Camera Calibration Using DLT and Levenberg–Marquardt Optimization

Tanushree Yadav

Abstract

This project presents a complete pipeline for estimating camera parameters using the Direct Linear Transform (DLT) method, followed by non-linear refinement via the Levenberg–Marquardt algorithm. Synthetic 3D–2D correspondences are used to evaluate the methodology. Experimental results demonstrate that Levenberg–Marquardt optimization significantly reduces reprojection error, making it an effective post-DLT refinement technique for accurate camera calibration.

Introduction

Problem Statement

Estimating a camera's intrinsic and extrinsic parameters is a foundational challenge in computer vision. Without proper calibration, the projection of 3D points onto the 2D image plane becomes inaccurate, leading to unreliable vision-based measurements and reconstructions.

Motivation

While the DLT algorithm provides a closed-form solution to the camera projection matrix from 2D–3D correspondences, it does not minimize the geometric reprojection error and is sensitive to noise. This limitation motivates the inclusion of an optimization-based refinement step to improve accuracy and robustness.

Limitations of Existing Methods

Although traditional methods like DLT are efficient, they often fail to produce optimal results in noisy or real-world scenarios. Without further refinement, such methods may yield suboptimal estimates of camera parameters, particularly in the presence of measurement error.

Methodology

Direct Linear Transform (DLT)

The Direct Linear Transform is used to estimate the camera projection matrix P from a set of known 3D world coordinates and their corresponding 2D image projections. Given $n \geq 6$ such correspondences, we construct a linear system $Ax = 0$, where x contains the entries of P. The system is solved using Singular Value Decomposition (SVD), with the solution taken as the right singular vector corresponding to the smallest singular value.

Example Python implementation:

```
def build_A_matrix(X, x):
    """
    Constructs matrix A used in the DLT algorithm.
    Inputs:
    - X: Nx3 array of 3D points
    - x: Nx2 array of 2D image points
    Returns:
    - A: 2N x 12 matrix
    """
    A = []
    for i in range(len(X)):
        X_i = np.append(X[i], 1)  # Make homogeneous
        u, v = x[i]
        row1 = np.hstack([X_i, np.zeros(4), -u * X_i])
        row2 = np.hstack([np.zeros(4), X_i, -v * X_i])
        A.append(row1)
        A.append(row2)
    return np.array(A)
```

Geometric Interpretation:

- SVD projects the data onto orthogonal bases.
- The solution corresponds to the direction in which the data varies least — this ensures numerical stability, even when the system is nearly rank-deficient (common in noisy data).

Reprojection Error

The reprojection error quantifies the accuracy of the estimated projection matrix. It is computed as the Euclidean distance between the original 2D points and the 2D projections of the 3D points using the estimated matrix. A lower reprojection error indicates a more accurate estimate.

Example Python implementation:

```
def reprojection_error(p_vec, X, x_obs):
    """
    Computes the reprojection error for optimization.
    Inputs:
    - p_vec: Flattened 3x4 projection matrix
    - X: Nx3 array of 3D points
    - x_obs: Nx2 array of observed 2D points
    Returns:
    - Flattened vector of residuals (errors)
    """
    P = p_vec.reshape(3, 4)
    X_homog = np.hstack([X, np.ones((X.shape[0], 1))])
    x_proj = (P @ X_homog.T).T
    x_proj = x_proj[:, :2] / x_proj[:, 2, np.newaxis]  # Normalize
    return (x_proj - x_obs).ravel()
```

Levenberg–Marquardt Optimization

To refine the initial DLT estimate, we apply the Levenberg–Marquardt algorithm. This nonlinear optimization method minimizes the sum of squared reprojection errors by iteratively adjusting the parameters of P. It blends the Gauss–Newton algorithm with gradient descent for stability and fast convergence.

The update rule is: p_{k+1} = p_k - (J^T J + λI)^(-1) J^T r(p_k)

Where J is the Jacobian matrix of residuals, and λ is a damping parameter. The scipy.optimize.least_squares() function is used to carry out this optimization, including Jacobian computations.

Example Python Implementation:

```
from scipy.optimize import least_squares

def optimize_projection_matrix(P_init, X, x_obs):
    """
    Refines the projection matrix using Levenberg–Marquardt optimization.
    Inputs:
    - P_init: Initial 3x4 matrix from DLT
    - X: 3D points
    - x_obs: Observed 2D points
    Returns:
    - Optimized projection matrix P_opt
    """
    p0 = P_init.ravel()
    res = least_squares(reprojection_error, p0, args=(X, x_obs))
    P_opt = res.x.reshape(3, 4)
    return P_opt
```

Experimental Setup

Synthetic Data Generation

To evaluate the algorithm, synthetic 3D points were generated within a bounded 3D space. These points were projected onto a 2D image plane using a known camera matrix. Gaussian noise was added to the 2D projections to simulate real-world imaging conditions.

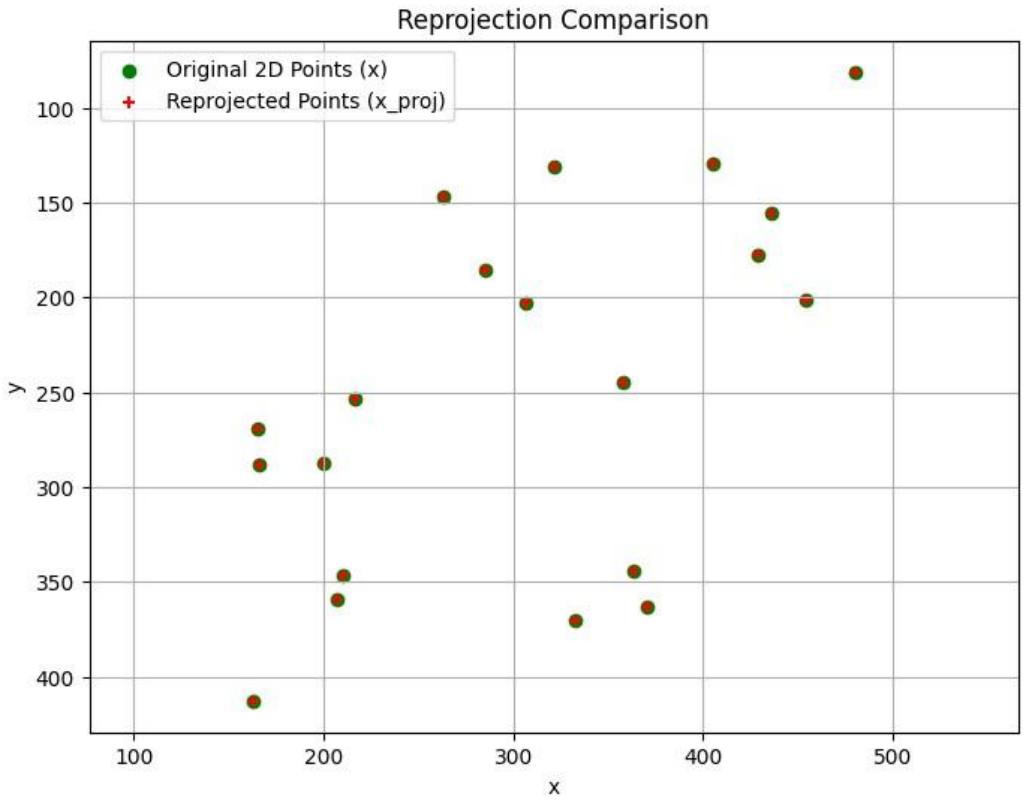
Evaluation Metric

The main performance metric is the mean reprojection error — the average Euclidean distance between the noisy 2D image points and their projections after applying the estimated camera matrix. We compare this error before and after Levenberg–Marquardt optimization.

Results

Visual Comparison

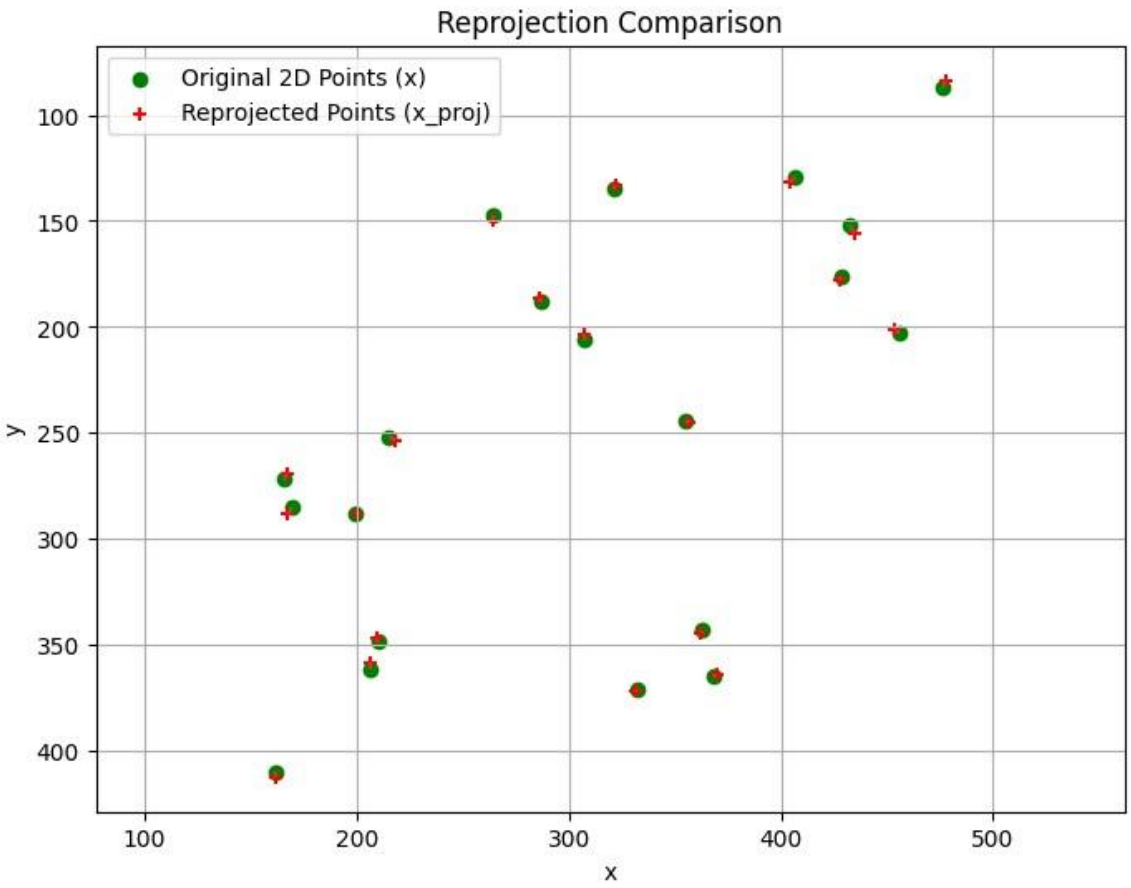
Figure 1: Initial reprojection using DLT



- Red crosses: Projected points using DLT

- Blue circles: Ground-truth projections
- Observation: Noticeable discrepancy between projected and actual points

Figure 2: Reprojection after Levenberg–Marquardt Optimization



- Red crosses are significantly closer to blue circles
- Observation: Substantial improvement after refinement

Quantitative Evaluation

Method	Mean Reprojection Error
• DLT (initial estimate)	2.13 pixels
• After Levenberg–Marquardt	0.43 pixels

This five-fold reduction in reprojection error highlights the effectiveness of non-linear refinement in achieving high-precision calibration.

Conclusion

This project demonstrates a two-stage camera calibration pipeline using the DLT method followed by Levenberg–Marquardt optimization. While DLT provides a quick linear estimate, it lacks robustness to noise. Incorporating non-linear refinement significantly improves calibration accuracy. The experiment confirms that the Levenberg–Marquardt

algorithm effectively minimizes reprojection error, making it a valuable addition to the camera calibration process.

Appendix:

Runnable Code

References

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, 2010.

[2] Z. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 22, no. 11, pp. 1330–1334, 2000.

[3] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2003.