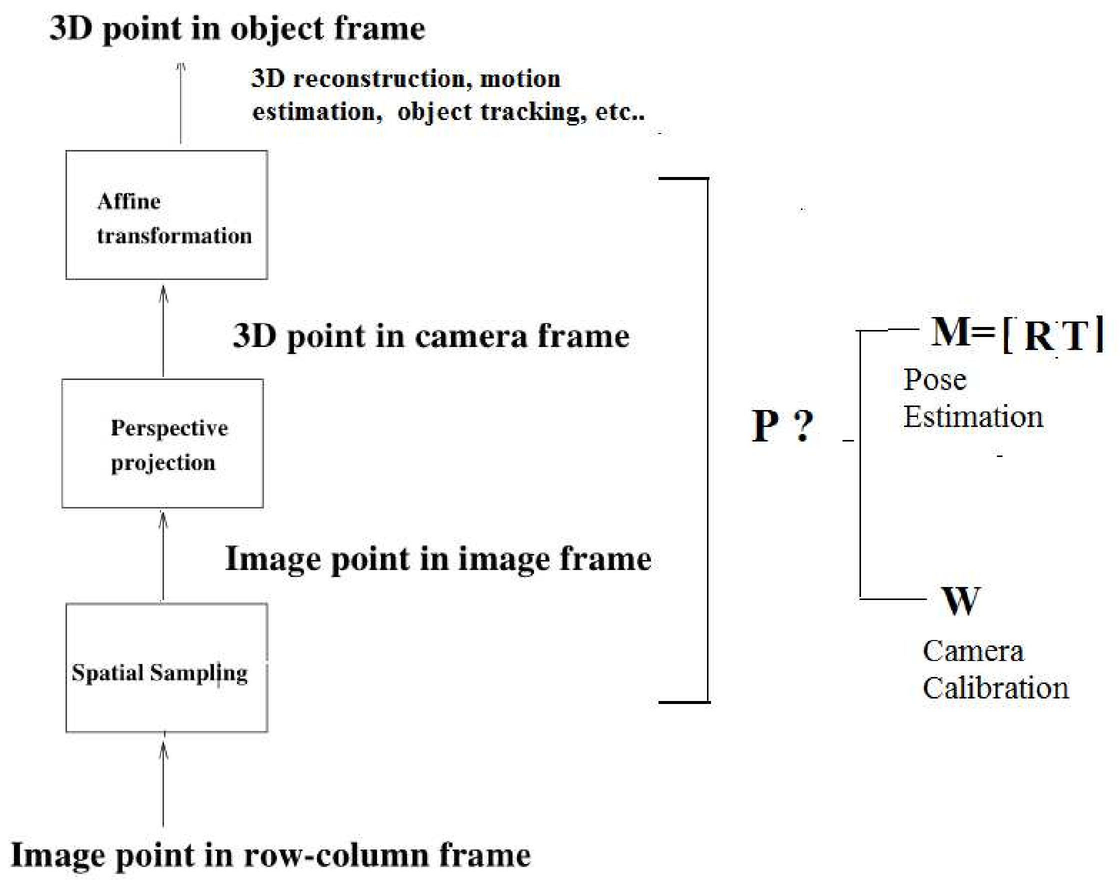
Camera Calibration and Pose Estimation



Camera Calibration and Pose Estimation

The purpose of camera calibration is to determine intrinsic camera parameters: c_0, r_0, s_x, s_y, and f. Camera calibration is also referred to as interior orientation problem in photogrammetry.

The goal of pose estimation is to determine exterior camera parameters: ω, ϕ, κ, t_x, t_y, and t_z. In other words, pose estimation is to determine the position and orientation of the object coordinate frame relative to the camera coordinate frame or vice versus.

Camera Calibration

The purpose of camera calibration is to determine the intrinsic camera parameters (c_0, r_0), f, s_x, s_y, skew parameter ($s = \cos\alpha$), and the lens distortion (radial distortion coefficient k_1). Skew parameter is a function of the skew angle between the c and r axes of a pixel. For most CCD camera, we have rectangular pixel. The skew angle is 90. The skew parameter is therefore zero. k_1 is often very small and can be assumed to be zero. The intrinsic camera matrix is

$$W = \begin{pmatrix} f s_x & \cos\alpha & c_0 \\ 0 & \frac{f s_y}{\sin\alpha} & r_0 \\ 0 & 0 & 1 \end{pmatrix}$$

If we assume $\alpha = 90$, then we have

$$W = \begin{pmatrix} fs_x & 0 & c_0 \\ 0 & fs_y & r_0 \\ 0 & 0 & 1 \end{pmatrix}$$

fs_x and fs_y are collectively treated as two separate parameters instead of three parameters.

Camera Calibration Methods

- Conventional method: use a calibration pattern that consists of a 2D and 3D data to compute camera parameters. It needs a single image of the calibration pattern.
- Camera self-calibration: only need 2D image data of the calibration pattern but need multiple 2D image data from multiple views. Problem is often less constrained than the conventional method.

Conventional Camera Calibration

Given a 3D calibration pattern, extract image features from the calibration pattern. Use 2D image features and the corresponding 3D features on the calibration pattern. The features can be points, lines, or curves or their combination.

- Determine the projection matrix P
- Derive camera parameters from P

$$P_{full} = \begin{pmatrix} s_x f \mathbf{r}_1 + c_0 \mathbf{r}_3 & s_x f t_x + c_0 t_z \\ s_y f \mathbf{r}_2 + r_0 \mathbf{r}_3 & s_y f t_y + r_0 t_z \\ \mathbf{r}_3 & t_z \end{pmatrix}$$

Basics in Linear Algebra, Calculus, and Optimization

- 3 spaces of a matrix - column, row, and null space.
- Range, determinant, and rank of a matrix
- Eigen-value and eigen-vector of a matrix
- Pseudo-inverse (generalized) inverse of a general matrix
- Eiegn decomposition for square matrix, singular value decomposition (SVD) for general matrix, and their relationships.
- A system of linear equations and its linear least squares solution
- Multivariate calculus
- Gradient based first order optimization method

Compute P: Linear Method using 2D/3D Points

Given image points $m_i^{2\times 1} = (c_i, r_i)^T$ and the corresponding 3D points $M_i^{3\times 1} = (x_i\ y_i\ z_i)^T$, where i=1, …, N, we want to compute P. Let P be represented as

$$P = \begin{pmatrix} p_1^T & p_{14} \\ p_2^T & p_{24} \\ p_3^T & p_{34} \end{pmatrix} \tag{1}$$

where p_i, i=1,2,3 are 3×1 vectors and p_{i4}, i=1,2,3, are scalers.

Compute P: Linear Method (cont'd)

Then for each pair of 2D-3D points, we have

$$\begin{aligned} M_i^T p_1 + p_{14} - c_i M_i^T p_3 - c_i p_{34} &= 0 \\ M_i^T p_2 + p_{24} - r_i M_i^T p_3 - r_i p_{34} &= 0 \end{aligned}$$

Each pair of 2D/3D points gives two equations on the projection matrix. For 12 unknowns in the projection matrix, we need a minimum of N=6 pairs of 2D/3D points. If N < 6, there is an infinite number of solution. For $N \geq 6$ points, depending on the configurations of the points, there can be one unique exact solutions.

Compute P: Linear Method (cont'd)

We can setup a system of linear equations involving matrix A and vector V, where A is a 12×12 matrix depending only on the 3-D and 2-D coordinates of the calibration points, and V is a 12×1 vector $(p_1^T\ p_{14}\ p_2^T\ p_{24}\ p_3^T\ p_{34})^T$.

$$A = \begin{pmatrix} M_1^T & 1 & \vec{0} & 0 & -c_1 M_1^T & -c_1 \\ \vec{0} & 0 & M_1^T & 1 & -r_1 M_1^T & -r_1 \\ & & \vdots & & & \\ M_N^T & 1 & \vec{0} & 0 & -c_N M_N^T & -c_N \\ \vec{0} & 0 & M_N^T & 1 & -r_N M_N^T & -r_N \end{pmatrix}$$

where $\vec{0}^{1\times 3} = [0\ 0\ 0]$

Compute P: Linear Method (cont'd)

To have a robust and accurate solution, we choose N such that $N > 6$. In this case, we can find one unique linear least squares solution (depending on the condition of A matrix) by minimizing $\epsilon^2 = ||AV||_2^2$, i.e.,

$$V^* = \arg\min_V ||AV||_2^2$$

Taking derivative of $\epsilon^2 = ||AV||_2^2$ w.r.t V and setting it to zero yields

$$\begin{aligned} \frac{\partial \epsilon^2}{\partial V} &= \frac{\partial (AV)^T(AV)}{\partial V} = 0 \\ A^T AV &= 0 \end{aligned}$$

Solution to V lies in the null space of $A^T A$. Note the null space of $A^T A$ is the same as the null space A. Hence, $A^T AV = 0$ is the same as $AV = 0$

Rank of A

The solutions depend on the rank of A. In general, given $AV = 0$, the rank of A is less than 12, which means the solution is not unique. But due to effect of noise and locational errors, A may be full rank, which means only the trivial solution $V = 0$ exists.

Besides the number of points, the rank of A may also change for certain special configurations of input 3D points, for example collinear points, coplanar points, etc. The issue is of practical relevance.

Rank of A

Rank of A is a function of the input points configurations (see section 3.4.1.3 of Faugeras). If 3D points are coplanar, then rank(A) < 11 (in fact, it equals 8), which means there is an infinite number of solutions. Faugeras proves that 1) in general for non-coplanar of more than 6 points, Rank(A)=11; 2) for coplanar points (N $\geq$4), rank(A)=8 since three points are needed to determine a plane (11-3=8).

How about the rank of A if points are located a sphere or on the planes that are orthogonal or parallel to each other ? Hint: how many points are needed to determine a sphere ?

Linear Solution 1

Given $\epsilon^2 = ||AV||_2^2$, the linear least-squares solution is

$$V^* = \arg\min_V \epsilon^2, \text{ which leads to}$$

$$A^T AV = 0$$

Solution to $A^T AV = 0$ is not unique (up to a scale factor). It lies in the null space of $A^T A$. As the null space of $A^T A$ is the same as that of A, we can find the null space of A. If rank(A)=11, then V is the only null vector of A, multiplied by a scaler. If on the other hand, rank(A) $\leq$ 11, the the solution to X is the linear combinations of all null vectors of A. The null space of A can be obtained by performing SVD on A, yielding

$$A^{m\times n} = U^{m\times m} D^{m\times n} (S^T)^{n\times n}$$

where U and S are orthnormal matrix and D is a diagonal matrix, whose diagonal entries are called singular values and are arranged in descending order.

If rank(A)=11, the only null vector A is the last column (corresponding to the smallest singular value) of S matrix.

Alternatively, we can also solve V by directly minimizing $||AV||_2^2$, which yields $(A^TA)V = 0$ or $(A^TA)V = \lambda V$, where $\lambda = 0$. As a result, solution to V is the eign vector of matrix (A^TA), corresponding to zero eigen value. This implies that the eignvectors of A^TA correspond to the columns of the S matrix.

Note V is solved only up to a scale factor. The scale factor can be recovered using the fact that $||p_3||_2^2 = 1$. Let the null vector of A be V'. The scale factor $\alpha = \sqrt{\frac{1}{V'^2(9)+V'^2(10)+V'^2(11)}}$. Hence, $V = \alpha V'$.

Linear Solution 2

Let $A = [B \;\; b]$, where

$$B = \begin{pmatrix} M_1^T & 1 & \vec{0} & 0 & -c_1 M_1^T \\ \vec{0} & 0 & M_1^T & 1 & -r_1 M_1^T \\ & & \vdots & & \\ M_N^T & 1 & \vec{0} & 0 & -c_N M_N^T \\ \vec{0} & 0 & M_N^T & 1 & -r_N M_N^T \end{pmatrix}$$

$$b = (-c_1 \;\; -r_1 \;\; \ldots \;\; -c_N \;\; -r_N)^T$$

$$\begin{aligned} V &= p_{34} \begin{pmatrix} Y \\ 1 \end{pmatrix} \\ Y &= (p_1^T \; p_{14} \; p_2^T \; p_{24} \; p_3^T)^T / p_{34} \end{aligned}$$

Then, $AV = p_{34}(BY + b)$. Since p_{34} is a constant, minimizing $||AV||_2^2$ is the same as minimizing $||BY + b||_2^2$, whose solution is $Y = -(B^T B)^{-1} B^T b$. The rank of matrix B must be eleven.

Linear Solution 2 (cont'd)

The solution to Y is up to a scale factor p_{34}. To recover the scale factor, we can use the fact that $|p_3| = 1$. The scale factor p_{34} can be recovered as $p_{34} = \frac{1}{\sqrt{Y^2(9)+Y^2(10)+Y^2(11)}}$, where $Y(9)$, $Y(10)$, and $Y(11)$ are the last 3 elements of Y. The final projection matrix (vector) is therefore equal to

$$V = \begin{pmatrix} p_{34}Y \\ p_{34} \end{pmatrix}$$

Linear Solution 3

Imposing the orthonormal constraint, $R^T = R^{-1}$, i.e., minimize

$$||AV||^2$$

subject to $R^T = R^{-1}$. Solution to this problem is non-linear !.

Linear Solution 3 (cont'd)

To yield a linear solution, we can impose one of the normal constraints, i.e., $||p_3||^2 = 1$, then the problem is converted to a constrained linear least-squares problem. That is, minimize $||AV||_2^2$ subject to $||p_3||_2^2 = 1$.

$$\epsilon^2 = ||AV||_2^2 + \lambda(||p_3||_2^2 - 1) \tag{2}$$

Note : the constraint is imposed in a hard way, i.e., it must be exactly satisfied. Hence, λ is not a hyper-parameter but another variable that must be solved as well (i.e., its first order derivative must be 0 too). KKT condition decides if there exists a hard constraint solution. If not, the constraint must be imposed a soft way, whereby the second term is changed to $(||p_3||_2^2 - 1)^2$ and λ becomes a hyper-parameter and is not required to be solved.

Decomposing A into two matrices B and C, and V into Y and Z

$$A = (B \;\; C)$$

$$V = \begin{pmatrix} Y \\ Z \end{pmatrix}$$

$$B^{2N\times 9} = \begin{pmatrix} M_1^T & 1 & \vec{0} & 0 & -c_1 \\ \vec{0} & 0 & M_1^T & 1 & -r_1 \\ & \vdots & & & \\ M_N^T & 1 & \vec{0} & 0 & -c_N \\ \vec{0} & 0 & M_N^T & 1 & -r_N \end{pmatrix} \qquad C^{2N\times 3} = \begin{pmatrix} -c_1 M_1^T \\ -r_1 M_1^T \\ \vdots \\ -c_N M_N^T \\ -r_N M_N^T \end{pmatrix}$$

$$Y = \begin{pmatrix} p_1 \\ p_{14} \\ p_2 \\ p_{24} \\ p_{34} \end{pmatrix} \qquad Z = p_3$$

Then equation 2 can be rewritten as

$$\epsilon^2 = ||BY + CZ||_2^2 + \lambda(||Z||_2^2 - 1)$$

Taking partial derivatives of ϵ^2 with respect to Y and Z and setting them to zeros yield

$$\frac{\partial \epsilon^2}{\partial Y} = 0 \Longrightarrow Y = -(B^T B)^{-1} B^T C Z$$

$$\frac{\partial \epsilon^2}{\partial Z} = 0 \Longrightarrow C^T (B(B^T B)^{-1} B^T - I) C Z = \lambda Z$$

Apparently, the solution to Z is the eigenvector of matrix $C^T (B(B^T B)^{-1} B^T - I) C$. Given Z, we can then obtain solution to Y.

Substituting Y into $||BY+CZ||^2$ leads to

$$
\begin{aligned}
& ||BY+CZ||^2 \\
= \; & ||-B(B^TB)^{-1}B^TCZ+CZ||^2 \\
= \; & ||(B(B^TB)^{-1}B^T-I)CZ||^2 \\
= \; & Z^TC^T(B(B^TB)^{-1}B^T-I)^T(B(B^TB)^{-1}B^T-I)CZ \\
= \; & Z^TC^T(B(B^TB)^{-1}B^T-I)CZ \\
= \; & Z^T\lambda Z \\
= \; & \lambda
\end{aligned}
$$

This proves that solution to Z corresponds to the eigen vector of the smallest positive eigenvalue of matrix
$C^T(B(B^TB)^{-1}B^T-I)C$.

Note
$(B(B^TB)^{-1}B^T-I)(I-B(B^TB)^{-1}B^T)=(B(B^TB)^{-1}B^T-I)$

Other Linear Techniques

- Another new method at https://www.ecse.rpi.edu/˜qji/CV/new_method.pdf

https://sites.ecse.rpi.edu/∼qji/CV/new_method.pdf

Note all linear methods we introduce belong to the category of direct linear transformation (DLT) formulation or the DLT algorithm.

Robust Linear Method with RANSAC

The linear LSQ method is sensitive to image errors and outliers. One solution is to use a robust method. The most commonly used robust method in CV is the RANSAC (Random Sample Consensus) method. It works as follows

- Step 1: Randomly pick a subset of K points from N (K>6) points in the image and compute the projection matrix P using the selected points.
- Step 2: For each of the remaining points in the image, compute its projection error using the P computed from step 1. If it is within a threshold distance, increment a counter of the number of points (the "inliers") that agree with the hypothesized P.
- Step 3: Repeat Steps 1 and 2 for a sufficient number of times

[a], and then select the subset of points corresponding to the P with the largest count.

- Step 4: Using the subset of points selected in Step 3 plus all of the other inlier points which contributed to the largest count, recompute the best P for all of these points.

More information on the RANSAC method may be found at https://opencv.org/evaluating-opencvs-new-ransacs/

[a]the exact number of times is determined by the required probability that one of subset does not contain the outliers

Unconstrained Non-linear Method

Let the 3D points be $M_i = (x_i, y_i, z_i)^T$ and the corresponding image points be $m_i = (c_i, r_i)^T$ for $i = 1, 2, \ldots, N$. From the perspective projection equation $\lambda(c_i, r_i, 1)^T = P(x_i, y_i, z_i, 1)^T$, we can construct the loss function as the sum of the projection (geometric) errors

$$\epsilon^2 = \sum_{i=1}^{N} (\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)^2 + (\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)^2 \qquad (3)$$

Note other loss function such as the Sampson error (section 3.2.6 of Hartley's book) function can be used. Sampson error represents the first order approximation to the geometric error in equation 3.

Constrained Non-linear Method 1

Introducing the soft constraint that $||p_3||_2^2 = 1$, the loss function to minimize is changed to

$$\begin{aligned} \epsilon^2 &= \sum_{i=1}^{N}(\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)^2 + (\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)^2 \\ &+ \lambda_1(||p_3||_2^2 - 1)^2 \end{aligned} \tag{4}$$

Note where $\lambda 1$ is the lagrangian multiplier and it is not a variable to estimate but a hyper-parameter to tune.

Constrained Non-linear Method 2

Imposing the full the orthonormal constraints about R as $||p_3||_2^2 = 1$ and $(p_1 \times p_3) \cdot (p_2 \times p_3) = 0$ in a soft way yields a new loss function

$$\begin{aligned} \epsilon^2 &= \sum_{i=1}^{N} (\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)^2 + (\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)^2 \\ &+ \lambda_1(||p_3||_2^2 - 1)^2 + \lambda_2[(p_1 \times p_3) \cdot (p_2 \times p_3)]^2 \end{aligned} \quad (5)$$

where the lagrangian multipliers (λs) are hyper-parameters to tune.

Non-linear Least-Squares (NLS) Solutions

The solutions to non-linear least-squares problems are typically iterative, starting from an initial guess.

- The gradient descent method (first order, applicable to any objective function)
- Gauss-Newton method (or Levenberg-Marquardt) (first order, applicable to only NLQ)
- The Newton method (second order, applicable to any objective function)

Given an initial p_i, non-linear methods iteratively update p_i, i.e.,

$$p_i^t = p_i^{t-1} + \eta \Delta p_i$$

They differ in the way Δp_i is computed.

Gradient Descent Method

First order gradient descent may be used to solve P iteratively. For the loss function in Eq. 5, we can solve p_3 as follows

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

where η is the learning rate and $\Delta p_3 = -\nabla_{p_3}\epsilon^2$, the gradient of p_3, is computed as follows

$$\begin{aligned}\nabla_{p_3}\epsilon^2 &= \sum_{i=1}^{N} \frac{\partial(\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)^2}{\partial p_3} + \frac{\partial(\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)^2}{\partial p_3} \\ &+ \lambda_1 \frac{\partial(||p_3||_2^2 - 1)^2}{\partial p_3} + \lambda_2 \frac{\partial[(p_1 \times p_3)\cdot(p_2 \times p_3)]^2}{\partial p_3}\end{aligned}$$

where

$$\frac{\partial(\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)^2}{\partial p_3} = -2(\frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i)\frac{(M_i^T p_1 + p_{14})M_i}{(M_i^T p_3 + p_{34})^2}$$

$$\frac{\partial(\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)^2}{\partial p_3} = -2(\frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i)\frac{(M_i^T p_2 + p_{24})M_i}{(M_i^T p_3 + p_{34})^2}$$

$$\frac{\partial(||p_3||_2^2 - 1)^2}{\partial p_3} = 4(p_3^T p_3 - 1)p_3$$

$$\begin{aligned} \frac{\partial[(p_1 \times p_3) \cdot (p_2 \times p_3)]^2}{\partial p_3} &= \frac{\partial[(p_1 \times p_3)^T (p_2 \times p_3)]^2}{\partial p_3} \\ = \quad & 2(p_1 \times p_3)^T (p_2 \times p_3)[\frac{\partial(p_1 \times p_3)}{\partial p_3}(p_2 \times p_3) \\ + \quad & \frac{\partial(p_2 \times p_3)}{\partial p_3}(p_1 \times p_3)] \end{aligned}$$

where

$$\frac{\partial(p_1 \times p_3)}{\partial p_3} = (\frac{\partial p_1}{\partial p_3}) \times p_3^T + p_1^T \times (\frac{\partial p_3}{\partial p_3})^T$$

$$= \begin{pmatrix} p_{11} \\ p_{12} \\ p_{13} \end{pmatrix}^T \times \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} (p_{11} & p_{12} & p_{13}) \times (1 & 0 & 0) \\ (p_{11} & p_{12} & p_{13}) \times (0 & 1 & 0) \\ (p_{11} & p_{12} & p_{13}) \times (0 & 0 & 1) \end{pmatrix} = \begin{pmatrix} 0 & p_{13} & -p_{12} \\ -p_{13} & 0 & p_{11} \\ p_{12} & -p_{11} & 0 \end{pmatrix}$$

Note: For $X \in \mathcal{R}^n$, $Y \in \mathcal{R}^n$, and $Z \in \mathcal{R}^K$, $\frac{\partial(X \times Y)}{\partial Z} = \frac{\partial X}{\partial Z} \times Y^T + X^T \times \frac{\partial Y}{\partial Z}$, where the cross product between a row vector and a matrix is computed by computing the cross product between the row vector with each row of the matrix as shown above.

Alternatively, the vector cross product can be changed to matrix and vector product

$$\frac{\partial(p_1 \times p_3)}{\partial p_3} = \frac{\partial(P_1 p_3)}{\partial p_3} = P_1^T$$

where P_1 is the skew matrix of vector p_1

$$P_1 = \begin{pmatrix} 0 & -p_{13} & p_{12} \\ p_{13} & 0 & -p_{11} \\ -p_{12} & p_{11} & 0 \end{pmatrix} \tag{6}$$

Similarly, we have

$$\frac{\partial(p_2 \times p_3)}{\partial p_3} = \frac{\partial(P_2 p_3)}{\partial p_3} = P_2^T$$

Similarly, we can follow the same procedure to compute the gradients for p_1, p_{14}, p_2, p_{24} and p_{34}.

Further information on multivariate calculus, see https://en.wikipedia.org/wiki/Matrix_calculus .

Gauss-Newton Method-unconstrained

For a loss function that involves minimization of the sum of squared function values, i.e., the least-squared problem, we can employ the Gauss-Newton method. Let
$f(x_i, y_i, z_i, c_i, r_i, P) = \frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} - c_i$ and
$g(x_i, y_i, z_i, c_i, r_i, P) = \frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} - r_i$, the loss function in Eq. 3 can be written as

$$\epsilon^2 = \sum_{i=1}^{N} f^2(x_i, y_i, z_i, c_i, r_i, P) + g^2(x_i, y_i, z_i, c_i, r_i, P) \qquad (7)$$

We can update p_3 using

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

where Δp_3 can be computed by performing the first order Taylor expansion of $f()$ and $g()$, and substituting them into Eq. 7 and taking the derivative of the loss function wrt Δp_3 and setting it to zero, yielding

$$\Delta p_3 = -[(\frac{\partial f}{\partial p_3})(\frac{\partial f}{\partial p_3})^T + (\frac{\partial g}{\partial p_3})(\frac{\partial g}{\partial p_3})^T]^{-1}[\frac{\partial f}{\partial p_3} f() + \frac{\partial g}{\partial p_3} g()]$$

We can similarly solve for p_1 and p_2.

Gauss-Newton Method-constrained

For constrained non-linear least squares problem, let $C(p_3)$ be the equality constraint on p_3. The loss function changes to
$\epsilon^2 = \sum_{i=1}^{N} f^2(x_i, y_i, z_i, c_i, r_i, P) + g^2(x_i, y_i, z_i, c_i, r_i, P) + \lambda C(p_3)$

The Gauss-Newton method can be changed as follows. For p_3 in above equation, we have

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

where Δp_3 can be computed as follows

$$\Delta p_3 = -[(\frac{\partial f}{\partial p_3})(\frac{\partial f}{\partial p_3})^T + (\frac{\partial g}{\partial p_3})(\frac{\partial g}{\partial p_3})^T]^{-1}[\frac{\partial f}{\partial p_3} f() + \frac{\partial g}{\partial p_3} g() + \lambda \frac{\partial C(p_3)}{\partial p_3}]$$

We can similarly solve for p_1 and p_2.

Levenberg Marquardt (LM) algorithm

LM algorithm is an improved version of Gauss-Newton method, i.e., for p_3, we have

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

where Δp_3 , in the unconstrained case, is computed

$$\Delta p_3 = -[(\frac{\partial f}{\partial p_3})(\frac{\partial f}{\partial p_3})^T + (\frac{\partial g}{\partial p_3})(\frac{\partial g}{\partial p_3})^T + \alpha \mathbf{I}]^{-1}[\frac{\partial f}{\partial p_3} f() + \frac{\partial g}{\partial p_3} g()]$$

where $\mathbf{I}$ is an identity matrix and α is a damping factor that varies with each iteration. The iteration starts with a large α and gradually reduces α value as the iteration goes. With a small α, the LM algorithm becomes Gauss-Newton method and becomes the gradient descent method with a large α.

Newton Method-unconstrained

Newton method is the second order method and it assumes the loss function is second order differentiable. Given a loss function $\epsilon^2 = f(x_i, y_i, z_i, c_i, r_i, P)$, it performs the second order Taylor expansion of $f()$ wrt to p_3 yields

$$\begin{aligned} f(x_i, y_i, z_i, c_i, r_i, P^t) &= f(x_i, y_i, z_i, c_i, r_i, P^{t-1}) + \frac{\partial f()}{\partial p_3}\Delta p_3 \\ &+ \frac{1}{2}\frac{\partial^2 f()}{\partial p_3^2}\Delta p_3 \Delta p_3^T \end{aligned}$$

Taking the partial derivative of $f()$ wrt Δp_3 and set to zero, yielding

$$\frac{\partial f()}{\partial \Delta p_3} = \frac{\partial f()}{\partial p_3} + (\frac{\partial^2 f()}{\partial p_3^2})\Delta p_3 = 0$$

Hence

$$\Delta p_3 = -(\frac{\partial^2 f()}{\partial p_3^2})^{-1}(\frac{\partial f()}{\partial p_3})$$

where $\frac{\partial^2 f()}{\partial p_3^2}$ is the Hessian matrix and $\frac{\partial f()}{\partial p_3}$ is the Jacobian matrix.

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

Newton Method-constrained case

For constrained minimization, the loss function becomes
$\epsilon^2 = f(x_i, y_i, z_i, c_i, r_i, P) + \lambda C(P)$

$$\Delta p_3 = -(\frac{\partial^2 f()}{\partial p_3^2} + \lambda \frac{\partial^2 C(P)}{\partial p_3^2})^{-1}(\frac{\partial f()}{\partial p_3} + \lambda \frac{\partial C(P)}{\partial p_3})$$

where $\frac{\partial^2 f()}{\partial p_3^2}$ is the Hessian matrix and $\frac{\partial f()}{\partial p_3}$ is the Jacobian matrix.

$$p_3^t = p_3^{t-1} + \eta \Delta p_3$$

Separable Non-linear Least-squares Method

Let $\hat{c}_i(\mathbf{p}_1, \mathbf{p}_3) = \frac{M_i^T p_3 + p_{14}}{M_i^T p_3 + p_{34}} = \frac{a_i(\mathbf{p}_1)}{b_i(\mathbf{p}_3)}$ and

$\hat{r}_i(\mathbf{p}_2, \mathbf{p}_3) = \frac{M_i^T p_2 + p_{24}}{M_i^T p_3 + p_{34}} = \frac{c_i(\mathbf{p}_2)}{b_i(\mathbf{p}_3)}$

Solve p_1 and p_2 in terms of p_3 and substituting their solutions to equation above. We can then solve for p_3 non-linearly by minimizing the projection error. It repeats until convergence.

Non-Linear Direct Solution to Camera Parameters

Another way of solving this problem is to perform minimization directly with respect to the intrinsic and extrinsic parameters.

Let

$$\begin{aligned} \Theta &= (c_0 \; r_0 \; f \; s_x \; s_y \; \omega \; \phi \; \kappa \; t_x \; t_y \; t_z)^T \\ f(\Theta, M_i) &= \frac{M_i^T p_1 + p_{14}}{M_i^T p_3 + p_{34}} \\ g(\Theta, M_i) &= \frac{M_i^2 p_2 + p_{24}}{M_i^T p_3 + p_{34}} \end{aligned}$$

Then the problem can be stated as follows :

Find Θ by minimizing

$$\epsilon^2 = \sum_{i=1}^{N} [f(M_i, \Theta) - c_i]^2 g[f(M_i, \Theta) - r_i]^2$$

Gradient descent can be used to solve for each parameter iteratively, i.e.,

$$\Theta^T = \Theta^{t-1} - \alpha \nabla_\Theta \epsilon^2$$

Other methods to solve for non-linear optimization include Newton method, Gauss-Newton, and Levenberg-Marquardt method. See chapter 3 of Forsyth and Ponce's book on how these methods work. Refer to appendix 4 of Hartley's book for additional iterative estimation methods. Non-linear methods all need good initial estimates to correctly converge. Implement one of the non-linear method using Matlab. It could improve the results a lot.

Linear Method v.s Non-linear Method

- Linear method is simple but less accurate and less robust
- Linear solution can be made robust via the robust method such as the RANSAC method
- Linear method does not require initial estimate
- Non-linear method is more accurate and robust but complex and require good initial estimates

The common approach in CV is two steps:

- Use a linear method to obtain initial estimates of the camera parameters.
- Refine the initial estimates using an non-linear method.

Data Normalization

Hartley introduces a data normalization technique to improve estimation accuracy. Details of the normalization technique may be found on section 3.4.4 of Hartley's book. The main idea is to subtract the image coordinates for each point by their mean to improve the numerical stability of the estimation. This normalization should precede all estimation that involves image data.

A brief discussion of this normalization procedure can also be found at page 156 of Trucco's book.

Compute Camera Parameters from P

$$P = \begin{pmatrix} s_x f \mathbf{r}_1 + c_0 \mathbf{r}_3 & s_x f t_x + c_0 t_z \\ s_y f \mathbf{r}_2 + r_0 \mathbf{r}_3 & s_y f t_y + r_0 t_z \\ \mathbf{r}_3 & t_z \end{pmatrix}$$

$$\begin{aligned}
\mathbf{r}_3 &= p_3 & t_z = p_{34} \\
c_0 &= p_1 p_3^T & r_0 = p_2 p_3^T \\
s_x f &= \sqrt{p_1 p_1^T - c_0^2} = ||p_1^T \times p_3^T||_2 \\
s_y f &= \sqrt{p_2 p_2^T - r_0^2} = ||p_2^T \times p_3^T||_2 \\
t_x &= (p_{14} - c_0 t_z)/(s_x f) \\
t_y &= (p_{24} - r_0 t_z)/(s_y f) \\
\mathbf{r}_1 &= (p_1 - c_0 r_3)/(s_x f) \\
\mathbf{r}_2 &= (p_2 - r_0 r_3)/(s_y f)
\end{aligned}$$

Compute Camera Parameters from P (cont'd)

Alternatively, we can compute W algebraically from P. Since $P = WM = W[R \;\; T] = [WR \;\; WT]$, let P_3 be the first 3×3 submatrix of P, the $P_3 = WR$.

Following the RQ decomposition algorithm (different from the QR decomposition) in

https://math.stackexchange.com/posts/1640762/revisions

we can decompose P_3 into $P_3 = AB$, where A is an upper triangle matrix and B is an orthnormal matrix.

Hence, we have $W = A$ and $R = B$.

Alternatively, we can use $C = P_3 P_3^T = WW^T$ to solve W via upper triangle Cholesky decomposition of C, i.e., $C = UU^T$ hence W=U.

Given W, T can be recovered as $T = W^{-1}P_4$, where P_4 is the last column of P.

Upper Triangle Cholesky Decomposition

Upper triangle Cholesky decomposition of a matrix A starts with a Cholesky decomposition of A, yielding

$$A = LL^T,$$

where L is a lower triangle matrix, followed by an RQ decomposition of L

$$L = RQ$$

where R is an upper triangle matrix and Q is an orthnormal matrix.

Hence,

$$C = LL^T = RQQ^T R^T = RR^T$$

Approximate solution to imposing orthonormality

Let $\hat{R}$ be the estimated rotation matrix R. It is not orthonormal. We can find another orthonormal matrix $\hat{\hat{R}}$ that is closest to $\hat{R}$ via SVD. Let $\hat{R} = UDV^T$, replacing D by the identity matrix I, yielding $\hat{\hat{R}} = UIV^T$. $\hat{\hat{R}}$ is an orthonormal matrix that is closest to $\hat{R}$.

Image Center using Vanishing Points

Let L_i, i=1,2, $\ldots$, N be parallel lines in 3D, l_i be the corresponding image lines. Due to perspective projection, lines l_i appear to meet in a point, called *vanishing point*, defined as the common intersection of all the image lines i_i. Given the orientation of the L_i lines be $N = (n_x, n_y, n_z)^T$ relative to the camera frame, then the coordinates of the vanishing point in the image frame are $(\frac{n_x}{n_z}, \frac{n_y}{n_z})$.

Let T be the triangle on the image plane defined by the three vanishing points of three mutually orthogonal sets of parallel lines in space. The image center, i.e., the principal point (c_0, r_0), is the orthocenter [a] of T.

[a]it is defined as the intersections of the three altitudes.

Estimating Scale Factor using Vanishing Points

Let (c_1, r_1) and (c_2, r_2) be the vanishing points for two sets of parallel lines that are orthogonal to each other. Assuming the image center (c_0, r_0) is known (i.e., at the image center), and the horizontal and vertical scale factors are the same, i.e., $fs_x = fs_y = fs$. We can easily prove that

$$(fs)^2 = (c_1 - c_0)(c_2 - c_0) + (r_1 - r_0)(r_2 - r_0)$$

The derivations of this equation use the facts: 1) $c_i = \frac{fsn_x^i}{n_z^i} + c_0$ and $r_i = \frac{fsn_y^i}{n_z^i} + r_0$, where i=1,2 and $N^i = (n_x^i, n_y^i, n_z^i)^T$ is the orientation of the ith line; and 2) $(N^1)^T(N^2) = 0$.

Calibration with Planar Object

Since planar points reduces the rank of A matrix to 8, we cannot uniquely solve for V using planar object as the solution to V is a linear combination of the 4 null vectors of A matrix (up to four scale factors).

But this becomes possible if we acquire two images of the planar object, producing two A matrices. With each A providing 8 independent equations, we can have a total of 16 independent equations, theoretically sufficient to solve for the intrinsic matrix W. But since the two A matrices share the same W but different extrinsic parameters, the extrinsic parameters must be eliminated from the system of linear equations to only determine W.

Calibration with Planar Object (cont'd)

For planar calibration object, we can place the object frame on the plane such that z=0. Hence, given the ith 2D/3D point, we have

$$\begin{aligned} \lambda_i \begin{pmatrix} c_i \\ r_i \\ 1 \end{pmatrix} &= W \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{T} \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \\ &= \mathbf{H} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \end{aligned} \tag{8}$$

where $\mathbf{r}_1$,$\mathbf{r}_2$, and $\mathbf{T}$ are respectively the first and second columns of the rotation matrix, and the translation vector. Given a minimum of 4 points, we can solve **H** using either linear or non-linear methods. Given **H**, we have

$$\begin{aligned} W\mathbf{r}_1 &= \mathbf{h}_1 \\ W\mathbf{r}_2 &= \mathbf{h}_2 \end{aligned} \tag{9}$$

Eq. 9 can be equivalently written as

$$\begin{aligned} \mathbf{r}_1 &= W^{-1}\mathbf{h}_1 \\ \mathbf{r}_2 &= W^{-1}\mathbf{h}_2 \end{aligned} \tag{10}$$

Using the fact that $\mathbf{r}_1$ and $\mathbf{r}_2$ are unit vectors and they are orthogonal to each other, we can derive the following

$$\begin{aligned} ||\mathbf{r}_1||_2^2 &= 1 \Rightarrow \mathbf{h}_1^T W^{-t} W^{-1} \mathbf{h}_1 = 1 \\ ||\mathbf{r}_2||_2^2 &= 1 \Rightarrow \mathbf{h}_2^T W^{-t} W^{-1} \mathbf{h}_2 = 1 \\ \mathbf{r}_1^T \mathbf{r}_2 &= 0 \Rightarrow \mathbf{h}_1^T W^{-t} W^{-1} \mathbf{h}_2 = 0 \end{aligned} \tag{11}$$

Let $K = W^{-t}W^{-1}$, Eq. 11 can be re-rewritten as

$$\begin{aligned} \mathbf{h}_1^T K \mathbf{h}_1 = 1 \\ \mathbf{h}_2^T K \mathbf{h}_2 = 1 \\ \mathbf{h}_1^T K \mathbf{h}_2 = 0 \end{aligned} \tag{12}$$

Eq. 12 provides three equations for K. We need another image to produce another three equations. In total, we have six equations to solve for six unknowns in K as K is symmetric. Given K, we can then apply upper triangle cholesky decomposition to K^{-1}, i.e., $(K^{-1} = UU^T)$to solve for W (W=U). For details see

Zhengyou Zhang's calibration method at
http://research.microsoft.com/en-us/um/people/zhang/Calib/

Camera Calibration using Lines and Conics

Besides using points, we can also perform camera calibration using correspondences between 2D/3D lines and 2D/3D conics. Furthermore, we can also extend the point-based camera calibration to estimate the lens distortion coefficient k_1. These can be topics for the final project.

Line Basics

- In general, a line equation can be expressed as $p = p_0 + \eta N$, where p is an any point on the line, p_0 is a given point on the line, N is the orientation of the line, and η is scalar that measures the distance between p and p_0. The parameters of the line include p_0 and N. If p_0 is chosen such that Op_0 is perpendicular to the line, we can reduce the number of line parameters by one for a total 4 independent parameters for a 3D line, where O is the origin. This line equation applies to both 2D and 3D line.
- Alternatively, given two points p_1 and p_2, the equation for the line that goes through the two points is $p = \alpha p_1 + (1 - \alpha)p_2$.
- A 2D line equation in row-column frame can also be expressed as $\alpha c + \beta r + \gamma = 0$, where (α, β, γ) are the line

parameters. The orientation of the line is $(\frac{\alpha}{\sqrt{\alpha^2+\beta^2}}, \frac{-\beta}{\sqrt{\alpha^2+\beta^2}})$. Alternatively, given p_1 and p_2 in image plane in homogeneous coordinates, the line between them can be computed as $l = p_1 \times p_2$, where $\times$ stands for the cross product and l is a 3×1 vector that encodes image line parameters (α, β, γ). As the norm of (α, β, γ) is 1, the the number of independent 2D line parameters is 2.

- The intersection of two lines in 2D can be computed from the cross product of their line parameters. Let $l = (\alpha, \beta, \gamma)^T$ and $l' = (\alpha', \beta', \gamma')^T$ be the parameters of two 2D lines, their intersection point p can be computed as $p = l \times l'$. The intersection point coordinates can be computed by dividing the first two elements of p by the third element.

- An image line $\alpha c + \beta r + \gamma = 0$ in row-column frame, together with the camera center, forms a plane (backprojection plane) and the equation of the plane in camera frame is $[W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}]^T \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} = 0$, where $n_c = W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$ is the normal of the back-projection plane relative to the camera frame. The intersection of this plane with the image plane $z_c = f$ gives the equation of the image plane in camera frame.

Camera Calibration using Lines

Let $\mathrm{M} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ be a 3D point w.r.t the object frame and $\mathrm{m} = \begin{pmatrix} c \\ r \end{pmatrix}$ be the corresponding image point.

Let the3D line be parameterized as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + \lambda \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix} = \mathrm{M}_0 + \eta \mathbf{N}$$

where $\mathbf{N} = (N_x, N_y, N_z)^T$ is the orientation of the 3D line wrt the object frame, M_0 is a known 3D point on the line and η is a scalar.

The corresponding image line in the row - column frame be parameterized as

$$\alpha c + \beta r + \gamma = 0 \Rightarrow \begin{pmatrix} c \\ r \\ 1 \end{pmatrix}^t \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}$$

$$\begin{pmatrix} c \\ r \\ 1 \end{pmatrix}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = (W \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix})^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}^T W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix}^T n_c$$

where $n_c \dfrac{W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix}}{\| W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} \|_2}$ is the normal of back - projection plane wrt the camera frame.

1) $m_0 = \begin{pmatrix} c_0 \\ r_0 \end{pmatrix}$, the corresponding image point for M_0, is on the 2D image line

From $\lambda \begin{pmatrix} c_0 \\ r_0 \\ 1 \end{pmatrix} = \mathbf{P} \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}$, we have

$$\begin{pmatrix} c_0 \\ r_0 \\ 1 \end{pmatrix}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \Rightarrow \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}^T \mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \qquad (1)$$

2) N is orthogonal to n_c, we have

$$(R\mathbf{N})^T n_c = (R\mathbf{N})^T W^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \qquad (2)$$

Alternatively, any 3D point on the line can be expressed as

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} + \eta \begin{pmatrix} N_x \\ N_y \\ N_z \end{pmatrix}$$

its image projection as

$$\lambda \begin{pmatrix} c \\ r \\ 1 \end{pmatrix} = \mathbf{P} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \mathbf{P}[\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix} + \eta \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix}]$$

It must be on the image line and hence

$$\lambda \begin{pmatrix} c \\ r \\ 1 \end{pmatrix}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \Rightarrow [\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}^T + \eta \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix}^T]\mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}^T \mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} + \eta \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix} \mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \Rightarrow$$

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix}^T \mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \qquad (3)$$

$$\begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix} \mathbf{P}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \Rightarrow \begin{pmatrix} N_x \\ N_y \\ N_z \\ 0 \end{pmatrix} [\mathbf{R}, \mathbf{T}]^T \mathbf{W}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \Rightarrow (\mathbf{RN})^T \mathbf{W}^T \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = 0 \qquad (4)$$

Eq (3) and (4) are the same as Eqs. (1) and (2)

Eqs (1) and (2) (and Eqs. 3 and 4) provide a total of 2 equations for **P**. A minmum of 6 pairs of 2D/3D lines are needed to solve **P** linearly. We can impose one orthnormal constraint $\| p3 \|_2 = 1$ to still have a linear solution.

Why does a pair of 2D/3D line provide the same number of equations as a pair of 2D/3D points ? This is because a pair of 2D/3D line only has 6 independent parameters (2 for 2D line and 4 for 3D line), 1 parameter more than a pair of 2D/3D point, which has 5 independent parameters (2 for 2D point and 3 for 3D point). The one extra parameter is not enough to produce another equation on the projection matrix.

Camera Calibration under Weak Perspective Projection

For weak perspective projection in relative coordinate system, we have

$$\begin{pmatrix} c' \\ r' \end{pmatrix} = M_{2\times 3} \begin{pmatrix} \bar{x}' \\ y' \\ z' \end{pmatrix}$$

where the relative coordinates (c'_i, r'_i) and (x'_i, y'_i, z'_i) are obtained by

$$
\begin{aligned}
c'_i &= c_i - \bar{c} \\
r'_i &= r_i - \bar{r} \\
x'_i &= x_i - \bar{x} \\
y'_i &= y_i - \bar{y} \\
z'_i &= z_i - \bar{z}
\end{aligned}
$$

where $(\bar{c}, \bar{y})$ and $(\bar{x}, \bar{y}, \bar{z})$ are respectively 2D and 3D centroid of a set of points. Given a minimum of 3 pairs of (c'_i, r'_i) and (x'_i, y'_i, z'_i), an unique solution to M can be found linearly. And, more importantly, these points can be coplanar points.

Camera Calibration with Weak Perspective Projection

Given M and the parameterization for M introduced in the previous chapter, $M = \begin{pmatrix} \frac{f s_x \mathbf{r}_1}{\bar{z}_c} \\ \frac{f s_y \mathbf{r}_2}{\bar{z}_c} \end{pmatrix}$, we have

$$\begin{aligned} \frac{f s_x}{\bar{z}_c} &= ||m_1||_2 \\ \frac{f s_y}{\bar{z}_c} &= ||m_2||_2 \end{aligned}$$

where m_1 and m_2 are the first row and the second row of the M matrix.

Then,

$$\begin{aligned}\mathbf{r}_1 &= \frac{m_1}{||m_1||_2}\\ \mathbf{r}_2 &= \frac{m_2}{||m_2||_2}\\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2\end{aligned}$$

With weak perspective projection, we need a minimum of 3 points. But we can only solve the above parameters, i.e., the $\frac{fs_x}{\bar{z}_c}$, $\frac{fs_y}{\bar{z}_c}$, and the orientation of the object frame relative to the camera frame.

Camera Calibration with Lens Distortion

We present an approach for simultaneous linear estimation of the camera parameters and the lens distortion, based on the *division* lens distortion model proposed by Fitzgibbon [a]. According to the *divisional* model, we have

$$\begin{pmatrix} \hat{c} - c_0 \\ \hat{r} - r_0 \end{pmatrix} = (1 + ks^2) \begin{pmatrix} c - c_0 \\ r - r_0 \end{pmatrix}$$

where $s^2 = (\hat{c} - c_0)^2 + (\hat{r} - r_0)^2$. This is an approximation to the

[a]the paper appears in CVPR 01, pages 125-132

camera true distortion model. Hence,

$$\frac{\lambda}{1+ks^2}\begin{pmatrix}\hat{c}-c_0\\ \hat{r}-r_0\\ 0\end{pmatrix} = P\begin{pmatrix}x\\ y\\ z\\ 1\end{pmatrix} - \lambda\begin{pmatrix}c_0\\ r_0\\ 1\end{pmatrix}$$

After solving for $\lambda = p_3(x\ y\ z\ 1)^T$ and with some algebraic simplifications yield

$$(D_1 + kD_2)V = 0$$

where V=$(p_1\ p_2\ p_3)^T$, p_i is the ith row of matrix P, and

$$D_1 = \begin{pmatrix} x & y & z & 1 & 0 & 0 & 0 & 0 & -\hat{c}x & -\hat{c}y & -\hat{c}z & -c_0 \\ 0 & 0 & 0 & 0 & x & y & z & 1 & -\hat{r}x & -\hat{r}y & -\hat{r}z & -r_0 \end{pmatrix}$$

$$D_2 = \left(\begin{matrix} xs^2 & ys^2 & zs^2 & s^2 & 0 & 0 & 0 & 0 & -c_0xs^2 & -c_0ys^2 & -c_0 \\ 0 & 0 & 0 & 0 & xs^2 & ys^2 & zs^2 & s^2 & -r_0xs^2 & -r_0ys^2 & -r_0 \end{matrix} \right.$$

The above equation can be solved as a polynomial eigen value problem. MATLAB function *polyeig* can be used to obtain the solution for both k and V. To use *ployeig* function, the matrices on the left side of above equations must be square matrices. To achieve this, multiple both sides of the above equation by $(D_1 + kD_2)^T$ yields the following, which can be solved for using *polyeig*

$$(D_1^T D_1 + k(D_1^T D_2 + D_2^T D_1) + k^2 D_2^T D_2)V = 0^{\mathrm{b}}$$

The solution, however, assumes the knowledge of the image center. We can fix it as the center of the image. Study shows that the precise location of the distortion center does not strongly affect the correction (see Ref. 9 of Fitz's paper).

Alternatively, we can perform alternation, i.e., assume image center as the principal point, then use the above to compute k and the internal camera parameters. Then, substitute the computed center back to recompute k and the camera parameters. This process repeats until it converges, i.e., when the change in the estimated parameters is small.

The procedure can be summarized as follows:

[b]when k is small, D_1 is close to the A matrix.

1. Assume principal center is at image center. This allows to compute $\hat{c} - c_0, \hat{r} - r_0$, and $s^2 = (\hat{c} - c_0)^2 + (\hat{r} - r_0)^2$ for each point.
2. Use Polyeig to solve for k and matrix P
3. Obtain the intrinsic camera parameters from P
4. Repeat steps 2) and 3) with the new principal center until the change in the computed intrinsic parameters is less than a pre-defined threshold.

Degeneracy with Camera Calibration

Degeneracy occurs when the solution to the projection matrix is not unique due to special spatial point configurations.

see sections 3.2.3 and 3.3.3 of Forsyth's book.

Camera Self Calibration

Self camera calibration refers to determining the interior camera parameters of a camera by using only image data obtained from different view directions or different view points.

Either camera or the object must move to acquire different images.

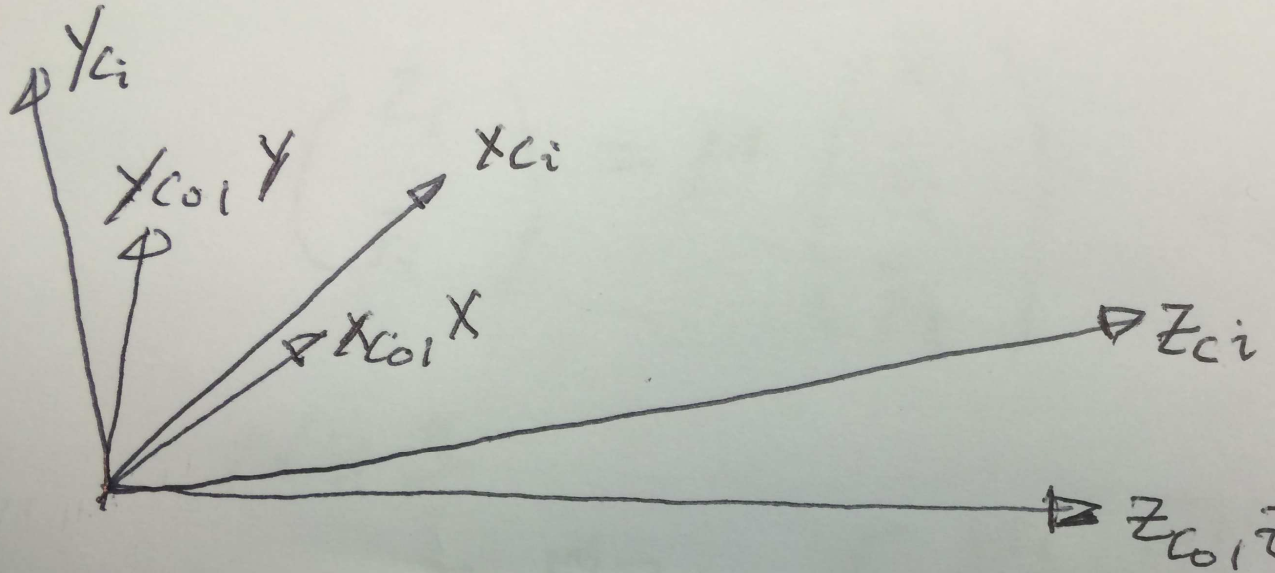
Methods for Camera Self-Calibration

- General camera movement (involving both rotation and translation)
- **Only rotational movement** (same view point but different view directions)
- Only translational movement (different view points but same view direction)

Camera Self-Calibration With Only Rotation

As we do not have 3D information, we can locate the object frame anywhere to simplify the subsequent calculations.

Let's assume that we select the initial camera frame as the reference frame and the object frame **coincide** with the initial camera frame. Let the image generated by the initial camera frame be represented with subscript 0 indexed by i. Let j index to the jth image point.



$X_{co}-Y_{co}-Z_{co}$ — originial camera frame before rotation

$X.-Y-Z$ — be the object frame.

Assume $X-Y-Z$ and $X_{co}-Y_{co}-Z_{co}$ coincide.

$$\lambda_{0,j} \begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} = \mathbf{W M_0} \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W}[\mathbf{I} \;\; \mathbf{0}] \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W} \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} \tag{13}$$

If we rotate the camera frame from the reference by a rotation matrix R_i, we have

$$\lambda_{i,j} \begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \mathbf{W M_i} \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W}[\mathbf{R_i} \; \mathbf{0}] \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W R_i} \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} \tag{14}$$

Denote $\lambda'_{i,j} = \frac{\lambda_{0,j}}{\lambda_{i,j}}$, substituting 13 to 14 to remove $(X_j, Y_j, Z_j)^T$ yields

$$\begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \lambda'_{i,j} \mathbf{W R_i W}^{-1} \begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} \tag{15}$$

Let $\mathbf{B_i} = \mathbf{W R_i W^{-1}} = \begin{pmatrix} B_{i_{11}} & B_{i_{12}} & B_{i_{13}} \\ B_{i_{21}} & B_{i_{22}} & B_{i_{23}} \\ B_{i_{31}} & B_{i_{32}} & B_{i_{33}} \end{pmatrix}$, we have

$$\begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \lambda_{i,j} \begin{pmatrix} B_{i_{11}} & B_{i_{12}} & B_{i_{13}} \\ B_{i_{21}} & B_{i_{22}} & B_{i_{23}} \\ B_{i_{31}} & B_{i_{32}} & B_{i_{33}} \end{pmatrix} \begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} \tag{16}$$

This leads to three equations

$$\begin{aligned} c_{i,j} &= \lambda_{i,j}(B_{i_{11}}c_{0,j} + B_{i_{12}}r_{0,j} + B_{i_{13}}) \\ r_{i,j} &= \lambda_{i,j}(B_{i_{21}}c_{0,j} + B_{i_{22}}r_{0,j} + B_{i_{23}}) \\ 1 &= \lambda_{i,j}(B_{i_{31}}c_{0,j} + B_{i_{32}}r_{0,j} + B_{i_{33}}) \end{aligned} \tag{17}$$

Since $\lambda_{i,j} = 1/(B_{i_{31}}c_0 + B_{i_{32}}r_0 + B_{i_{33}})$, substituting $\lambda_{i,j}$ to the above equations yields

$$\begin{aligned} c_i(B_{i_{31}}c_0 + B_{i_{32}}r_0 + B_{i_{33}}) &= (B_{i_{11}}c_0 + B_{i_{12}}r_0 + B_{i_{13}}) \\ r_i(B_{i_{31}}c_0 + B_{i_{32}}r_0 + B_{i_{33}}) &= (B_{i_{21}}c_0 + B_{i_{22}}r_0 + B_{i_{23}}) \end{aligned}$$

Given N (j=1,2, ..,N) points, we can setup a system of linear equations, through which we can solve B as the null vector of the measurement matrix up to a scale factor.

Alternatively, we can divide the both sides of the above 2 equations by $B_{i_{33}}$ and they can be rewritten in matrix format

$$\begin{pmatrix} -c_0 & -r_0 & -1 & 0 & 0 & 0 & c_i c_0 & c_i r_0 \\ 0 & 0 & 0 & -c_0 & -r_0 & -1 & r_i c_0 & r_i r_0 \end{pmatrix} \mathbf{b_i} = \begin{pmatrix} -c_i \\ -r_i \end{pmatrix} \tag{18}$$

where $\mathbf{b_i} = (B_{i_{11}}\ B_{i_{12}}\ B_{i_{13}}\ B_{i_{21}}\ B_{i_{22}}\ B_{i_{23}}\ B_{i_{31}}\ B_{i_{32}})^T / B_{i_{33}}$. If we know at least 4 points in two images (such as $i = 0, 1$), we can solve for $\mathbf{b_i}$ up to a scale factor. The scale factor can be solved using the fact that the determinant of $\mathbf{W R_i W^{-1}}$ is unit.

If R_i is known, then we can solve for W using the equation $B_i W = W R_i$. In this case, one rotation, i.e., a total of two images, is enough to solve for W.

To solve for W with a unknown R_i. From $\mathbf{B_i} = \mathbf{W R_i W^{-1}}$, we have

$$\mathbf{R_i} = \mathbf{W^{-1} B_i W} \qquad \mathbf{R_i^{-T}} = \mathbf{W^T B_i^{-1} W^{-T}}$$

Since $\mathbf{R} = \mathbf{R^{-T}}$, therefore we have

$$(\mathbf{WW^T})\mathbf{B_i^{-T}} = \mathbf{B_i}(\mathbf{WW^T}) \tag{19}$$

Assume $\mathbf{C} = \mathbf{WW^T}$, we have

$$\begin{aligned} \mathbf{C} &= \begin{pmatrix} s_x f & 0 & c_0 \\ 0 & s_y f & r_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_x f & 0 & 0 \\ 0 & s_y f & 0 \\ c_0 & r_0 & 1 \end{pmatrix} \\ &= \begin{pmatrix} s_x^2 f^2 + c_0^2 & c_0 r_0 & c_0 \\ c_0 r_0 & s_y^2 f^2 + r_0^2 & r_0 \\ c_0 & r_0 & 1 \end{pmatrix} \end{aligned} \tag{20}$$

equation 19 can be rewritten

$$\mathbf{B_i C B_i^T} - \mathbf{C} = 0 \tag{21}$$

Since **C** is symmetric, Eq. 21 provides only six equations. To solve for C, it is necessary to use two B_i, i.e., two rotations, which leads to three images. Given two or more B_i, C can be solved using equation 21 up to a scale factor. The scale factor can subsequently be resolved using the fact that the last element of W is 1.

Given C, from equation $C = WW^T$, we can obtain W directly from upper triangle Choleski factorization.

Section 11.1.3 of [5] introduced a method to recover the focus length from a single rotation, assuming the principal point is located in the image center.

Camera Self-Calibration With Only Translation

Like for the previous case, the camera frame coincides with the object frame. We then translate the camera frame by $\mathbf{T}_i$. For the image points in the reference frame and the newly translated frame, we have

$$\lambda_{0,j} \begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} = \mathbf{W}[\mathbf{I}\ \mathbf{T_0}] \begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W} \begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} \tag{22}$$

$$\lambda_{i,j}\begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \mathbf{W}[\mathbf{I}\ \mathbf{T_i}]\begin{pmatrix} X_j \\ Y_j \\ Z_j \\ 1 \end{pmatrix} = \mathbf{W}\begin{pmatrix} X_j \\ Y_j \\ Z_j \end{pmatrix} + \mathbf{WT_i} \quad (23)$$

From the above equations, we have

$$\lambda_{i,j}\begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \lambda_{0,j}\begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} + \mathbf{WT_i} \quad (24)$$

Equation 24 can be rewritten

$$\lambda_{i,j}\begin{pmatrix} c_{i,j} \\ r_{i,j} \\ 1 \end{pmatrix} = \lambda_{0,j}\begin{pmatrix} c_{0,j} \\ r_{0,j} \\ 1 \end{pmatrix} + \begin{pmatrix} s_x f & 0 & c_0 \\ 0 & s_y f & r_0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} t_{i_x} \\ t_{i_y} \\ t_{i_z} \end{pmatrix} \tag{25}$$

then, we get three equations, assuming T is known [a]. Note the three equations are not independent. In fact, each pair of points offers only two independent equations for two unknowns $\lambda_{i,j}$ and $\lambda_{0,j}$, plus the unknown W.

$$\begin{aligned} \lambda_{i,j}c_{i,j} &= \lambda_{0,j}c_{0,j} + (s_x f t_{i_x} + c_0 t_{i_z}) \\ \lambda_{i,j}r_{i,j} &= \lambda_{0,j}r_{0,j} + (s_y f t_{i_y} + r_0 t_{i_z}) \\ \lambda_{i,j} &= \lambda_{0,j} + t_{i_z} \end{aligned} \tag{26}$$

[a] there is no linear solution if T is unknown

Substituting $\lambda_{i,j} = \lambda_{0,j} + t_{i_z}$ to the above two equations yields

$$\begin{aligned} \lambda_{0,j} c_{i,j} + t_{i_z} c_{i,j} &= \lambda_{0,j} c_{0,j} + s_x f t_{i_x} + c_0 t_{i_z} \\ \lambda_{0,j} r_{i,j} + t_{i_z} r_{i,j} &= \lambda_{0,j} r_{0,j} + s_y f t_{i_y} + r_0 t_{i_z} \end{aligned} \tag{27}$$

Combining the two equations above to remove $\lambda_{0,j}$ yields

$$\begin{aligned} & (r_{ij} - r_{0,j}) t_{i_x} f s_x - (c_{ij} - c_{0,j}) t_{i_y} f s_y + \\ & (r_{i,j} - r_{0,j}) t_{i_z} c_0 - (c_{i,j} - c_{0,j}) t_{i_z} r_0 \\ = \quad & (r_{ij} - r_{0,j}) t_{i_z} c_{i,j} - (c_{i,j} - c_{0,j}) t_{i_z} r_{i,j} \end{aligned}$$

where $\mathbf{W}' = (s_x f \quad s_y f \quad c_0 \quad r_0)^T$. If we know at least 4 points in three images (producted by two translations), we can solve **W'**, then we can get the solution to **W**. Note the matrix is rank deficient (rank=3) since the first and third columns as well as the second and fourth columns are different only by a scale factor given the same translation. Therefore, one translation is not enough.

Camera Self Calibration Summary

Camera self calibration can be carried out without using 3D data. It requires camera to move to produce different images. Camera motions can be

- general motion-involving both rotation and translation. Solution is unstable and less robust
- pure rotation-requires a minimum of two rotations (or three images) if rotation is unknown. If rotation is known, one rotation or two images is enough.
- pure translation-requires a minimum of two translations and they must be known to have a linear solution.
- degenerate motions may happen and they cannot be used for self calibration.

Pose Estimation

The goal of pose estimation is to estimate the relative orientation and position between the object frame and the camera frame, i.e., determining R and T or extrinsic camera parameters.

Pose estimation is an area that has many applications including HCI, robotics, photogtametry, etc..

Linear Pose Estimation

For pose estimation, it is necessary to know 3D features and their 2D image projections. They are then used to solve for R and T, depending on if W is known or not.

Case 1: assume W is known, then from the projection equation

$$\lambda W^{-1}\begin{pmatrix} c \\ r \\ 1 \end{pmatrix} = [R, T]\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Given more than 6 sets of 2D/3D points, we can solve for R and T linearly in the similar fashion to that of linear camera calibration. The solution, however, does not impose the constraint that R be orthnormal, i.e., $R^{-1} = R^T$.

We can perform a postprocessing of the estimated R to find another $\hat{R}$, that is closest to R and orthnormal. See previous pages for details.

Alternatively, we can still have a linear solution if we impose one constraint, i.e., $||r_3|| = 1$ during optimization.

Case 2: if W is unknown, we can follow the same procedure as camera calibration to first solve for P and then extract R and T from P. Alternatively, we can assume some prior knowledge about W such as the horizontal and vertical scale factors are equal, i.e., $fs_x = fs_y$ and the principal point is located in the image center. This knowledge can be imposed as constraints during the estimation.

Non-linear Pose Estimation (cont'd)

Alternatively, we can set it up as a non-linear optimization problem, with the constraint of $R^{-1} = R^T$ imposed, along with any prior knowledge on W as discussed in the last slide.

Pose Estimation Under Weak Perspective Projection

For weak perspective projection in relative coordinate system, we have

$$\begin{pmatrix} c' \\ r' \end{pmatrix} = M_{2\times 3} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}$$

Given 2D/3D relative coordinates (c'_i, r'_i) and (x'_i, y'_i, z'_i), the goal is to solve for matrix M. A minimum of 3 points are enough to uniquely solve for the matrix M.

Pose Estimation Under Weak Perspective Projection

Given M and the parameterization for M introduced in the previous chapter, we have

$$\begin{aligned} \frac{fs_x}{\bar{z}_c} &= |m_1| \\ \frac{fs_y}{\bar{z}_c} &= |m_2| \end{aligned}$$

where m_1 and m_2 are the first row and the second row of the M matrix.

Then,

$$\begin{aligned} r_1 &= \frac{m_1}{|m_1|} \\ r_2 &= \frac{m_2}{|m_2|} \\ r_3 &= r_1 \times r_2 \end{aligned}$$

Pose Estimation Under Weak Perspective Projection

Given R, assuming W is known (i.e., calibrated camera), T can be solved using

$$\lambda W^{-1} \begin{pmatrix} c \\ r \\ 1 \end{pmatrix} = R \begin{pmatrix} x \\ y \\ z \end{pmatrix} + T$$

If W is unknown, under weak projection, we have

$$\bar{z}_c \begin{pmatrix} c \\ r \\ 1 \end{pmatrix} = \begin{pmatrix} f s_x \mathbf{r}_1 & f s_x t_x + c_0 \bar{z}_c \\ f s_y \mathbf{r}_2 & f s_y t_y + r_0 \bar{z}_c \\ 0 & \bar{z}_c \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Dividing both sides by $\bar{z}_c$ yields

$$\begin{pmatrix} c \\ r \end{pmatrix} = M_{2\times 3} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

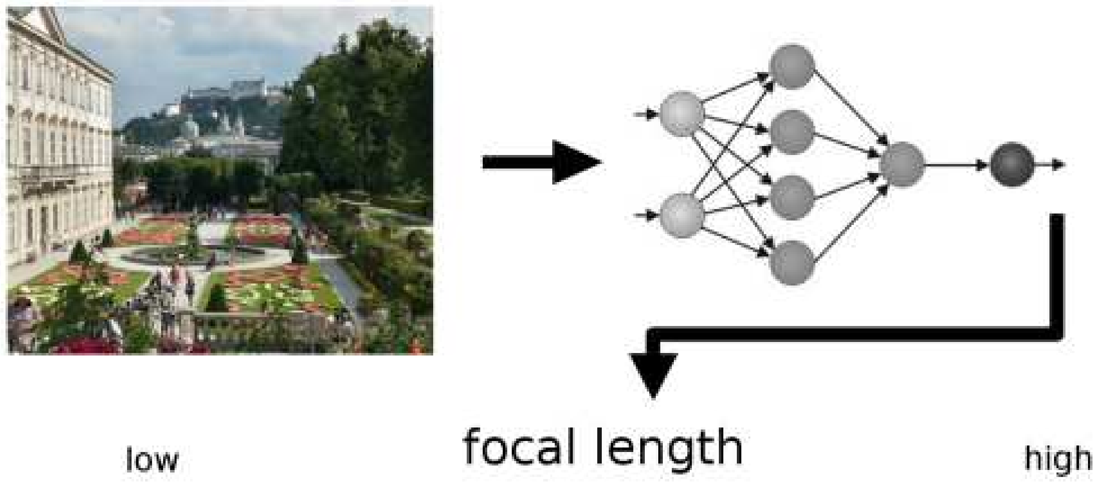
where $M_{2\times 3}$ is the first 2×3 submatrix, $v_x = \frac{p_{14}}{p_{34}} = \frac{f s_x}{\bar{z}_c} t_x + c_0$ and $v_y = \frac{p_{24}}{p_{34}} = \frac{f s_x}{\bar{z}_c} t_y + r_0$. If we assume c_0 and r_0 are in image center, we can solve for the translation t_x and t_y up to a scale factor. We can never solve for t_z.

With weak perspective projection, we need a minimum of 3 points. We can solve R but T up to a scale factor.

Learning-based Camera Calibration and Pose Estimation

- Direct regression approach

  DeepFocal and DeepCalib [6, 1] that trains a convolutional neural network that takes an image as input and outputs it focus length.



PoseNet [3] that trains a convolutional neural network to regress the 6-DOF camera pose from a single RGB image in an end-to-end manner.

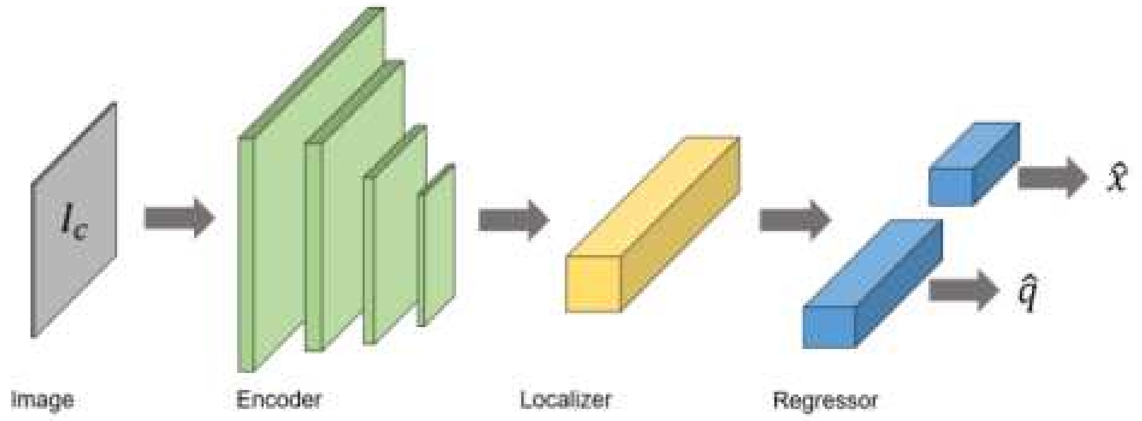
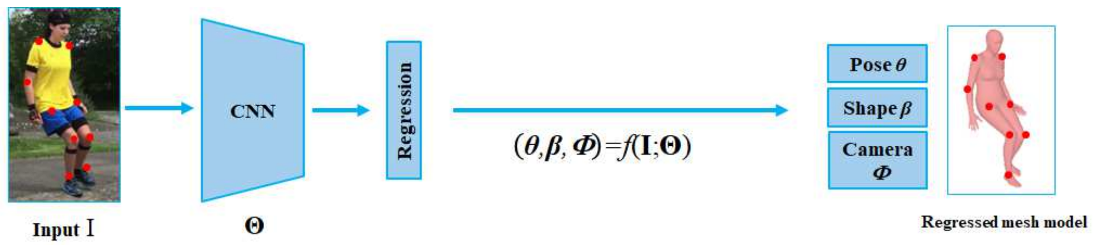


Figure 1. A schematization of the PoseNet's architecture.

- Geometric model-based approach that employs a deformable object model to perform simultaneous 3D reconstruction, camera calibration, and pose estimation.

End-to-end recovery of human shape and pose [2]



$$\Theta^* = \text{argmin}\; L_{\text{mesh}}(\theta,\beta;\hat{\theta},\hat{\beta})+L_{\text{proj}}(x;\hat{x}),\quad x \text{ is projection of body landmarks}$$

Further information on the application of deep learning for camera calibration and pose estimation may be found in this paper [4].

References

[1] Oleksandr Bogdan, Viktor Eckstein, Francois Rameau, and Jean-Charles Bazin. Deepcalib: A deep learning approach for automatic intrinsic calibration of wide field-of-view cameras. In *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*, pages 1–10, 2018.

[2] Angjoo Kanazawa, Michael J Black, David W Jacobs, and Jitendra Malik. End-to-end recovery of human shape and pose. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7122–7131, 2018.

[3] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[4] Kang Liao, Lang Nie, Shujuan Huang, Chunyu Lin, Jing Zhang, Yao Zhao, Moncef Gabbouj, and Dacheng Tao. Deep learning for camera calibration and beyond: A survey. *arXiv preprint arXiv:2303.10559*, 2023.

[5] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2021.

[6] Scott Workman, Connor Greenwell, Menghua Zhai, Ryan Baltenberger, and Nathan Jacobs. Deepfocal: A method for direct focal length estimation. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 1369–1373. IEEE, 2015.