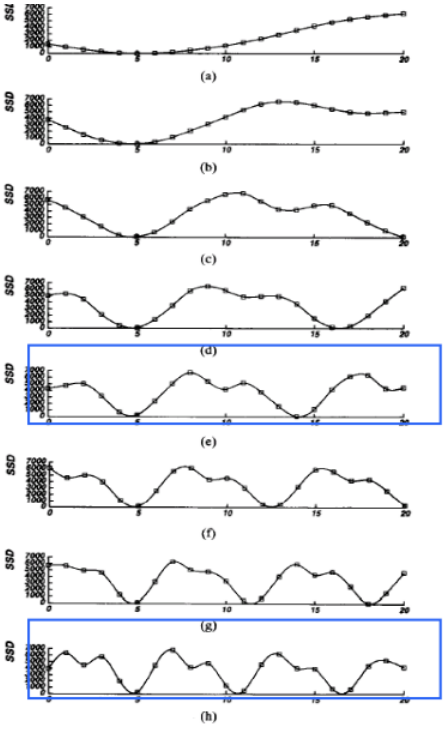
$$R_{\text{rect}} = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix}$$

    - $r_1 = e_1 = \frac{T}{||T||}$
    - $r_2 = -\frac{1}{\sqrt{T_x^2 + T_y^2}}[-T_y, T_x, 0]$
    - $r_3 = r_1 \times r_2$
    - This satisfies the property that the homography of the epipole is the point at positive x infinity and minimizes the necessary rotation (there are infinite solutions for this)

Multi-View Stereo

- Problem Formulation: Given several images of the same object or scene, compute a representation of its 3D shape
  - Input: Calibrated images from several viewpoints with known intrinsics, extrinsics, and projection matrices
  - Output: 3D object model
- Exploiting Multiple Neighbors

- Can match windows using more than 1 neighboring view to get a stronger match signal
- Possible Options for Picking Neighbors
  - Pick neighbors based on baseline
    - Problem: Too small of a baseline results in large depth error and too large of a baseline is a difficult search problem
  - Best: Combine All
    - Add all graphs for pixel matching score vs depth and pick the minimum



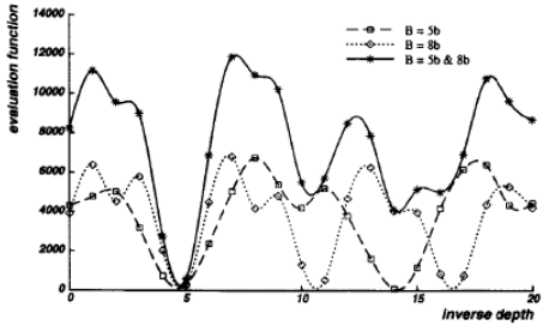
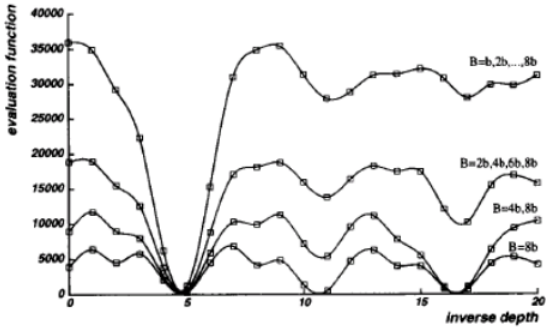


Fig. 6. Combining two stereo pairs with different baselines.



- Plane-Sweep Stereo
  - An efficient way to compute multi-view stereo
  - Algorithm
    - Sweep family of planes parallel to reference camera image plane
    - Reproject neighbors onto each plane (via homography) and compare reprojections
    - At each iteration, we pretend that each camera's entire image was of a single plane at depth z from the reference camera and backproject onto the plane from each camera and see how much the neighbors agree for each pixel
    - When neighbors agree at a pixel, that pixel is likely to have depth z_0. The pixel's cost for depth z is the variance over the neighbor backprojections
    - z is then incremented and the next iteration begins
    - At the end of the "sweep" over z, the min-cost z is selected for each pixel, to form the full depth map
  - How to backproject onto plane $Z = z_0$

- At each iteration, pretend that each camera's entire image was of a single plane at depth z_0 from the reference camera, and backproject onto that plane from each camera
- Imaging the world plane $Z = 0$ corresponds to the homography from world to image $H = K[r_1, r_2, t]$
- How does this homography change when imaging the plane $Z = z_0$? Idk yet tbd