Eigendecomposition-free Training of Deep Networks with Zero Eigenvalue-based Losses

Zheng Dang[1,2][0000−0003−2028−6096], Kwang Moo Yi[3][0000−0001−9036−3822], Yinlin Hu[4][0000−0003−2614−5200], Fei Wang[1,2][0000−0003−3462−8472], Pascal Fua[4][0000−0002−6702−9970], and Mathieu Salzmann[4][0000−0002−8347−8637]

[1] National Engineering Laboratory for Visual Information Processing and Application, Xi'an Jiaotong University, 99 Yanxiang Road, Xi'an, Shaanxi 710054, China
[2] School of Electronic and Information Engineering, Xi'an Jiaotong University, 28 West Xianning Road, Xi'an, Shaanxi 710049, China
dangzheng713@stu.xjtu.edu.cn, wfx@mail.xjtu.edu.cn
[3] Visual Computing Group, University of Victoria, Canada
kyi@uvic.ca
[4] CVLab, EPFL, Switzerland
{yinlin.hu, pascal.fua, mathieu.salzmann}@epfl.ch

Abstract. Many classical Computer Vision problems, such as essential matrix computation and pose estimation from 3D to 2D correspondences, can be solved by finding the eigenvector corresponding to the smallest, or zero, eigenvalue of a matrix representing a linear system. Incorporating this in deep learning frameworks would allow us to explicitly encode known notions of geometry, instead of having the network implicitly learn them from data. However, performing eigendecomposition within a network requires the ability to differentiate this operation. While theoretically doable, this introduces numerical instability in the optimization process in practice.

In this paper, we introduce an eigendecomposition-free approach to training a deep network whose loss depends on the eigenvector corresponding to a zero eigenvalue of a matrix predicted by the network. We demonstrate on several tasks, including keypoint matching and 3D pose estimation, that our approach is much more robust than explicit differentiation of the eigendecomposition. It has better convergence properties and yields state-of-the-art results on both tasks.

Keywords: End-to-end learning, eigendecomposition, singular value decomposition, geometric vision.

1 Introduction

In traditional Computer Vision, many tasks can be solved by finding the singular- or eigen-vector corresponding to the smallest, often zero, singular- or eigen-value of the matrix encoding a linear system. Examples include estimating essential

matrices or homographies from matched keypoints and computing pose from 3D to 2D correspondences.

In the era of Deep Learning, there is growing interest in embedding these methods within a deep architecture to allow end-to-end training. For example, it has recently been shown that such an approach can be used to train networks to detect and match keypoints in image pairs while accounting for the global consistency of the correspondences [37]. More generally, this approach would allow us to explicitly encode notions of geometry within deep networks, thus sparing the network the need to re-learn what has been known for decades and making it possible to learn from smaller amounts of training data.

One way to implement this approach is to design a network whose output defines a matrix and train it so that the smallest singluar- or eigen-vector of the matrices it produces are as close as possible to ground-truth ones. This is the strategy used in [37] to simultaneously establish correspondences and compute the corresponding Essential Matrix: The network's outputs are weights discriminating inlier correspondences from outliers and are used to assemble an auxiliary matrix whose smallest eigenvector is the sought-for Essential Matrix.

The main obstacle to implementing this approach is that it requires being able to differentiate the singular value decomposition (SVD) or the eigendecomposition (ED) in a stable manner to train the network, a non-trivial problem that has already received considerable attention [26,9,16]. As a result, these decompositions are already part of standard Deep Learning frameworks, such as TensorFlow [1] or PyTorch [27]. However, they ignore two key practical issues. First, when optimizing with respect to the matrix itself or with respect to parameters defining it, the vector corresponding to the smallest singular value or eigenvalue may switch abruptly as the relative magnitudes of these values change, which is essentially non-differentiable. This is illustrated in the example of Fig. 1, discussed in detail in Section 2. Second, computing the gradient requires dividing by the difference between two singular values or eigenvalues, which could be zero. While a solution to the latter was proposed in [26], the former is unavoidable.

In this paper, we therefore introduce an approach to training a deep network whose loss depends on the eigenvector corresponding to a zero eigenvalue of a matrix $\mathbf{M}$, which is either the output of the network or a function of it, *without* explicitly performing an SVD or ED. Our loss is fully differentiable, does *not* suffer from the instabilities the above-mentioned problems can cause, and can be naturally incorporated in a deep learning architecture. In practice, because image measurements are never perfect, the eigenvalue is never strictly zero. This, however, does not affect the computation either, which makes our approach robust to noise.

To demonstrate this in a Deep Learning context, we evaluate our approach on the tasks of training a network to find globally-consistent keypoint correspondences using the essential matrix and training another to remove outliers for pose estimation when solving the Perspective-n-Point (PnP) problem. In both cases, our approach delivers state-of-the-art results, whereas using the standard

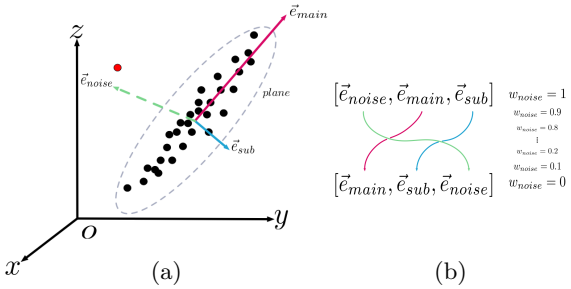


Fig. 1: **Eigenvector switching. (a)** 3D points lying on a plane in black and distant outlier in red. **(b)** When the weights assigned to all the points are one, the eigenvector corresponding to the smallest eigenvalue is $\mathbf{e}_{sub}$, the vector shown in blue in (a), and on the right in the top portion of (b), where we sort the eigenvectors by decreasing eigenvalue. As the optimization progresses and the weight assigned to the outlier decreases, the eigenvector corresponding to the smallest eigenvalue switches to $\mathbf{e}_{noise}$, the vector shown in green in (a), which introduces a sharp change in the gradient values.

implementation of singular- and eigen-value decomposition provided in TensorFlow results in either the learning procedure not converging or in significantly worse performance.

2 Motivation

To illustrate the problems associated with differentiating eigenvectors and eigenvalues, consider the outlier rejection toy example depicted by Fig. 1. The inputs are 3D points lying on a plane and drawn in black, and an outlier 3D point shown in red, which we assume to be very far from the plane. Suppose we want to assign a binary weight to each point (1 for inliers, 0 for outliers) such that the eigenvector corresponding to the smallest eigenvalue of the weighted covariance matrix is close to the ground-truth one in the least-square sense. When the weight assigned to the outlier is 0, it would be $\mathbf{e}_{noise}$, which is also the normal to the plane and is shown in green. However, if at some point during optimization, typically at initialization, we assign the weight 1 to the outlier, $\mathbf{e}_{noise}$ will correspond to the largest eigenvalue instead of the smallest, and the eigenvector corresponding to the smallest eigenvalue will be the vector $\mathbf{e}_{sub}$ shown in blue, which is perpendicular to $\mathbf{e}_{noise}$. As a result, if we initially set all weights to 1 and optimize them so that the smallest eigenvector approaches the plane normal, the gradient values will depend on the coordinates of $\mathbf{e}_{sub}$. At one point during the optimization, if everything goes well, the weight assigned to the outlier will become small enough so that the smallest eigenvector switches from being $\mathbf{e}_{sub}$ to being $\mathbf{e}_{noise}$, which introduces a large jump in the gradient vector whose values will now depend on the coordinates of $\mathbf{e}_{noise}$ instead of $\mathbf{e}_{sub}$.

In this simple case, this kind of instability does not preclude eventual convergence. However, in more complex situations, we found that it does, as evidenced by our experiments. This problem was already noted in [37] in the context of learning keypoint correspondences. To circumvent this issue, the algorithm in [37] had to first rely on a classification loss to determine the potential inlier correspondences before incorporating the loss based on the essential matrix to impose geometric constraints, which requires eigendecomposition. This ensured that the network weights were already good enough to prevent eigenvector switching when starting to minimize the geometry-based loss.

3 Related Work

In recent years, the need to integrate geometric methods and mathematical tools into Deep Learning frameworks has led to the reformulation of a number of them in network terms. For example, [17] considers spatial transformations of image regions with CNNs. The set of such transformations is extended in [10]. In a different context, [24] derives a differentiation of the Cholesky decomposition that could be integrated in Deep Learning frameworks.

Unfortunately, the set of geometric Computer Vision problems that these methods can handle remains relatively limited. In particular, there is no widely accepted deep-learning way to solve the many geometric problems that reduce to finding least-square solution of linear systems. In this work, we consider two such problems: Computing the essential matrix from keypoint correspondences in an image pair and estimating the 3D pose of an object from 3D-to-2D correspondences, both of which we briefly discuss below.

Estimating the Essential matrix from correspondences. The eigenvalue-based solution to this problem has been known for decades [23,12,11] and remains the standard way to compute Essential matrices [25]. The real focus of research in this area has been to establish reliable keypoint correspondences and to eliminate outliers. In this context, variations of RANSAC [7], such as MLESAC [33] and Least median of squared (LMeds) [29], and very recently GMS [2], have become popular. For a comprehensive study of such methods, we refer the interested reader to [28]. With the emergence of Deep Learning, there has been a trend towards moving away from this decades-old knowledge and apply instead a black-box approach where a Deep Network is trained to directly estimate the rotation and translation matrices [38,34] without *a priori* geometrical knowledge. The very recent work of [37] attempts to reconcile these two opposing trends by embedding the geometric constraints into a Deep Net and has demonstrated superior performance for this task when the correspondences are hard to establish.

Estimating 3D pose from 3D-to-2D correspondences. This is known as the Perspective-n-Point (PnP) problem. It has also been investigated for decades and is also amenable to an eigendecomposition-based solution [11], many variations of which have been proposed over the years [21,19,40,6]. DSAC [3] is

the only approach we know of that integrates the PnP solver into a Deep Network. As explicitly differentiating through the PnP solver is not optimization friendly, the authors apply the log trick used in the reinforcement learning literature. This amounts to using a numerical approximation of the derivative from random samples, which is not ideal, given that an analytical alternative exists. Moreover, DSAC only works for grid configurations and known scenes. By contrast, the method we propose in this work has an analytical form, with no need for stochastic sampling.

Differentiating the eigen- and singular value decomposition Whether computing the essential matrix, estimating 3D pose, or solving any other least-squares problem, incorporating an eigendecomposition-solver into a deep network requires differentiating the eigendecomposition. Expressions for such derivatives have been given in [26,9] and reformulated in terms that are compatible with back-propagation in [16]. Specifically, as shown in [16], for a matrix $\mathbf{M}$ written as $\mathbf{M} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T$, the variations of the eigenvectors $\mathbf{U}$ with respect to the matrix, used to compute derivatives, are

$$d\mathbf{U} = 2\mathbf{U}\left(\mathbf{K} \odot (\mathbf{U}^T d\mathbf{M}\mathbf{U})_{sym}\right) \ , \tag{1}$$

where $\mathbf{S}_{sym} = \frac{1}{2}(\mathbf{S}^T + \mathbf{S})$, and

$$\mathbf{K}_{ij} = \begin{cases} \frac{1}{\sigma_i - \sigma_j}, & i \neq j \\ 0, & i = j \end{cases} \ . \tag{2}$$

As can be seen from Eq. 2, if two eigenvalues are equal, that is, $\sigma_i = \sigma_j$, the denominator becomes 0, thus creating numerical instabilities. The same can be said about singular value decomposition.

A solution to this was proposed in [26], and singular- and eigen-value decomposition have been used within deep networks for problems where all the singular values are used and their order is irrelevant [14,15]. In the context of spectral clustering, the approach of [20] also proposed a solution that eliminates the need for explicit eigendecomposition. This solution, however, was dedicated to the scenario where one seeks to use all non-zero eigenvalues, assuming a matrix of constant rank.

Here, by contrast, we tackle problems where what matters is a single eigen- or singular-value. In this case, the order of the eigenvalues is important. However, this order can change during training, which results in a non-differentiable switch from one eigenvector to another, as in the toy example of Section 2. In turn, this leads to numerical instabilities, which can prevent convergence. In [37], this problem is finessed by first training the network using a classification loss that does not depend on eigenvectors. Only once a sufficiently good solution is found, that is, a solution close enough to the correct one for vector switching not to happen anymore, is the loss term that depends on the eigenvector associated to the smallest eigenvalue turned on. As we will show later, we can achieve state-of-the-art results without the need for such a heuristic, by deriving a more robust, eigendecomposition-free loss function.

4 Our Approach

We introduce an approach that enables us to work with eigenvectors corresponding to zero eigenvalues within an end-to-end learning formalism, while being subject to neither the gradient instabilities due to vector switching discussed in Section 2 nor to difficulties caused by repeated eigenvalues. To this end, we derive a loss function that directly operates on the matrix whose eigen- or singular-vectors we are interested in but without explicitly performing an SVD or ED.

Below, we first discuss the generic scenario in which the matrix of interest directly is the output of the network. We then consider the slightly more involved case where the network predicts weights that themselves define the matrix, which corresponds to our application scenarios. Note that, while we discuss our approach in the context of Deep Learning, it is applicable to *any* optimization framework where one seeks to optimize a loss function based on the smallest eigenvector of a matrix with respect to the parameters that defining this matrix.

4.1 Generic Scenario

Given an input measurement $\mathbf{x}$, let us denote by $f_\theta(\mathbf{x})$ the output of a deep network with parameters θ. Here, we consider the case where the output of the network is a matrix, which we write as $\mathbf{A}_\theta = f_\theta(\mathbf{x})$. Our goal is to tackle problems where the loss function of the network depends on the smallest eigenvector $\mathbf{e}_\theta$ of $\mathbf{A}_\theta^T\mathbf{A}_\theta$, which ensures that the matrix is symmetric.

Typically, one can use an ℓ_2 loss of the form $\|\mathbf{e}_\theta - \tilde{\mathbf{e}}\|^2$, where $\tilde{\mathbf{e}}$ is the ground-truth smallest eigenvector. The standard approach to addressing this, as followed in [16,37], consists of explicitly differentiating this loss w.r.t. $\mathbf{e}_\theta$, then $\mathbf{e}_\theta$ w.r.t. $\mathbf{A}_\theta$ and finally $\mathbf{A}_\theta$ w.r.t. θ via backpropagation. As discussed above, however, this is not optimization friendly.

To overcome this, we propose to define a new loss motivated by the linear equation that defines eigenvectors and eigenvalues. Specifically, if $\mathbf{e}_\theta$ is an eigenvector of $\mathbf{A}_\theta^T\mathbf{A}_\theta$ with eigenvalue λ, it satisfies $\mathbf{A}_\theta^T\mathbf{A}_\theta\mathbf{e}_\theta = \lambda\mathbf{e}_\theta$. Since eigenvectors have unit-norm, i.e., $\mathbf{e}_\theta^T\mathbf{e}_\theta = 1$, multiplying both sides of this equation from the left by $\mathbf{e}_\theta^T$ yields

$$\mathbf{e}_\theta^T\mathbf{A}_\theta^T\mathbf{A}_\theta\mathbf{e}_\theta = \lambda \; . \tag{3}$$

In this paper, we consider zero eigenvalue problems, that is, $\lambda = 0$. Since $\mathbf{A}_\theta^T\mathbf{A}_\theta$ is positive semi-definite, we have that $\mathbf{e}^T\mathbf{A}_\theta^T\mathbf{A}_\theta\mathbf{e} \geq 0$ for any $\mathbf{e}$. Given the ground-truth eigenvector $\tilde{\mathbf{e}}$ that we seek to predict, this lets us define the loss function

$$L_{eig}(\theta) = \tilde{\mathbf{e}}^T\mathbf{A}_\theta^T\mathbf{A}_\theta\tilde{\mathbf{e}} \; . \tag{4}$$

Intuitively, this loss aims to find the parameters θ such that $\tilde{\mathbf{e}}$ is an eigenvector of the resulting matrix $\mathbf{A}_\theta^T\mathbf{A}_\theta$ with minimum eigenvalue, that is, zero in our case, assuming that we can truly reach the global minimum of our loss. However, this loss alone has multiple, globally-optimal solutions, including the trivial one $\mathbf{A}_\theta = 0$.

To address this, we note that this trivial solution has not only one zero eigenvalue corresponding to eigenvector $\tilde{\mathbf{e}}$, but that all its eigenvalues are zero. Since, in practice, we typically search for matrices that have a single zero eigenvalue, we propose to maximize the projection of the data along the directions orthogonal to $\tilde{\mathbf{e}}$. Such a projection can be achieved by making use of the orthogonal complement to $\tilde{\mathbf{e}}$, given by $(\mathbf{I} - \tilde{\mathbf{e}}\tilde{\mathbf{e}}^T)$, where $\mathbf{I}$ is the identity matrix. By defining $\bar{\mathbf{A}}_\theta = \mathbf{A}_\theta(\mathbf{I} - \tilde{\mathbf{e}}\tilde{\mathbf{e}}^T)$, we can then re-write our loss function as

$$\tilde{L}(\theta) = \tilde{\mathbf{e}}^T \mathbf{A}_\theta^T \mathbf{A}_\theta \tilde{\mathbf{e}} - \alpha \text{tr}\left(\bar{\mathbf{A}}_\theta^T \bar{\mathbf{A}}_\theta\right) \ , \tag{5}$$

where $\text{tr}(\cdot)$ computes the trace of a matrix and α sets the relative influence of the two terms. Note that we can apply the same strategy to cases where multiple eigenvalues are zero, by reducing the orthogonal space to only the directions corresponding to non-zero eigenvalues, and introducing the first term for all eigenvectors whose eigenvalues we want to be zero.

For numerical stability, we further propose to bound the second term in the range $[0, 1]$. To do so, we therefore re-write our loss as

$$L(\theta) = \tilde{\mathbf{e}}^T \mathbf{A}_\theta^T \mathbf{A}_\theta \tilde{\mathbf{e}} + \alpha \exp\left(-\beta \text{tr}\left(\bar{\mathbf{A}}_\theta^T \bar{\mathbf{A}}_\theta\right)\right) \ , \tag{6}$$

where β is a scalar. This loss is fully differentiable, and can thus be used to learn the parameters θ of a deep network. Since it does not explicitly depend on performing an eigendecomposition at every iteration of the optimization, it suffers from neither the eigenvector switching problem, nor the non-unique eigenvalue problem.

4.2 Learning to Predict Weights

In practice, the problem of interest is often more constrained than training a network to directly output a matrix $\mathbf{A}_\theta$. In particular, in this paper, we consider problems where the goal is to predict a weight w_i for each element of the input. This typically leads to formulations where $\mathbf{A}_\theta^T \mathbf{A}_\theta$ has the form $\mathbf{X}^T \mathbf{W} \mathbf{X}$, with $\mathbf{X}$ a data matrix and $\mathbf{W}$ a diagonal matrix whose elements are the w_is. Below, we introduce the formulation for each of the applications in our experiments.

Outlier Rejection with 3D Points. To show that we can indeed back-propagate nicely through the proposed loss formulation where directly using the analytical gradient fails, we first briefly revisit the toy outlier rejection problem used to motivate our approach in Section 1. For this experiment, we do not train a Deep Network, or perform any learning procedure. Instead, given N 3D points $\mathbf{x}_i$, including inliers and outliers, we directly optimize the weight w_i of each point. At every step of optimization, given the current weight values, we compute the weighted mean of the points $\mu = \frac{1}{\sum_{i=1}^N w_i} \sum_{i=1}^N w_i \mathbf{x}_i$. Let $\mathbf{X}$ be the $3 \times N$ matrix of mean-subtracted 3D points. We then compute the weighted covariance matrix $\mathbf{C} = \mathbf{X}^T \mathbf{W} \mathbf{X}$, where $\mathbf{W}$ is a diagonal matrix whose elements are the w_is. The smallest eigenvector of $\mathbf{C}$ then defines the direction of noise.

Given the ground-truth such eigenvector $\tilde{\mathbf{e}}$, let $\bar{\mathbf{X}} = \mathbf{I} - \tilde{\mathbf{e}}\tilde{\mathbf{e}}^T$. We adapt the general formulation of Eq. 6 and formulate the outlier rejection problem as

$$\underset{\mathbf{w}}{\text{minimize}} \quad \tilde{\mathbf{e}}^T\mathbf{X}^T\mathbf{W}\mathbf{X}\tilde{\mathbf{e}} \; + \alpha \exp\left(-\beta tr(\bar{\mathbf{X}}^T\mathbf{W}\bar{\mathbf{X}})\right) \; . \tag{7}$$

Note that this translates directly to Eq. 6 by defining $\mathbf{A}_\theta = \mathbf{W}^{\frac{1}{2}}\bar{\mathbf{X}}$, where $\mathbf{W}^{\frac{1}{2}}$ is a diagonal matrix with elements $\sqrt{w_i}$.

Keypoint Matching with the Essential Matrix. For this task, to isolate the effect of the loss function only, we followed the same setup as in [37]. Specifically, we used the same network architecture as in [37], which takes C correspondences between two 2D points as input and outputs a C-dimensional vector of weights, that is, one weight for each correspondence. Formally, let

$$\mathbf{q}_i = [u_i, v_i, u'_i, v'_i]^T \; , \tag{8}$$

encode the coordinates of correspondence i in the two images. Following the 8 points algorithm [23], we construct as matrix $\mathbf{X} \in \mathbb{R}^{C\times 9}$, each row of which is computed from one correspondence vector $\mathbf{q}_i$ as

$$\mathbf{X}^{(i)} = [u_iu'_i, u_iv_i, u_i, v_iu'_i, v_iv'_i, v_i, u'_i, v'_i, 1] \; , \tag{9}$$

where $\mathbf{X}^{(i)}$ denotes row i of $\mathbf{X}$. A weighted version of the 8 points algorithm [39] then computes the essential matrix as the smallest eigenvector of $\mathbf{X}^T\mathbf{W}\mathbf{X}$, with $\mathbf{W}$ the diagonal matrix of weights.

Let $\bar{\mathbf{X}} = \mathbf{X}(\mathbf{I} - \tilde{\mathbf{e}}\tilde{\mathbf{e}}^T)$, where $\tilde{\mathbf{e}}$ is the ground-truth eigenvector representing the true essential matrix. We write our eigendecomposition-free essential loss as

$$L(\mathbf{W}) = \tilde{\mathbf{e}}^T\mathbf{X}^T\mathbf{W}\mathbf{X}\tilde{\mathbf{e}} + \alpha \exp\left(-\beta \text{tr}\left(\bar{\mathbf{X}}^T\mathbf{W}\bar{\mathbf{X}}\right)\right) \; . \tag{10}$$

Given a set of training samples, consisting of N image pairs with ground-truth essential matrices, we can then use this loss, instead of the classification loss or essential loss of [37], to train a network to predict the weights.

Note that, as suggested by [11] and done in [37], we use the 2D coordinates normalized to $[-1, 1]$ using the camera intrinsics as input to the network.

When calculating the loss, as suggested by [12], we move the centroid of the reference points to the origin of the coordinate system and scale the points so that their RMS distance to the origin is equal to $\sqrt{2}$. This means that we also have to scale and translate $\tilde{\mathbf{e}}$ accordingly.

3D-to-2D Correspondences for Pose Estimation. The goal of this problem, also known as the Perspective-n-Point (PnP) problem [21], is to determine the absolute pose (rotation and translation) of a calibrated camera, given known 3D points and corresponding 2D image points.

For this task, as we are still dealing with sparse correspondences, we use the same network architecture as for 2D-to-2D correspondences, except that we now have one additional input dimension, since we have *3D*-to-2D correspondences.

This network takes C correspondences between 3D and 2D points as input and outputs a C-dimensional vector of weights, one for each correspondence.

Mathematically, we can denote the input correspondences as

$$\mathbf{q}_i = [x_i, y_i, z_i, u_i, v_i]^T \ , \tag{11}$$

where x_i, y_i, z_i are the coordinates of a 3D point, and u_i, v_i denote the corresponding image location. According to [11], we have

$$f_{scale} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}, \mathbf{t} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \\ p_5 & p_6 & p_7 & p_8 \\ p_9 & p_{10} & p_{11} & p_{12} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \ . \tag{12}$$

To recover the pose, we then follow the Direct Linear Transform (DLT) method [11]. This consists of constructing the matrix $\mathbf{X} \in \mathbb{R}^{2C \times 12}$, every two rows of which are computed from one correspondence $\mathbf{q}_i$ as

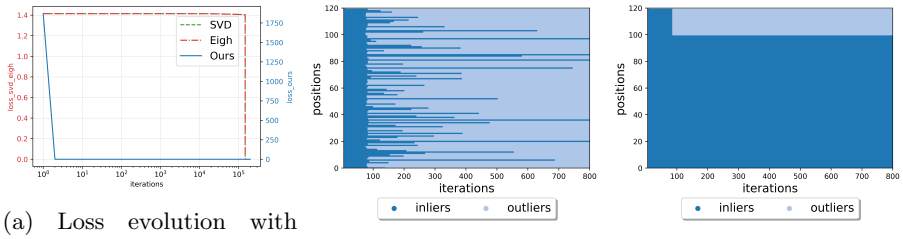
$$\begin{bmatrix} \mathbf{X}^{(2i-1)} \\ \mathbf{X}^{(2i)} \end{bmatrix} = \begin{bmatrix} x_i & y_i & z_i & 1 & 0 & 0 & 0 & 0 & -u_i x_i & -u_i y_i & -u_i z_i & -u_i \\ 0 & 0 & 0 & 0 & x_i & y_i & z_i & 1 & -v_i x_i & -v_i y_i & -v_i z_i & -v_i \end{bmatrix} \ , \tag{13}$$

where $\mathbf{X}^{(i)}$ denotes row i of $\mathbf{X}$. Then, the solution of the weighted PnP problem can be obtained as the eigenvector of $\mathbf{X}^T\mathbf{W}\mathbf{X}$ corresponding to the smallest eigenvalue. Therefore, we can define a PnP loss similar to the one of Eq. 10 for 2D-to-2D correspondences, but with $\mathbf{X}$ defined as discussed above, and, given N training samples, each consisting of a set of 3D-to-2D correspondences with corresponding ground-truth eigenvector encoding the pose, train a network to predict weights such that we obtain the correct pose via DLT. As in the 2D-to-2D case, we use the normalized coordinate system for the 2D coordinates.

Note that the characteristics of the rotation matrix, that is, orthogonality and determinant 1, are not preserved by the DLT solution. Therefore, to make the result a valid rotation matrix, we refine the DLT results by the generalized Procrustes algorithm [8,30], which is a common post-processing technique for PnP algorithms. Note that this step is not involved during training, but only in the validation process to select the best model and at test time.

5 Experiments

We now present our results for the three tasks discussed above, that is, plane fitting as in Section 2, distinguishing good keypoint correspondences from bad ones, and solving the Perspective-n-Point (PnP) problem. We rely on a TensorFlow implementation using the Adam [18] optimizer, with a learning rate of 10^{-4}, unless stated otherwise, and default parameters. When training a network for keypoint matching and PnP, we used mini-batches of 32 samples and, in the plane fitting case, we also tested vanilla gradient descent.



(a) Loss evolution with SVD in the toy example.

(b) Inliers with SVD

(c) Inliers with Ours

Fig. 2: **Plane fitting in the presence of one or multiple outliers.** We report results for Singular Value Decomposition (SVD), self-adjoint Eigendecomposition (Eigh), and for our loss function. For each loss, we tried multiple learning rates within the range $[10^{-5}, 1]$ and report the best results in terms of convergence. (a) Loss evolution with a single outlier. (b) With multiple outliers, the SVD baseline discards many inliers (Positions 1 to 100 are true inliers), while accepting outliers. By contrast, as shown in (c), our approach correctly rejects the outliers and accepts the inliers.

Plane Fitting. The setup is the one discussed in Section 2. We randomly sampled 100 3D points on the $z = 1$ plane. Specifically, we uniformly sampled $x \in [0, 40]$ and $y \in [0, 2]$. We then added zero-mean Gaussian noise with standard deviation 0.001 in the z dimension. We also generated outliers in a similar way, where x and y are uniformly sampled in the same range, and z is sampled from a Gaussian distribution with mean 50 and standard deviation 5. For the baselines that directly use the analytical gradients of SVD and ED, we take the objective function to be $\min \|\mathbf{e}_{min}(\mathbf{w}) \pm \mathbf{e}_{gt}\|_2$, where $\mathbf{e}_{min}(\mathbf{w})$ is the minimum eigenvector of $\mathbf{X}^\top \mathbf{W} \mathbf{X}$ in Eq. 7 and $\mathbf{e}_{gt}$ is the ground-truth noise direction, which is also the plane normal and is the vector $[0, 0, 1]$ in this case. Note that we consider both $+\mathbf{e}_{gt}$ and $-\mathbf{e}_{gt}$ and take the minimum distance, denoted by the $\pm$ and the min in the loss function. For this problem, both solutions are correct due to the sign ambiguity of ED, which should be taken into account.

We consider two ways of computing analytical gradients, one using the SVD and the other the self-adjoint eigendecomposition (Eigh), which both yield mathematically valid solutions. To implement our approach, we rely on Eq. 7.

Fig. 2(a) shows the evolution of the loss as the optimization proceeds when using vanilla gradient descent and with a single outlier. Note that SVD and Eigh have exactly the same behavior because they constitute two equivalent ways of solving the same problem. Using gradient descent in conjunction with either one initially yields a very slow decrease in the loss function, until it suddenly drops to zero after millions of iterations, when a switch of the eigenvector with the smallest eigenvalue occurs. By contrast, our approach produces a much more gradual decrease in the loss.

We also evaluate the behavior of our method and the baselines in the presence of more outliers. Both our method and the baseline present the same convergence

patterns as before, but, as shown in Fig. 2 (b,c), our approach correctly recovers the inliers and outliers, while the SVD baseline discards many outliers and even accepts outliers. Note that, while in this example the SVD- or Eigh-based methods converge, in the more complex cases below, this is not always true.

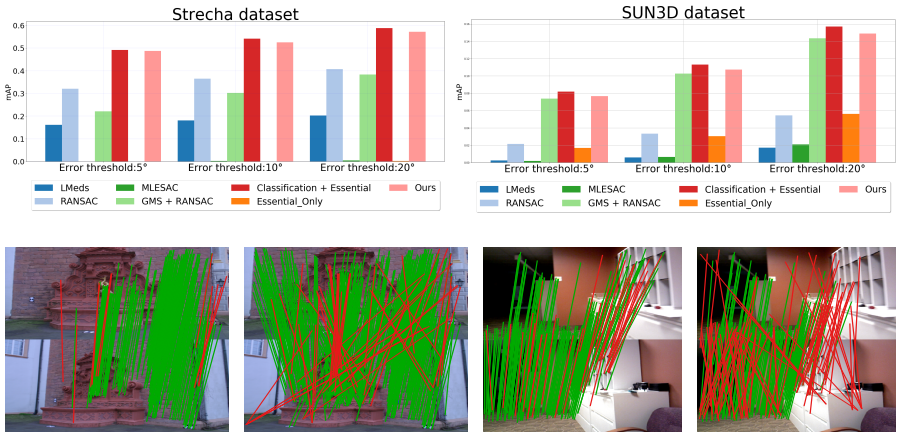
Keypoint Matching. To evaluate our method on a real-world problem, we use the SUN3D dataset [36]. For a fair comparison, we trained our network on the same data as [37], that is, the "brown-bm-3-05" sequence, and evaluate it on the test sequences used for testing in [34,37] . Additionally, to show that our method is not overfitting, we also test on a completely different dataset, the "fountain-P11" and "Herz-Jesus-P8" sequences of [32].

We follow the evaluation protocol of [37], which constitutes the state-of-the-art in keypoint matching, and only change the loss function to our own loss of Eq. 10. We use $\alpha = 10$ and $\beta = 10^{-3}$, which we empirically found to work well for 2D-to-2D keypoint matching. We compare our method against that of [37], both in its original implementation that involves minimizing a classification loss first and then without that initial step, which we denote as "Essential_Only". The latter is designed to show how critical the initial classification-based minimization of [37] is. In addition, we also compare against standard RANSAC [4], LMeds [31], MLESAC [33], and GMS [2] to provide additional reference points. We do this in terms of the performance metric used in [37] and referred to as mean Average Precision (mAP). This metric is computed by observing the ratio of accurately recovered poses given a certain maximum threshold, and taking the area under the curve of this graph.

We summarize the results in Fig. 3. Our approach performs on par with [37], the state-of-the-art method for keypoint matching, and outperforms all the other baselines, without the need of any pre-training. Importantly, "Essential_Only" severely underperforms and even often fails completely. In short, instead of having to find a workaround to the eigenvector switching problem as in [37], we can directly optimize our objective function, which is far more generally applicable. Furthermore, the workaround in [37] would converge to a sub-optimal solution, as it the classification loss depends on a user-selected decision boundary, that is, a heuristic definition of inliers. By contrast, our method can simply discover the inliers automatically while training, thanks to the second term in Eq. 6.

In the bottom row of Fig. 3, we compare the correspondences classified as inlier by our method to those of RANSAC on image pairs from the dataset of [32] and SUN3D, respectively. Note that even the correspondences that are misclassified as inliers by our approach are very close to being inliers. By contrast, RANSAC yields much larger errors.

PnP. Following standard practice for evaluating PnP algorithms [21,6], we generate a synthetic dataset composed of 3D-to-2D correspondences with noise and outliers. Each training example comprises two thousand 3D points, and we set the ground-truth translation of the camera pose $\mathbf{t}_{gt}$ to be their centroid. We then create a random ground-truth rotation $\mathbf{R}_{gt}$, and project the 3D points to



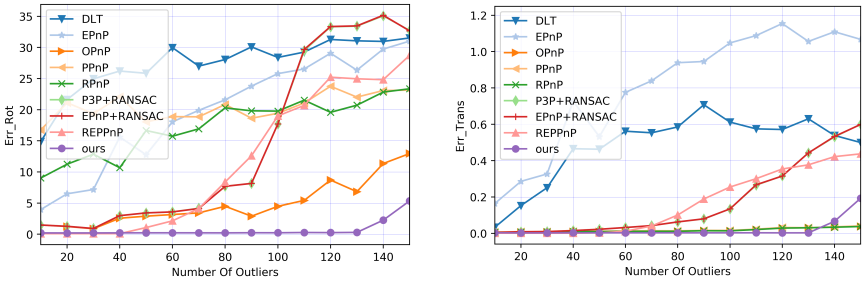
(a) Ours (b) RANSAC (c) Ours (d) RANSAC

Fig. 3: **Results for the keypoint matching task.** Note the significant performance gap between "Essential_Only", which utilizes eigendecomposition directly, and our approach. (Bottom left two images) Comparison of our results with RANSAC results on the "fountain-P11" image pair of [32]. (Bottom right two images) Similar comparison on the "brown-bm-3-05" image pair of SUN3D. We display the correspondences that the algorithms labeled as inliers. True positives are shown in green and false ones in red. The false positives of our approach are still close to being correct, while those of RANSAC are truly wrong.

the image plane of our virtual camera. As in REPPnP [6], we apply Gaussian noise with a standard deviation of 5 to these projections. We generate random outliers by assigning 3D points to arbitrary valid 2D image positions.

We train a neural network with the same architecture as in the keypoint matching case, except that it now takes 3D-to-2D correspondences as input. We empirically found that $\alpha = 1$ and $\beta = 5 \times 10^{-3}$ works well for this task. During training, to learn to be robust to outliers, we randomly select between 100 and 1000 of the two thousand matches and turn them into outliers. In other words, the two thousand training matches will contain a random number of outliers that our network will learn to filter out.

We compare our method against modern PnP methods, EPnP [21], OPnP [40], PPnP [8], RPnP [22] and REPPnP [6]. We also evaluate the DLT [11], since our loss formulation is based on it. Among these methods, REPPnP is the one most specifically designed to handle outliers. As in the keypoint matching case, we tried to compute the results of a network relying explicitly on eigendecomposition and minimizing the ℓ_2 norm of the difference between the ground-truth eigenvector and the predicted one. However, we found that such a network was unable to converge. We also report the performance of two commonly used baselines that leverage RANSAC [7], P3P [19]+RANSAC and EPnP+RANSAC. For



(a) Rotation error (degrees)

(b) Translation error

Fig. 4: **Quantitative PnP results.** Rotation and translation errors for our method and several baselines. Our method gives extremely stable results despite the abundance of outliers, whereas all compared methods perform significantly worse as the number of outliers increases. Even when these methods perform well on either rotation or translation, they do not perform well on both. By contrast, Ours yields near zero errors for both measures up to 130 outliers (i.e., 65%).

other methods, RANSAC did not bring noticeable improvements, and we omitted them in the graph for better visual clarity.

For this comparison, we use standard rotation and translation error metrics [5]. Specifically, we report the closest arc distance in radians for the rotation matrix measured using quaternions, and the distance between the translation vectors normalized by the ground truth. To demonstrate the effect of outliers at test time, we fix the number of matches to 200 and vary the number of outliers from 10 to 150. We run each experiment 100 times and report the average.

Fig. 4 summarizes the results. We outperform all other methods significantly, especially when the number of outliers increases. REPPnP is the one competing method that seems least affected. As long as the number of outliers is small, it is on a par with us but passed a certain point—when there are more than 40 outliers, that is, 20% of the total—its performance, particularly in terms of rotation error, decreases quickly whereas ours does not.

We evaluated our PnP approach on the real dataset of [13]. Specifically, the 3D points in this dataset were obtained using the SfM algorithm of [35], which also provides a rotation matrix and translation vector for each image. We treat these rotations and translations as ground truth to compare different PnP algorithms. Given a pair of images, we extract SIFT features at the reprojection of the 3D points in one image, and match these features to SIFT keypoints detected in the other image. This procedure produces erroneous correspondences, which a robust PnP algorithm should discard. In this example, we used the model trained on the synthetic data described before. Note that we apply the model **without any fine-tuning**, that is, the model is only trained with purely synthetic data. We observed that, except for EPnP+RANSAC, OPnP and P3P+RANSAC, the predictions of the baselines are far from the ground truth, which led to points

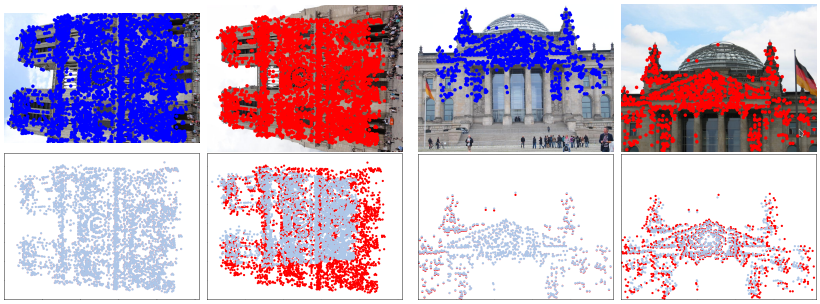


Fig. 5: **Qualitive PnP results.** Top: Two pairs of images (Left: Reichstag, Right: Notre-dame). For each pair, we seek to estimate the pose in the second image. Bottom: For each pair, we show in gray the reprojection of the 3D point cloud after applying the rotation and translation predicted by our model and EPnP+RANSAC, respectively. The red dots correspond to the ground-truth locations. Note that our model's predictions cover the ground truth much more closely than the baseline.

reprojecting outside the image. In Fig. 5, we compare the reprojection of the 3D points on the input image after applying the rotation and translation obtained with our model and with EPnP+RANSAC. Note our better accuracy.

6 Conclusion

We have introduced a novel approach to training deep networks that rely on losses computed from an eigenvector corresponding to a zero eigenvalue of a matrix defined by the network's output. Our loss does not suffer from the numerical instabilities of analytical differentiation of eigendecomposition, and converges to the correct solution much faster. Our approach achieves the state-of-the-art results on the tasks of keypoint matching and outlier rejection for the PnP problem.

Many Computer Vision tasks rely on least-square solutions to linear systems. We will therefore investigate the use of our approach for other ones. Furthermore, we hope that our work will contribute to imbuing Deep Learning techniques with traditional Computer Vision knowledge, thus avoiding discarding decades of valuable research, and develop more principled frameworks.

7 Acknowledgements

This research was partially supported by the National Natural Science Foundation of China: Grant 61603291, the program for introducing talents of discipline to university B13043 and the National Science, Technology Major Project: 2018ZX01008103, and by a grant from the Swiss Innovation Agency (CTI/InnoSuisse). This work was performed while Zheng Dang was visiting the CVLab at EPFL.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A System for Large-Scale Machine Learning. In: USENIX Conference on Operating Systems Design and Implementation. pp. 265–283 (2016)
2. Bian, J., Lin, W., Matsushita, Y., Yeung, S., Nguyen, T., Cheng, M.: GMS: Grid-Based Motion Statistics for Fast, Ultra-Robust Feature Correspondence. In: CVPR (2017)
3. Brachmann, E., Krull, A., Nowozin, S., Shotton, J., Michel, F., Gumhold, S., Rother, C.: DSAC – Differentiable RANSAC for Camera Localization. ARXIV (2016)
4. Cantzler, H.: Random Sample Consensus (RANSAC) (2005), cVonline
5. Crivellaro, A., Rad, M., Verdie, Y., Yi, K.M., Fua, P., Lepetit, V.: Robust 3D Object Tracking from Monocular Images Using Stable Parts. PAMI (2018)
6. Ferraz, L., Binefa, X., Moreno-noguer, F.: Very Fast Solution to the PnP Problem with Algebraic Outlier Rejection. In: CVPR. pp. 501–508 (2014)
7. Fischler, M., Bolles, R.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Communications ACM **24**(6), 381–395 (1981)
8. Garro, V., Crosilla, F., Fusiello, A.: Solving the PnP Problem with Anisotropic Orthogonal Procrustes Analysis. In: 3DPVT. pp. 262–269 (2012)
9. Giles, M.: Collected Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation. In: Advances in Automatic Differentiation. pp. 35–44 (2008)
10. Handa, A., Bloesch, M., Patraucean, V., Stent, S., McCormac, J., Davison, A.: Gvnn: Neural Network Library for Geometric Computer Vision. In: ECCV (2016)
11. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press (2000)
12. Hartley, R.: In Defense of the Eight-Point Algorithm. PAMI **19**(6), 580–593 (June 1997)
13. Heinly, J., Schoenberger, J., Dunn, E., Frahm, J.M.: Reconstructing the World in Six Days. In: CVPR (2015)
14. Huang, G., Liu, Z., Weinberger, K., van der Maaten, L.: Densely Connected Convolutional Networks. In: CVPR (2017)
15. Huang, Z., Wan, C., Probst, T., Gool, L.V.: Deep learning on lie groups for skeleton-based action recognition. In: CVPR. pp. 6099–6108 (2017)
16. Ionescu, C., Vantzos, O., Sminchisescu, C.: Matrix backpropagation for Deep Networks with Structured Layers (2015)
17. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial Transformer Networks. In: NIPS. pp. 2017–2025 (2015)
18. Kingma, D., Ba, J.: Adam: A Method for Stochastic Optimisation. In: ICLR (2015)
19. Kneip, L., Scaramuzza, D., Siegwart, R.: A Novel Parametrization of the Perspective-Three-Point Problem for a Direct Computation of Absolute Camera Position and Orientation. In: CVPR. pp. 2969–2976 (2011)
20. Law, M., Urtasun, R., Zemel, R.S.: Deep spectral clustering learning. In: ICML. pp. 1985–1994 (2017)
21. Lepetit, V., Moreno-noguer, F., Fua, P.: EPnP: An Accurate $o(n)$ Solution to the PnP Problem. IJCV (2009)

22. Li, S., Xu, C., Xie, M.: A Robust O(n) Solution to the Perspective-N-Point Problem. PAMI pp. 1444–1450 (2012)
23. Longuet-Higgins, H.: A Computer Algorithm for Reconstructing a Scene from Two Projections. Nature **293**, 133–135 (1981)
24. Murray, I.: Differentiation of the Cholesky Decomposition. arXiv Preprint (2016)
25. Nister, D.: An Efficient Solution to the Five-Point Relative Pose Problem. In: CVPR (June 2003)
26. Papadopoulo, T., Lourakis, M.: Estimating the jacobian of the singular value decomposition: Theory and applications. In: ECCV. pp. 554–570 (2000)
27. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NIPS Autodiff Workshop (2017)
28. Raguram, R., Chum, O., Pollefeys, M., Matas, J., Frahm, J.M.: USAC: A Universal Framework for Random Sample Consensus. PAMI **35**(8), 2022–2038 (2013)
29. Rousseeuw, P., Leroy, A.: Robust Regression and Outlier Detection. Wiley (1987)
30. Schönemann, P.: A Generalized Solution of the Orthogonal Procrustes Problem. Psychometrika **31**(1), 1–10 (1966)
31. Simpson, D.: Introduction to Rousseeuw (1984) Least Median of Squares Regression. In: Breakthroughs in Statistics, pp. 433–461. Springer (1997)
32. Strecha, C., Hansen, W., Van Gool, L., Fua, P., Thoennessen, U.: On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery. In: CVPR (2008)
33. Torr, P., Zisserman, A.: MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. CVIU **78**, 138–156 (2000)
34. Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., Brox, T.: Demon: Depth and Motion Network for Learning Monocular Stereo. In: CVPR (2017)
35. Wu, C.: Towards Linear-Time Incremental Structure from Motion. In: 3DV (2013)
36. Xiao, J., Owens, A., Torralba, A.: SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels. In: ICCV (2013)
37. Yi, K.M., Trulls, E., Ono, Y., Lepetit, V., Salzmann, M., Fua, P.: Learning to Find Good Correspondences. In: CVPR (2018)
38. Zamir, A.R., Wekel, T., Agrawal, P., Malik, J., Savarese, S.: Generic 3D Representation via Pose Estimation and Matching. In: ECCV (2016)
39. Zhang, Z.: Determining the Epipolar Geometry and Its Uncertainty: A Review. IJCV **27**(2), 161–195 (1998)
40. Zheng, Y., Kuang, Y., Sugimoto, S., Astrom, K., Okutomi, M.: Revisiting the PnP Problem: A Fast, General and Optimal Solution. In: ICCV (2013)