FAST AND INTERPRETABLE 2D HOMOGRAPHY DECOMPOSITION: SIMILARITY-KERNEL-SIMILARITY AND AFFINE-CORE-AFFINE TRANSFORMATIONS

Shen Cai^{†,*},Zhanhao Wu[†],Lingxi Guo[†],Jiachun Wang[†],Siyu Zhang[†],Junchi Yan[‡],and Shuhan Shen[§]

†Visual and Geometric Perception Lab, Donghua University

†Department of Computer Science and Engineering, Shanghai Jiao Tong University

§National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences

*Corresponding author: hammer_cai@163.com

ABSTRACT

In this paper, we present two fast and interpretable decomposition methods for 2D homography, which are named Similarity-Kernel-Similarity (SKS) and Affine-Core-Affine (ACA) transformations respectively. Under the minimal 4-point configuration, the first and the last similarity transformations in SKS are computed by two anchor points on target and source planes, respectively. Then, the other two point correspondences can be exploited to compute the middle kernel transformation with only four parameters. Furthermore, ACA uses three anchor points to compute the first and the last affine transformations, followed by computation of the middle core transformation utilizing the other one point correspondence. ACA can compute a homography up to a scale with only 85 floating-point operations (FLOPs), without even any division operations. Therefore, as a plug-in module, ACA facilitates the traditional feature-based Random Sample Consensus (RANSAC) pipeline, as well as deep homography pipelines estimating 4-point offsets. In addition to the advantages of geometric parameterization and computational efficiency, SKS and ACA can express each element of homography by a polynomial of input coordinates (7th degree to 9th degree). extend the existing essential Similarity-Affine-Projective (SAP) decomposition and calculate 2D affine transformations in a unified way. Source codes are released in https: //github.com/cscvlab/SKS-Homography.

Index Terms— Homography, square linear system, matrix decomposition, geometric transformation, tensorization

1. INTRODUCTION

Planar homography, also referred as two-dimensional (2D) projective transformation, plays an important role in many

tasks of geometric vision, such as camera calibration [92], plane based pose estimation [28], image stitching [90] and monocular motion estimation [22]. A 2D homography denoted by a 3*3 homogeneous matrix has 8 degrees of freedom (DOF). Assuming coplanar geometric primitives (such as points, lines and conics) are given, a homography may be computed under the minimal condition or under the overdetermined condition, depending on DOF of primitives. In this paper, we focus on the homography computation problem under the minimal condition. Previous approaches explore various algebraic or geometric ways to study the minimal planar configurations. According to the categories of geometric primitives, they could be roughly divided into three categories: i) four points or lines; ii) conic-involved patterns; iii) special patterns used in the hierarchical homography decomposition. Please see Fig. 1 for a brief overview.

i) Four points or lines. Among all planar minimal configurations, 4-point pattern is the most common partly due to the mature interest points extraction and matching algorithms, such as traditional SIFT [56], SURF [16], ORB [72] and deep learning based LIFT [88], SOSNet [80], SuperPoint [30]. Moreover, predicting 4-point offsets in target image (while fixing 4-point in source image) is also popular in deep homography estimation algorithms [63, 54, 21, 65, 47, 75, 66] as this parameterization of homography [10] is meaningful and beneficial for training neural networks [29].

Previous 4-point homography computation methods firstly construct a square system of linear equations, followed by adopting different mathematical solvers to improve computational efficiency for the constructed linear system. The classical homography computation method is the normalized direct linear transform (NDLT) [40] [p. 109], which normalizes data first and then adopts singular value decomposition (SVD) [34] [p. 111] to solve the formed 9-variable linear system. This method is called NDLT-SVD in this paper to highlight its mathematical solver. The HO-SVD method [38] utilizes the perpendicular properties of columns in coefficient matrix to simplify the 9-variable linear system to a 3-variable style, which significantly reduces the computation amount

The work is partially supported by National Major State Research Development Program (2020AAA0107600), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), and the Foundation of Key Laboratory of Artificial Intelligence, Ministry of Education, P.R. China (AI2020003).

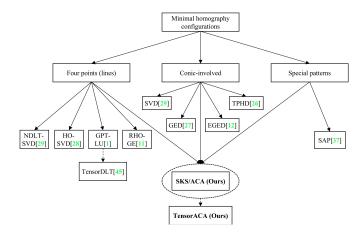


Fig. 1. Sketch of homography computation methods under the minimal condition. According to various primitive configurations and underlying mathematical principles, solving methods may differ considerably. The proposed SKS and ACA can deal with three categories of primitive configurations in a unified way, while is suitable to be tensorized in deep homography pipeline.

of SVD without loss of robustness. However, the above two methods simultaneously dealing with 4-point and n-point (n>4) patterns contain many redundant operations when solving square linear systems. Specially designed for the 4-point pattern, GPT-LU [4] forms an square linear system of equations and adopts LU factorization [34] [p. 114] to solve it. RHO-GE [17] designs a customized Gaussian elimination (GE) [34] [p. 111] algorithm for the coefficient matrix A containing a large number of 0 elements, which further accelerates the 4-point homography computation. However, all the above 4-point homography computation methods are limited by the strategy of constructing the square linear system first, which exists a lot of redundant operations. As shown in Fig. 2 (a), for these four methods, four point correspondences (drawn with red dots) are indiscriminately utilized to construct a linear system where the entire homography is unknown.

The above mentioned 4-point homography methods are widely applied in traditional random sample consensus (RANSAC) [33] framework and its variants (e.g., USAC [70] and MAGSAC++ [14]) to estimate homography between two images with outliers. In end-to-end deep homography methods predicting the offsets of four corner points, the adopted solver is the tensorized GPT-LU (firstly named TensorDLT in [63]).

For line primitives which may be extracted by traditional algorithms [43, 35] or deep learning algorithms [52, 69], it is proved that 4-line pattern is geometrically dual to 4-point pattern. Therefore, all the above 4-point homography methods can be directly adopted for the 4-line pattern.

ii) Conic-involved patterns. Conic-involved patterns mainly include two conics or one conic with coplanar points

(lines). There already exist many works to study patterns with a pair of coplanar conics, such as concentric circles [44], coplanar circles [86, 24], confocal conics [37] and coplanar conics [20]. Moreover, the conic-line(point) hybrid configuration is also studied for homography computation and camera calibration [36]. These methods generally build the link between the general intersection points of conics and one special kind of imaginary points, such as the circular points [73] [p. 33], [40] [p. 53], to avoid solving the nonlinear constraints formed by projection equality of one conic correspondence. However, the computational efficiency of these methods is not high as the time-consuming operations, such as SVD, are needed. Furthermore, the geometric meaning of the matrices in these methods is limited without explanation of each parameter.

iii) Special patterns used in the hierarchical homography decomposition. There exists another well-known homography decomposition method [50], [40] [p. 42-43], which express a projective transformation into a geometrical hierarchy chain of essential similarity, affine, and projective (SAP) transformations. The pattern used to hierarchically recover affine and metric properties should include two parts consisting of specific primitives. The first part [40] [p. 49-51] consists of two sets of three collinear points with known length ratios or two two sets of parallel lines, which are used to obtain the vanishing line and compute the essential projective transformation. The second part [40] [p. 56-57] consists of two orthogonal line pairs, which are used to constrain the conic dual to the circular points to compute the affine transformation. However, this decomposition method is almost never used in practice, mainly because its decomposition chain cannot be computed from general primitives, such as unconstrained points or lines.

The above three categories of homography computation methods under the minimal condition have their own drawbacks. The geometric homography decomposition methods require special primitives configurations or have high computational complexity. The algebraic 4-point homography methods lack geometric meaning and are not fully explored in computation efficiency. A desirable homography computation method preferably has the following properties:

- i) Different planar primitives can be handled in a unified way.
 - ii) Parameters involved have clear geometric meanings.
 - iii) Homography is computed at a superior efficiency.
 - iv) Solving process is suitable for being tensorized.

In this paper, a new homography decomposition formula with stratified geometric meaning and high computation efficiency is proposed to deal with the common four point correspondences. This 4-point homography computation method enjoys all aforementioned desirable properties. Specifically, a homography is decomposed into three sub-transformations as shown in Fig. 2 (b), which are *similarity-kernel-similarity*

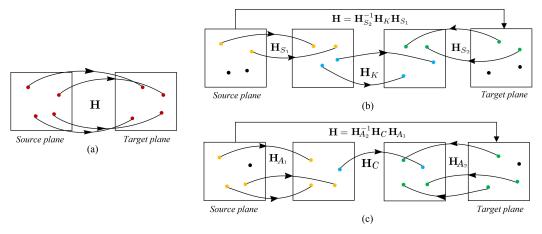


Fig. 2. Comparison of solving process between previous 4-point homography methods and ours. (a) The previous 4-point homography methods (NDLT-SVD [40], HO-SVD [38], GPU-LU [4], RHO-GE [17]) indiscriminately utilize all four point correspondences (drawn with red dots) to construct a linear system (Eq. 3) and solves it using well-established matrix factorization methods. (b) The proposed SKS decomposition includes three stratified sub-transformations, which are the first similarity transformation \mathbf{H}_{S_1} , the middle kernel transformation \mathbf{H}_K and the last similarity transformation \mathbf{H}_{S_2} . Two corresponding points on source plane (drawn with yellow) and target plane (drawn with green) are used to compute \mathbf{H}_{S_1} and \mathbf{H}_{S_2} respectively. The residual two point correspondences (drawn with blue) on the two intermediate planes are used to compute \mathbf{H}_K with 4 DOF. (c) The proposed ACA decomposition is similar to SKS, except three point correspondence is used to compute the affine transformation \mathbf{H}_{A_1} and \mathbf{H}_{A_2} , while one point correspondence is used to compute the core transformation \mathbf{H}_C with 2 DOF.

(SKS) transformations respectively. Two point correspondences as two anchor points (TAP) drawn with yellow dots on source plane and drawn with green dots on target plane are utilized to compute the last and first similarity transformations \mathbf{H}_{S_1} and \mathbf{H}_{S_2} , respectively. The similarity transformation plays the role of normalizing TAP to a canonical form. After transforming the two planes under \mathbf{H}_{S_1} and \mathbf{H}_{S_2} respectively, the other two point correspondences drawn with blue can be used to linearly solve the kernel transformation \mathbf{H}_K with only 4 parameters. Since the complicated and time-consuming operations, such as singular value decomposition (SVD), are avoided, the calculation amount for each step of homography computation is only a few dozens of floating-point operations (FLOPs). Moreover, an improved affine-core-affine (ACA) decomposition shown in Fig. 2 (c) is also derived to further accelerate homography computation. Three anchor points (HAP) in source and target planes are utilized to compute the the last and first affine transformations \mathbf{H}_{A_1} and \mathbf{H}_{A_2} , respectively. The transformed fourth point correspondence is used to solve the middle core transformation \mathbf{H}_C with only 2 parameters.

The main contributions of this paper are concluded as follows:

- i) We propose the similarity-kernel-similarity (SKS) and affine-core-affine (ACA) transformations for 2D homography decomposition, with a variety of matrix factorization forms. We also prove that the existing SAP decomposition is just one special case of our formulas and 2D affine transformations can be computed in a unified way.
 - ii) In geometry, each sub-transformation of SKS or ACA,

including each parameter in them, has a clear meaning. The first and last transformations in SKS or ACA symmetrically normalize target and source planes respectively, while the middle transformation determine the projective distortion between two normalized planes.

- iii) In algebra, SKS is superior to previous 4-point homography methods in computational efficiency. Furthermore, ACA requires only 97 FLOPs (85 FLOPs and no division operations for homographies up to a scale), which is about 5% and 0.4% of the state-of-the-art (SOTA) methods GPT-LU and NDLT-SVD respectively. In addition, we directly derive the polynomial expression of 16 coordinates of four point correspondences for each element of homography.
- iv) SKS or ACA can be integrated into traditional feature-based RANSAC pipelines to replace their default 4-point homography solvers when estimating homography with outliers. At the same time, the tensorized ACA (TensorACA) is also suitable for integration into deep homography pipelines due to its concise homography expression and a small number of vector operations.

The rest of this paper is organized as follows. Section 2 reviews the related works of homography computation. Section 3 presents the derivation of SKS for 4-point pattern. Section 4 illustrates the improved ACA homography decomposition. Section 5 summaries the applications of our method in extension of SAP decomposition, decomposition for 2D affine transformation and tensorization for deep homography pipeline. Section 6 shows the experimental results. Section 7 gives the conclusion and future work.

2. PROBLEM FORMULATION AND RELATED WORKS

The proposed method mainly involves two sub-directions related to homography, which are points based homography computation and homography decomposition, assuming that corresponding coplanar primitives are given. two sub-directions are formulated and reviewed in Sec. 2.1 and Sec. 2.4, respectively. Meanwhile, 4-point homography computation algorithms are generally called as a standalone module in the methods of estimating homography between two images. Thus, the traditional feature-based RANSAC pipelines and deep learning based homography estimation are reviewed in Sec. 2.2 and Sec. 2.3, respectively. Sec. 2.5 reviews methods evaluating computational amount of algorithms. Table 1 and Table 2 summarize the main differences between the proposed methods and the existing homography computation methods to provide an overall understanding and context.

2.1. Points based Homography Computation

Points are the most common geometric primitives. Previous 4-point homography methods [40, 38, 4, 17] follow the same way to construct a square system of linear equations, whose first step is to remove the homogeneous equality of one point correspondence by utilizing the cross-product, i.e.,

$$\mathbf{x}_2 \equiv \mathbf{H} \mathbf{x}_1 \Rightarrow \mathbf{x}_2 \times (\mathbf{H} \mathbf{x}_1) = 0,$$
 (1)

where **H** denotes a 3*3 homography matrix; \mathbf{x}_1 and \mathbf{x}_2 are 3*1 homogeneous vectors of one point correspondence $\{X_1 \xrightarrow{\mathbf{H}} X_2\}$; ' \equiv ' denotes the equality up to a scale.

After some simplifications, two linear equations of homography are obtained,

$$\begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} & -X_2.x * \mathbf{x}_1^{\mathrm{T}} \\ \mathbf{0}^{\mathrm{T}} & -\mathbf{x}_1^{\mathrm{T}} & X_2.y * \mathbf{x}_1^{\mathrm{T}} \end{bmatrix} \mathbf{h} = 0,$$
 (2)

where the suffixes .x and .y denote the coordinates of one 2D point; \mathbf{h} is the vector form of homography.

Stacking the eight linear equations provided by four point correspondences, a linear system of equations is constructed,

$$\mathbf{A}_{8*9} * \mathbf{h}_{9*1} = 0, \tag{3}$$

where A denotes the coefficient matrix.

For the constructed linear system in Eq. 3, previous methods may change its form and apply different matrix factorization methods to solve it. For example, NDLT-SVD [40] [p. 109] first normalizes all points on source and target planes and then applies SVD to solve the formed linear system under the constraint $||\mathbf{h}||=1$. HO-SVD [38] takes advantage of the sparse nature of the coefficient matrix to simplify the linear system representation to three unknowns. Thus the linear system for n points is simplified

to $A_{2n*3}*h_{3*1}=0$, which is adopted by the SOTA plane-based pose estimation method [28] to accelerate homography computation. The above two methods can simultaneously deal with 4-point pattern and n-point (n>4) pattern. As a result, the computational amount (FLOPs) of the above two methods for 4-point pattern is quite large, as shown in the second and third rows in Table 1. Since SVD is inherently iterative, we here give an estimated lower bound of FLOPs based on the OpenCV's implementation.

Specially designed for 4-point pattern, the OpenCV's function getPerspectiveTransform [4] (called GPT-LU here) converts the initial linear system to an inhomogeneous form: $\mathbf{A}_{8*8}*\mathbf{h}_{8*1}=\mathbf{b}_{8*1}$. Then this inhomogeneous linear system is solved by LU factorization [34] [p. 114] which internally contains Gaussian elimination (GE) with partial pivoting¹. Another fast algorithm specifically for 4-point pattern is RHO-GE [17], which utilizes the characteristics of the initial coefficient matrix A_{8*9} (containing a number of 0 and 1 elements) to simplify the process of GE. However, since this customized algorithm gets rid of choosing pivot element, this strategy may suffer from numerical robustness issues and is not recommended for use in linear algebra libraries, such as LAPACK and Eigen. Although these two specialized 4-point homography methods (shown in the fourth and fifth rows in Table 1) improve the computational efficiency, both of them are still limited by the redundancy of the constructed linear system.

All the previous 4-point homography methods shown in Table 1 follow Eq. 3 to construct a linear system. Consequently, the linear system can be solved by using a well-established matrix factorization method. Totally different from the previous 4-point homography methods, both the proposed SKS and ACA directly compute the subtransformations in homography. Therefore, they are not related to the constructed linear system of equations and involve little computation. Moreover, SKS and ACA enjoy geometric meaning in each decomposition matrix, even in each parameter, as well as other algebraic advantages revealed later.

2.2. Traditional Feature-based Pipelines

Traditional pipelines of estimating homography between two images often include two stages. In first stage, feature points are extracted by traditional methods [56, 16, 72] or deep learning based methods [88, 30, 80], followed by coarse descriptor matching through brute force search. In second stage, outliers are removed and final homography is obtained under the RANSAC scheme [33] or its variants, such as PROSAC [25], USAC [70], MAGSAC [13] and MAGSAC++ [14]. These traditional homography estimation

¹There are also other ways to solve this inhomogeneous linear system in various implementations, such as directly inverting **A** and utilizing QR decomposition. Here we choose the most common LU factorization with partial pivoting as a representative for this category of method.

Table 1. Comparison of 4-point homography computation methods. Different from the previous four methods (shown in the second to fifth row), the proposed SKS and ACA do not need to form a linear system $\mathbf{A}*\mathbf{h}=0/\mathbf{b}$ and thus are denoted by \bigotimes . The notation \S means that FLOPs is exactly counted from each arithmetical operation in source codes written in C++ (where no external functions is called), while \ge and \sim denote an estimated lower bound and approximate value of FLOPs respectively. Among all methods, RHO-GE is not robust as pivoting is ignored for rapid calculation. In traditional feature-based pipelines, the previous four methods are integrated as 4-point homography solver. In end-to-end deep homography pipelines, only the tensorized GPT-LU (called TensorDLT by [63]) is adopted to compute homography from 4-point offsets.

1) Method	Formed Linear System	Solving Way	FLOPs	Robustness	Traditional Pipeline	End-to-End Deep Pipeline
2) NDLT-SVD [40]	$\mathbf{A}_{8*9} * \mathbf{h}_{9*1} = 0$	algebraic	≥ 27400	✓	[40, 70, 22]	/
3) HO-SVD [38]	$\mathbf{A}_{8*3} * \mathbf{h}_{3*1} = 0$	algebraic	≥ 1800	✓	[28]	/
4) GPT-LU[4] or TensorDLT[63]	$\mathbf{A}_{8*8} * \mathbf{h}_{8*1} = \mathbf{b}_{8*1}$	algebraic	~1950	✓	[70, 14]	[65, 47, 54], [63, 75, 66, 21]
5) RHO-GE [17]	$\mathbf{A}_{8*9} * \mathbf{h}_{9*1} = 0$	algebraic	223§	×	[17]	/
6) SKS (Ours)	\otimes	algebraic & geometric	169§	✓	1	/
7) ACA (Ours)	\otimes	algebraic & geometric	978	✓	/	/

pipelines differ from several aspects, of which this paper focuses on 4-point homography solver.

The second column from right to left in Table 1 summaries the 4-point homography algorithms in existing traditional pipelines. Specifically, NDLT-SVD is used in the standard RANSAC [40] [p. 123] implemented in OpenCV library, official implementation of USAC and the optimization-free RANSAC in official implementation of ORB-SLAM3 [22]. HO-SVD is used in the SOTA plane-based pose estimation method [28]. GPT-LU is used in USAC and MAGSAC++ implemented in OpenCV library. RHO-GE is proposed in the work [17] mainly based on PROSAC. Due to the circular hypothesis and verifying scheme of RANSAC, the speed of 4-point homography computation becomes non-negligible for the whole process, especially facing high outlier ratios. The proposed SKS or ACA algorithm as a standalone module can be plugged into any of the above traditional pipelines to greatly reduce the runtime of 4-point homography computation step, as well as the running time of whole process.

2.3. Deep Homography Pipelines

Deep learning pipelines of estimating homography between two images often include: (1) extracting feature map using CNN backbones; (2) predicting one kind of parameterization of homography; (3) computing homography. The first deep homography network [29] predicts the offsets of four corresponding corners in target image, which comes from a traditional 4-point parameterization method for homography computation [10]. However, the traditional NDLT-SVD or GPT-LU is also needed to compute 4 -point homography after network inference. The subsequent two works [67, 91] follow this pipeline and still require traditional 4-point homography solvers as a post-processing module. The first end-to-end deep homography method UDHN [63] integrates TensorDLT into neural network and warp image with the com-

puted homography to measure photometric loss between two images. The proposed TensorDLT is actually same to the GPU-LU method to solve the formed inhomogeneous linear system $\mathbf{A}_{8*8} * \mathbf{h}_{8*1} = \mathbf{b}_{8*1}$. The only difference is that TensorDLT is differential and implemented under deep learning framework, such as TensorFlow or PyTorch. Most of recent works [47, 54, 65, 75, 66, 21] follow this methodology of predicting the offsets of four corners and adopt TensorDLT as the 4-point homography solver in their pipelines. The main contributions of these methods are reflected in other aspects. For example, a multi-scale neural network is proposed to predict motion mask and handle dynamic scenes [47]. An unsupervised deep image stitching method is proposed to predict coarse homography for large-baseline scenes and reconstruct the stitched images from feature to pixel [65]. Another recent work iteratively estimates 4-point offsets and achieves realtime inference [21]. The rightmost column in Table 1 lists the end-to-end deep homography pipelines adopting Tensor-DLT. Due the concise representation of involving parameter and the form of stratified transformations, the proposed SKS or ACA algorithm is capable to be tensorized in deep homography pipeline to accelerate homography computation.

It is worth noting that there also exist two other kinds of homography parameterization in deep learning pipelines. The first is inverse compositional Lucas-Kanade (IC-LK) [11]. The works [23, 93] fuse IC-LK into their deep homography pipelines and predict eight parameters of a transformed homography. The second is to estimate the coefficients of eight displacement field bases consisting of projective distortion for small baseline scenes [87, 41]. Although these methods are not directly related to 4-point homography, the novel parameterization forms which are their emphases are also investigated in this paper.

Table 2. Overview of geometry-based homography decomposition. \bigotimes denotes the SAP method is not based on anchor points (AP) and does not have AP related FLOPs. † denotes that all previous AP based methods use SVD to obtain decomposed matrices (refer to [34] [p. 298] for a theoretical FLOPs of SVD). Our SKS and ACA avoid SVD and their FLOPs drop dramatically (the details are illustrated in Sec. 3.4 and Sec. 4.4 respectively.).

1) Method	Pattern Configuration	Anchor Points (AP)	Decomposition Formula	AP related FLOPs		
2) SVD [40]	1. two circles	two circular points:	$[\tilde{I}\tilde{J}]$ = $\mathbf{U}[IJ]\mathbf{U}^{\top}$	$12*3^3 = 324^{\dagger}$		
2) 3 (D [40]	2. one circle and the infinity line	$\{I,J\}$	$\mathbf{H} = \mathbf{U}\mathbf{H}_S$	12*3 -324		
3) GED [37]	two confocal conics	two circular points:	$\mathbf{\tilde{C}}_1 - \lambda_{min} \mathbf{\tilde{C}}_2 = [\tilde{I}\tilde{J}] = \mathbf{U}[IJ]\mathbf{U}^{ op}$	$12*3^3 = 324^{\dagger}$		
3) GED [37]	two comocar comes	$\{I,J\}$	$\mathbf{H} = \mathbf{U}\mathbf{H}_S$	12.0 -021		
		two complex conjugate points:	$ [MN] = \mathbf{H}_P^{-1}[IJ]\mathbf{H}_P^{-\top} $ $ [\tilde{M}\tilde{N}] = \mathbf{H}_Q^{-1}[IJ]\mathbf{H}_Q^{-\top} $			
4) EGED [20]	two general conics	$\{M, N\}$		$2*12*3^3 = 648^{\dagger}$		
		[172, 17]	$\mathbf{H} = \mathbf{H}_Q^{-1} \mathbf{H}_S \mathbf{H}_P$			
	1 and comic and one line	tuvo mool mainto.	$[MN] = \mathbf{H}_P^{-1}[I'J']\mathbf{H}_P^{-\top}$			
5) TPHD [36]	1. one conic and one line 2. four 2D points	two real points: $\{M, N\}$	$[MN] = \mathbf{H}_{Q}^{-1}[I'J']\mathbf{H}_{Q}^{-1}$	$2*12*3^3 = 648^{\dagger}$		
	2. Tour 2D points	{1/1, 1/1}	$\mathbf{H} = \mathbf{H}_{Q}^{-1} \mathbf{H}_{S}^{'} \mathbf{H}_{P}^{}$			
6) SAP [50]	the infinity line and	\otimes	$\mathbf{H} = \mathbf{E}_{S} \mathbf{E}_{A} \mathbf{E}_{P}$	\otimes		
0) SAI [30]	two sets of orthogonal lines		$\mathbf{H} = \mathbf{E}_S \mathbf{E}_A \mathbf{E}_P$	⊗		
		tyria maal mainta.	$\{M,N\} \xrightarrow{\mathbf{H}_{S_1}} [\mp 1,0,1]^{\top}$			
7) SKS (Ours)	four 2D points	two real points: $\{M, N\}$	$\{\tilde{M}, \tilde{N}\} \stackrel{\mathbf{H}_{S_2}}{\longrightarrow} [\mp 1, 0, 1]^{\top}$	2*9=18		
		(===, = +)	$\mathbf{H} = \mathbf{H}_{S_2}^{-1} \mathbf{H}_K \mathbf{H}_{S_1}$			
		three real points: $\{M, N, P\} \xrightarrow{\mathbf{H}_{A_1}} [0, 0, 1]^\top, [0, 1, 1]^\top, [1, 0, 1]$				
8) ACA (Ours)	four 2D points	three real points: $\{M, N, P\}$	$\{\tilde{M}, \tilde{N}, \tilde{P}\} \xrightarrow{\mathbf{H}_{A_2}} [0, 0, 1]^{\top}, [0, 1, 1]^{\top}, [1, 0, 1]^{\top}$	2*7 = 14		
			$\mathbf{H} = \mathbf{H}_{A_2}^{-1} \mathbf{H}_C \mathbf{H}_{A_1}$			

2.4. Geometry based Homography Decomposition

In this subsection, we will introduce the evolution of the underlying idea of this paper step by step, based on three works of other researchers and our two previous works. These five works, together with this paper, gradually extend a geometry-based homography decomposition way to camera calibration with various 2D conic patterns and fast homography computation with 4 points. Different from all previous 4-point homography methods mentioned in Sec. 2.1, geometry based on methods do not construct the linear system, but utilize homography decomposition to solve the constraints provided by part of corresponding primitives.

Geometry based homography decomposition originally arises from the study of the circular points $\{I,J\}$ [73] [p. 33] and their dual conic [IJ]. Their projection points denoted by $\{\tilde{I},\tilde{J}\}$ on target plane only have 4 DOF, but can be used to compute the homography up to a similarity transformation by applying SVD [40] [p. 56],

$$[\tilde{I}\tilde{J}] = \mathbf{U}[IJ]\mathbf{U}^{\top},\tag{4}$$

where $[\tilde{I}\tilde{J}]$ denotes the dual conic of $\{\tilde{I},\tilde{J}\}$ and is a 3*3 symmetric matrix with 4 DOF; $\mathbf{U} = \mathbf{H}$ up to a similarity transformation. However, this method is seldom used in practice as the images of two circular points are difficult to be extracted unless the images of two circles [86] or the images of one circle and the infinity line can be obtained. The second row of Table 2 shows the characteristics of this method.

The work [37] summarized in the third row of Table 2 improves the above method, which utilizes the general-

ized eigenvalue decomposition (GED) of two confocal conics. Confocal conics is a special kind of pencil of conics [73] [p. 166-170]. Denote two confocal conics by $\mathbf{C}_1, \mathbf{C}_2$. The conic dual to the image of the circular points $[\tilde{I}\tilde{J}]$ can be obtained by utilizing the image of the confocal conics $\tilde{\mathbf{C}}_1, \tilde{\mathbf{C}}_2$ and their minimum generalized eigenvalue λ_{min} . However, this work extracting $[\tilde{I}\tilde{J}]$ from another special 2D pattern still does not extend homography computation to a pair of general imaginary points.

In one of our previous works [20] summarized in the fourth row of Table 2, a transformation mapping a pair of complex conjugate points on the source plane or the target plane to the circular points is constructed under different conics cases. Consequently, a pair of conics intersecting at the pair of complex conjugate points can be transformed to confocal conics or two circles under this projective transformation, which is similar to the above GED method [37]. The difference is that we consider the general conics case and GED is used twice. Thus, this method is called the extended generalized eigenvalue decomposition (EGED). The two intersection points (a pair of complex conjugate points) are denoted by $\{M, N\}$ on the source plane and $\{M, N\}$ on the target plane respectively, both of which can be transformed to the circular points $\{I, J\}$ by using SVD. Thus, the homography between a pair of conics and their images can be decomposed into three parts: first projective transformation \mathbf{H}_{O} , middle similarity transformation \mathbf{H}_{S} maintaining the invariance of the circular points, last projective transformation \mathbf{H}_P . However, this work decomposing homography of general conics pattern is still limited to the imaginary points.

A two real points based homography decomposition (TPHD) method is proposed in another of our previous works [36], which decomposes a homography into three parts: first projective transformation H_Q , middle hyperbolic similarity transformation \mathbf{H}_{S}^{\prime} , last projective transformation \mathbf{H}_{P} . See the fifth row of Table 2. The two real intersection points $\{M, N\}$ on the source plane and $\{\tilde{M}, \tilde{N}\}$ on the target plane can be transformed to the rectangular hyperbolic points $\{I', J'\}$ with the coordinates $[\mp 1, 1, 0]^{\top}$ under the projective transformation \mathbf{H}_{Q} and \mathbf{H}_{P} by using SVD, respectively. In that work, the problem of camera calibration with two open configurations is handled, which are a conic with a coplanar line and four 3D points. Although the proposed homography decomposition has utilized two real points, the computing efficiency is still not explored. For example, SVD with the computational complexity $O(3^3)$ has to be run twice. Moreover, the geometric interpretation of the derived hyperbolic similarity transformation \mathbf{H}_{S}' with four parameters is still unclear, which further limits its practical application.

Another well-known and meaningful homography decomposition method SAP [50], [40] [p. 42-43] is shown in the sixth row. A homography **H** is expressed by

$$\mathbf{H} = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v}^{\mathrm{T}} & v \end{bmatrix} = \mathbf{E}_{S} \mathbf{E}_{A} \mathbf{E}_{P}$$

$$= \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{K} & \mathbf{0} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{v}^{\mathrm{T}} & v \end{bmatrix},$$
(5)

where s, \mathbf{R} and \mathbf{t} denotes the scale, rotation and translation respectively in the called essence similarity transformation \mathbf{E}_S ; \mathbf{K} denoting the affine components is an upper-triangular matrix whose determinant is 1 in the essence affine transformation \mathbf{E}_A ; \mathbf{v}^T and v denote the projective components in the essence projective transformation \mathbf{E}_P ; $\mathbf{A} = s\mathbf{R}\mathbf{K}$ denotes affine component. Since SAP is not based on anchor points, its anchor points related FLOPs does not exist and is categorized into one individual category of primitives configuration in Sec. 1. Moreover, SAP is only suitable for a few of limited scenarios, requiring that the image of the infinity line is firstly available to recover its affine properties, followed by two sets of perpendicular lines to recover the metric properties.

Except the above previous methods, Table 2 also lists the pattern configuration, the chosen anchor points (AP), the decomposition formula, and AP-related computational amount (FLOPs) of our SKS and ACA. It is clear that SKS utilizes two similarity transformations \mathbf{H}_{S_1} and \mathbf{H}_{S_2} to manage $\{M,N\}$ and $\{\tilde{M},\tilde{N}\}$, respectively. Thus, SKS avoids the use of SVD and promotes the computational efficiency significantly. Furthermore, the proposed ACA method based on three anchor points computes the first and last subtransformations with less FLOPs.

2.5. Computational Amount and Runtime

Computational amount of an algorithm is typically evaluated by the number of floating-point operations (FLOPs).

Compared to the rough statistics of FLOPs (all four arithmetic operations are considered as 1 FLOPs) provided in the textbooks [34, 46], we follow the Livermore Loops Benchmark [59] to count FLOPs. Addition, subtraction and multiplication are considered as 1 FLOPs, while division is considered as 4 FLOPs.

In the mathematical society, the computational amount of an algorithm may be evaluated more deeply. Take matrix multiplication for an instance. Strassen presents that the usual method of computing product of two square matrices (requiring approximately $2n^3$ arithmetical operations) is not optimal [77]. Recently, AlphaTensor [32] is proposed to discover faster matrix multiplication algorithms utilizing reinforcement learning. In Strassen's method and AlphaTensor, for small-size matrices, the number of scalar multiplications decreases, but addition and subtraction operations actually increase. Therefore, it makes sense for an algorithm to avoid operations with high complexity, whether in theory or in practice. In this paper, of the four arithmetic operations, division is the most complicated and should be avoided as much as possible.

Besides FLOPs, the runtime of an algorithm on a single core is also influenced by many other hardware factors, such as data transfer and compiler optimization [68]. In simple terms, for a program running on a single core of a central processing unit (CPU), compiler optimizations significantly affect the runtime due to the serial scheme of data access and transfer [9, 5]. Conversely, for programs running on graphics processing units (GPUs), parallel data transfer and processing are more important than other factors. For a fair comparison in this paper, the homography computation algorithms are tested on CPU with different compiler optimization options and re-implemented on GPU under the same conditions.

3. DERIVATION OF SKS DECOMPOSITION

This section presents the similarity-kernel-similarity (SKS) decomposition for 4-point homography computation. Specifically, the overview of decomposition chain is firstly introduced in Sec. 3.1. Then, the detailed derivation of the first and last similarity transformations is given in Sec. 3.2, followed by the derivation of the middle kernel transformation shown in Sec. 3.3. The computational amount (FLOPs) of SKS with four points is investigated in Sec. 3.4. The n-point (n>4) homography computation problem is discussed in Sec. 3.5.

3.1. Overview of Decomposition Chain

The first step of SKS is to utilize two points, such as $\{M_1, N_1\}$ and $\{M_2, N_2\}$, to compute a unique similarity transformation on their own plane respectively. See the left part of Fig. 3. The similarity transformation deduced by these two points on π_1 and π_2 are denoted by \mathbf{H}_{S_1} and \mathbf{H}_{S_2} , respectively. Under the similarity transformation, the two points will be transformed to two special points with the homogeneous coordi-

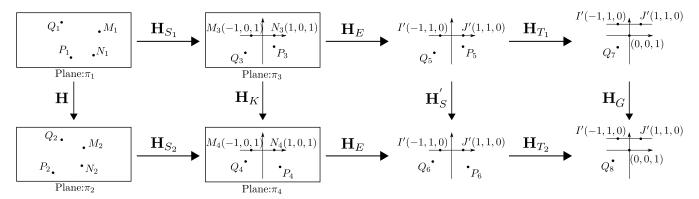


Fig. 3. Sub-transformations in SKS decomposition.

nates $[\mp 1, 0, 1]^{\top}$ (we default that M and N are in the negative and positive directions of the x-axis respectively). This can be done as both the 2D similarity transformation and two points have 4 DOF. The chosen two points are called two anchor points (TAP), because they play a role on normalizing all planar points to a new rectangular plane coordinate system. The two special points $[\mp 1, 0, 1]^{\top}$ are thus called canonical TAP. The remaining unknown component in the homography is called kernel transformation \mathbf{H}_K . Under \mathbf{H}_K , the canonical TAP $[\mp 1, 0, 1]^{\top}$ should be invariant. Moreover, the other two pairs of corresponding points can be utilized to compute \mathbf{H}_K which also has 4 DOF. The detailed computation process of \mathbf{H}_K , including further decomposing it to other transformations (the right part of Fig. 3), will be explained in Sec. 3.3.

3.2. Similarity Transformation based on Two Anchor Points

Take the two 2D points $\{M_1, N_1\}$ to calculate the similarity transformation \mathbf{H}_{S_1} as an example. The other similarity transformation \mathbf{H}_{S_2} can be similarly calculated from $\{M_2, N_2\}$.

The midpoint O_1 of the line segment M_1N_1 is given by

$$O_1.x = 0.5 * (M_1.x + N_1.x),$$

 $O_1.y = 0.5 * (M_1.y + N_1.y),$
(6)

Then the vector $\overrightarrow{O_1N_1}$ is calculated by

$$\overrightarrow{O_1N_1}.x = N_1.x - O_1.x,$$

$$\overrightarrow{O_1N_1}.y = N_1.y - O_1.y.$$
(7)

Instead of directly computing the scale parameter s_{S_1} and the rotation parameter θ_{S_1} in \mathbf{H}_{S_1} , \mathbf{H}_{S_1} can be expressed more easily without the radical and trigonometric operations

$$\mathbf{H}_{S_1} \equiv \begin{bmatrix} \overrightarrow{O_1 N_1} x & \overrightarrow{O_1 N_1} y \\ -\overrightarrow{O_1 N_1} y & \overrightarrow{O_1 N_1} x \end{bmatrix} \begin{bmatrix} 1 & -O_1 x \\ 1 & -O_1 y \\ 1 \end{bmatrix},$$

$$(8)$$

with

$$f_{S_1} = (\overrightarrow{O_1 N_1}.x)^2 + (\overrightarrow{O_1 N_1}.y)^2,$$
 (9)

and we also have

$$\mathbf{H}_{S_1}^{-1} \equiv \begin{bmatrix} \overrightarrow{O_1 N_1}.x & -\overrightarrow{O_1 N_1}.y & O_1.x \\ \overrightarrow{O_1 N_1}.y & \overrightarrow{O_1 N_1}.x & O_1.y \\ & & 1 \end{bmatrix}. \tag{10}$$

Under \mathbf{H}_{S_1} , all the 2D points on the plane π_1 are normalized according to the translation, the scale and the rotation transformations defined by $\{M_1, N_1\}$. $\{M_1, N_1\}$ as the TAP on π_1 will be transformed to $\{M_3, N_3\}$ (with the coordinates $[\mp 1, 0, 1]^{\top}$) as the canonical TAP on π_3 . Similarly, \mathbf{H}_{S_2} in Fig. 3 can also be calculated based on the TAP $\{M_2, N_2\}$ on the plane π_2 . The remaining unknown component \mathbf{H}_K in \mathbf{H} will be introduced in next subsection.

3.3. Kernel Transformation

For \mathbf{H}_K , a unique property is the invariance of the canonical TAP $[\mp 1, 0, 1]^{\top}$. We give the following proposition to obtain the expression of \mathbf{H}_K .

Proposition 1. The canonical TAP $[\mp 1, 0, 1]^{\top}$ are fixed under the 2D projective transformation \mathbf{H}_K , if and only if \mathbf{H}_K is expressed by

$$\mathbf{H}_K \equiv \begin{bmatrix} a_K & u_K & b_K \\ & 1 \\ b_K & v_K & a_K \end{bmatrix} . \tag{11}$$

The proof is shown in Appendix A.

Let the transformed P_1 and Q_1 under \mathbf{H}_{S_1} on π_3 be P_3 and Q_3 respectively. And the transformed P_2 and Q_2 under \mathbf{H}_{S_2} on π_4 are denoted by P_4 and Q_4 respectively. \mathbf{H}_K with 4 DOF can be calculated by these two pairs of corresponding points $\{P_3 \xrightarrow{\mathbf{H}_K} P_4\}$ and $\{Q_3 \xrightarrow{\mathbf{H}_K} Q_4\}$. \mathbf{H}_K is further associated with the hyperbolic similarity transformation [36] as follows.

We notice that the points $[\mp 1,0,1]^{\top}$ can be transformed to the rectangular hyperbolic points $\{I^{'},J^{'}\}$ (= $[\mp 1,1,0]^{\top}$)

under an elementary transformation \mathbf{H}_E , which is expressed by

$$\mathbf{H}_E = \begin{bmatrix} 1 & & \\ & & 1 \\ & 1 & \end{bmatrix}. \tag{12}$$

Then, the hyperbolic similarity transformation \mathbf{H}_S' , under which $\{I',J'\}$ are fixed, is further decomposed as follows for computing convenience.

$$\mathbf{H}_{S}^{'} = \mathbf{H}_{T_{2}}^{-1} \mathbf{H}_{G} \mathbf{H}_{T_{1}}$$

$$= \begin{bmatrix} 1 & P_{6}.x \\ 1 & P_{6}.y \\ 1 & 1 \end{bmatrix} \begin{bmatrix} a_{G} & b_{G} \\ b_{G} & a_{G} \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & -P_{5}.x \\ 1 & -P_{5}.y \\ & & 1 \end{bmatrix},$$
(13)

where P_5 and P_6 are the transformed points of P_3 and P_4 under \mathbf{H}_E , respectively; \mathbf{H}_{T_1} and \mathbf{H}_{T_1} denote the 2D translation transformations; \mathbf{H}_G denoting hyperbolic rotation and scale can be computed by solving the following linear equations in two variables:

$$Q_8.x = a_G * Q_7.x + b_G * Q_7.y,$$

$$Q_8.y = b_G * Q_7.x + a_G * Q_7.y,$$
(14)

where Q_7 is the transformed point of Q_3 under $\mathbf{H}_{T_2} * \mathbf{H}_E$, and Q_8 is the transformed point of Q_4 under $\mathbf{H}_{T_2} * \mathbf{H}_E$.

Fig. 3 depicts each sub-transformation of the proposed SKS decomposition. Obviously, \mathbf{H}_{S_1} and \mathbf{H}_{S_2} do not influence the shape of the quadrangles on two planes. Therefore the transformation $\mathbf{H}_K = \mathbf{H}_E^{-1}\mathbf{H}_S'\mathbf{H}_E$ results in the real projective distortion. We call \mathbf{H}_K the kernel transformation, in which four variables are calculated by

$$a_K = a_G, \quad b_K = b_G,$$

 $u_K = P_6.x - a_K * P_5.x - b_K * P_5.y,$ (15)
 $v_K = P_6.y - b_K * P_5.x - a_K * P_5.y.$

Finally, the homography **H** is expressed by the above *similarity-kernel-similarity* (SKS) decomposition form:

$$\mathbf{H} = \mathbf{H}_{S_2}^{-1} \mathbf{H}_K \mathbf{H}_{S_1} = \mathbf{H}_{S_2}^{-1} \mathbf{H}_E^{-1} \mathbf{H}_S' \mathbf{H}_E \mathbf{H}_{S_1}.$$
(16)

The distinction between SKS (Eq. 16) and TPHD [36] is that SKS uses a simpler and more efficient expression $\mathbf{H}_{S_1}\mathbf{H}_E$ to transform the two anchor points to the rectangular hyperbolic points $\{I',J'\}$. While in TPHD, it needs to repeat the calculation of SVD for a 3*3 matrix (a projective transformation) twice, which is not ignorable compared with a small number of arithmetic calculations of SKS. Moreover, the kernel transformation \mathbf{H}_K in Eq. 16 denoting the projective distortion has more algebraic and geometrical properties than \mathbf{H}_S' .

3.4. FLOPs Analysis of SKS

In this subsection, we analyze floating-point operations (FLOPs) of the proposed SKS method under 4 points pattern. The counting method of FLOPs has been introduced in Sec. 2.5. Owing to our simple expression and calculation of a homography in Eq. 16, only a small number of FLOPs is required for each step in SKS. The specific FLOPs statistics of the proposed SKS method with 4 points is further divided into two situations according to different practical applications.

i) The homography between two images \mathcal{I}_1 and \mathcal{I}_2 , i.e., $\{\mathbf{H}\colon \mathcal{I}_1\to\mathcal{I}_2\}$. This situation often occurs in image stitching or visual simultaneous localization and mapping (SLAM), where a sequence of images is captured by a moving camera. In this situation, each sub-transformation of the SKS shown in Fig. 3 should be calculated. We call this situation the complete SKS. FLOPs of each manipulation step in the complete SKS is shown in Table 3. For example, in the complete SKS, the computation of \mathbf{H}_S' involves four sub-steps, which are computing Q_7 , computing Q_8 , solving $\{a_K,b_K\}$ and computing the multiplication of the three sub-transformations. The sum FLOPs of these steps in SKS is 157. If normalization of homography is required (generally based on the lower right element h_{33} of \mathbf{H}), there are 1 additional division and 8 multiplication operations (12 FLOPs).

ii) The homography between an object plane ${\cal O}$ and one its image \mathcal{I} , i.e., $\{\mathbf{H}: \mathcal{O} \rightarrow \mathcal{I}\}$. This situation often occurs in camera calibration and metric rectification with a known planar pattern. In this situation, the pattern (e.g., chessboard or QR code) with known geometry is captured to obtain the homography between the object plane and one its image. Changing the object coordinate system into a new one determined by two pre-selected anchor points in advance, we no longer need to calculate the similarity transformation \mathbf{H}_{S_1} . The homography up to a similarity can be utilized for the computation of intrinsic parameters in camera calibration, or recover planar metric information [40] [p. 57]. This situation is thus called the simplified SK. Most applications under this situation (e.g., camera calibration and object detection) do not care about the change of the origin and rotation of the object coordinate system. When the real translation in the relative pose needs to be calculated, we only need to multiply the final result by a pre-known scale factor. Under this situation, the homography $\bar{\mathbf{H}}$ to be solved satisfies

$$\bar{\mathbf{H}} = \mathbf{H}\mathbf{H}_{S_1}^{-1} = \mathbf{H}_{S_2}^{-1}\mathbf{H}_K.$$
 (17)

In addition, other calculation steps are also simplified, such as the calculation of \mathbf{H}_{T_1} and the operation of matrix multiplication. The FLOPs of each step in this simplified SK decomposition is also listed in Table 3. Compared to the complete SKS, the simplified SK reduces the calculation amount by about half.

FLOPs of other 4-point homography computation methods, including NDLT-SVD [40], HO-SVD [38], GPT-LU [4]

Table 3. FLOPs of each step in the proposed SKS method with 4 points. * denotes the operation of matrix multiplication. After obtaining the homography up to a scale, normalization can be done with extra 12 FLOPs at last. Detailed FLOPs notations can be seen in the released source code.

	\mathbf{H}_{S}	S_1	\mathbf{H}_{S}	S_2	Н	T_1	Н	T_2		Н	$[_{K}$		*			
Step	$\overrightarrow{O_1N_1}$	f_{S_1}	$\overrightarrow{O_2N_2}$	f_{S_2}	P_3	P_5	P_4	P_6	Q_7	Q_8	a_K b_K	$u_K \ v_K$	$\mathbf{H}_{S_2}^{-1} \! * \! \mathbf{H}_K$	$*\mathbf{H}_{S_1}$	Sum	w. Norm.
SKS (complete)	6	3	6	3	8	6	8	6	14	14	16	8	20	39	157	169
SK (simplified)	0	0	6	3	0	0	8	6	0	14	8	8	20	0	73	85

and RHO-GE [17], have been given in Table 1. Compared to NDLT-SVD, HO-SVD, GPT-LU and RHO-GE, SKS represents effective speedup about 162x, 11x, 12x and 1.3x in terms of FLOPs respectively. These comparison of FLOPs are basically consistent with the experimental results of rumtime on CPU and GPU shown in Sec. 6.1 and Sec. 6.4 respectively.

3.5. Discussion of n-Point Homography Computation

In this subsection, we briefly discuss the problem of n-Point homography computation. NDLT-SVD [40] [p. 109] and HO-SVD [38] can handle n-point configuration, but at the cost of a lot of redundancy in their 4-point homography computation. Although SKS can be adjusted to process n-point pattern to bring speed improvements, the stratified computation strategy is unchanged. Thus, each sub-transformation in SKS computed with a part of n point correspondences is sub-optimal. As a result, for general situations where speed is not the first factor, we still recommend using the HO-SVD method to compute n-point homography. As pointed out in Sec. 2.2, n-point homography computation is not needed in the standard RANSAC [40] [p. 123]. Although there exists the method [27] using n-point homography computation to replace final global optimization algorithms, the computational cost of the RANSAC framework mainly depends on the calculation and verification of 4-point homography, especially facing high outlier ratios.

4. AFFINE-CORE-AFFINE (ACA) DECOMPOSITION

This section presents an improved affine-core-affine (ACA) decomposition for 4-point homography computation. Specifically, other choices of canonical two anchor points is introduced in Sec. 4.1. Then, the detailed derivation of the first and last affine transformations based on three anchor points is given in Sec. 4.2, followed by the derivation of the middle core transformation shown in Sec. 4.3. The computation amount (FLOPs) of ACA with 4 points is demonstrated in Sec. 4.4. Further polynomial expression of each element of homography in given in Sec. 4.5.

4.1. Other Choices of Canonical Two Anchor Points

In the derivation of the SKS decomposition, we set $[\mp 1, 0, 1]^{\top}$ to be the canonical TAP to compute two similarity transformations on the source and target planes. However, the

canonical TAP can also be selected at other positions. Consequently, the geometric meaning and algebraic calculation of the new homography decomposition will be a little different, especially for the kernel transformation \mathbf{H}_K . Denote the similarity transformation mapping $[\mp 1,0,1]^{\top}$ to the new TAP by \mathbf{H}_{S_3} . The SKS decomposition based on new canonical TAP is expressed by

$$\mathbf{H} = \underbrace{\mathbf{H}_{S_{2}}^{-1} \mathbf{H}_{S_{3}}^{-1}}_{\text{new } \mathbf{H}_{S_{2}}^{-1}} * \underbrace{\mathbf{H}_{S_{3}} \mathbf{H}_{K} \mathbf{H}_{S_{3}}^{-1}}_{\text{new } \mathbf{H}_{K}} * \underbrace{\mathbf{H}_{S_{3}} \mathbf{H}_{S_{1}}}_{\text{new } \mathbf{H}_{S_{1}}}.$$
 (18)

For example, if the canonical TAP are set to be $[0, \mp 1, 1]^{\top}$, most of sub-transformations in the decomposition chain shown in Fig. 3 remain the same except \mathbf{H}_E and \mathbf{H}_K . \mathbf{H}_E will play the role of transforming $[0, \mp 1, 1]^{\top}$ to the rectangular hyperbolic points $\{I', J'\}$. The new \mathbf{H}_K will be expressed by

$$\mathbf{H}_{K} \equiv \begin{bmatrix} -1 \\ 1 \\ & 1 \end{bmatrix} \begin{bmatrix} a_{K} & u_{K} & b_{K} \\ & 1 \\ b_{K} & v_{K} & a_{K} \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ & 1 \end{bmatrix}$$

$$\equiv \begin{bmatrix} 1 \\ -u_{K} & a_{K} & b_{K} \\ -v_{K} & b_{K} & a_{K} \end{bmatrix}.$$
(19)

Another mapping choice is to set the canonical TAP be $\{[0,0,1]^{\top},[1,0,1]^{\top}\}$. Therefore, \mathbf{H}_{S_3} is given by

$$\mathbf{H}_{S_3} \equiv \begin{bmatrix} 0.5 & 0.5 \\ & 0.5 \\ & & 1 \end{bmatrix}, \tag{20}$$

and the new \mathbf{H}_K is given by

$$\mathbf{H}_{K} \equiv \begin{bmatrix} a_{K} + b_{K} & u_{K} + v_{K} \\ & 1 \\ 2*b_{K} & 2*v_{K} & a_{K} - b_{K} \end{bmatrix}$$

$$\equiv \begin{bmatrix} 1 & u_{K}' \\ & v_{K}' \\ 1 - a_{K}' & b_{K}' & a_{K}' \end{bmatrix}, \tag{21}$$

where the four parameters are re-named and re-scaled as fol-

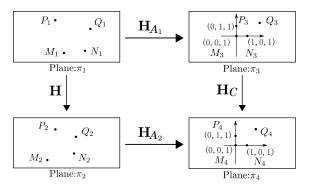


Fig. 4. Sub-transformations in ACA decomposition.

lows

$$a'_{K} = \frac{a_{K} - b_{K}}{a_{K} + b_{K}}, \quad b'_{K} = \frac{2 * b_{K}}{a_{K} + b_{K}},$$

$$u'_{K} = \frac{u_{K} + v_{K}}{a_{K} + b_{K}}, \quad v'_{K} = \frac{2 * v_{K}}{a_{K} + b_{K}}.$$
(22)

4.2. Affine Transformation based on Three Anchor Points

Compared with varying the position of the canonical two anchor points, choosing three anchor points (HAP) to compute the first and last transformations will bring more significant changes. Let the canonical HAP be $\{[0,0,1]^{\top},[1,0,1]^{\top},[0,1,1]^{\top}\}$. Three points on the source plane π_1 and the target plane π_2 , such as $\{M_1,N_1,P_1\}$ and $\{M_2,N_2,P_2\}$, can be utilized to compute a unique affine transformation with 6 DOF on their own plane respectively. Take $\{M_1,N_1,P_1\}$ to calculate the affine transformation \mathbf{H}_{A_1} on π_1 as an example. It is not difficult to obtain

$$\mathbf{H}_{A_{1}} \equiv \begin{bmatrix} \overrightarrow{M_{1}P_{1}}.y & -\overrightarrow{M_{1}P_{1}}.x \\ -\overrightarrow{M_{1}N_{1}}.y & \overrightarrow{M_{1}N_{1}}.x & \\ & & f_{A_{1}} \end{bmatrix} \begin{bmatrix} 1 & -M_{1}.x \\ 1 & -M_{1}.y \\ & 1 \end{bmatrix},$$
(23)

with

$$f_{A_1} = \overrightarrow{M_1 N_1}.x * \overrightarrow{M_1 P_1}.y - \overrightarrow{M_1 N_1}.y * \overrightarrow{M_1 P_1}.x, \quad (24)$$

and

Similarly, the affine transformation \mathbf{H}_{A_2} on π_2 can also be computed.

4.3. Core Transformation

Similar to SKS, we can obtain a homography decomposition with three components after obtaining the two affine transformations on the source and target planes. As shown in Fig. 4, under \mathbf{H}_{A_1} and \mathbf{H}_{A_2} , the plane π_1 and π_2 are transformed to two new planes π_3 and π_4 , respectively. The projective transformation between π_3 and π_4 is denoted by \mathbf{H}_C . Because of

the invariance of the canonical HAP, the following proposition is reached to obtain the expression of \mathbf{H}_C .

Proposition 4. The canonical three anchor points $\{[0,0,1]^{\top},[1,0,1]^{\top},[0,1,1]^{\top}\}$ are fixed under the 2D projective transformation \mathbf{H}_C , if and only if \mathbf{H}_C is expressed by

$$\mathbf{H}_C \equiv \begin{bmatrix} a_C \\ b_C \\ a_C - 1 & b_C - 1 & 1 \end{bmatrix}. \tag{26}$$

The proof is similar to it in Appendix A and omitted.

Since \mathbf{H}_C imposes the projective distortion between two affine normalization planes, we call \mathbf{H}_C the core transformation. Therefore, a homography \mathbf{H} is expressed by the affine-core-affine (ACA) decomposition form:

$$\mathbf{H} = \mathbf{H}_{A_2}^{-1} \mathbf{H}_C \mathbf{H}_{A_1}. \tag{27}$$

Obviously, \mathbf{H}_C with 2 DOF can be computed from $\{Q_3,Q_4\}$, which are transformed from the fourth point correspondence $\{Q_1,Q_2\}$ under \mathbf{H}_{A_1} and \mathbf{H}_{A_2} respectively. Specifically, the point Q_3 denoted by $[Q_3.x,Q_3.y,f_{A_1}]^{\top}$ is obtained by

$$\begin{bmatrix}
Q_3.x \\
Q_3.y \\
f_{A_1}
\end{bmatrix} = \mathbf{H}_{A_1} * \begin{bmatrix}
Q_1.x \\
Q_1.y \\
1
\end{bmatrix} \\
= \begin{bmatrix}
\overline{M_1Q_1}.x * \overline{M_1P_1}.y - \overline{M_1Q_1}.y * \overline{M_1P_1}.x \\
\overline{M_1N_1}.x * \overline{M_1Q_1}.y - \overline{M_1N_1}.y * \overline{M_1Q_1}.x
\end{bmatrix}, (28)$$

where $Q_3.x$ and $Q_3.y$ denotes the coordinates of Q_3 up to a scale f_{A_1} for computational convenience.

Similarly, the point Q_4 denoted by $[Q_4.x, Q_4.y, f_{A_2}]^{\top}$ can be obtained. As Q_4 is the projection of Q_3 under the core transformation \mathbf{H}_C , a_C and b_C in Eq. 26 are solved as:

$$a_C = \frac{t_1 * Q4.x}{t_2 * Q3.x}, \quad b_C = \frac{t_1 * Q4.y}{t_2 * Q3.y},$$
 (29)

with

$$t_1 = f_{A_1} - Q_3.x - Q_3.y, \quad t_2 = f_{A_2} - Q_4.x - Q_4.y.$$
 (30)

To avoid the division operations in the above equations, we change the expression of \mathbf{H}_C up to a scale as follows,

$$\mathbf{H}_C \equiv \begin{bmatrix} c_{11} & & \\ & c_{22} & \\ c_{11} - c_{33} & c_{22} - c_{33} & c_{33} \end{bmatrix}, \tag{31}$$

with

$$c_{11} = t_1 * Q_3.y * Q_4.x,$$

$$c_{22} = t_1 * Q_3.x * Q_4.y,$$

$$c_{33} = t_2 * Q_3.x * Q_3.y.$$
(32)

Table 4. FLOPs of each step in ACA with 4 points. * denotes the operation of matrix multiplication. Detailed FLOPs notation can be seen in the released source code. Without normalization, all other steps to compute the homography up to a scale do not involve any division operations.

Step		\mathbf{H}_{A_1}			\mathbf{H}_{A_2}		\mathbf{H}_C				*						
зкр	$\overrightarrow{M_1N_1}$	$\overrightarrow{M_1P_1}$	f_{A_1}	$\overrightarrow{M_2N_2}$	$\overrightarrow{M_2P_2}$	f_{A_2}	$\overrightarrow{M_1N_1}$	$\overrightarrow{M_2N_2}$	Q_3	Q_4	t_1	t_2	c_{11}, c_{22}, c_{33}	${\bf H}_{A_2}^{-1}*{\bf H}_C$	$*\mathbf{H}_{A_1}$	Sum	w. Norm.
ACA	2	2	3	2	2	3	2	2	6	6	2	2	8	10	33	85	97

Substituting the above expression of \mathbf{H}_C into Eq. 27, the homography \mathbf{H} up to a scale is finally computed without any division operations.

4.4. FLOPs Analysis of ACA

Similar to the computation analysis of SKS shown in Sec. 3.4, we give the number of FLOPs in each step of ACA shown in Table 4. FLOPs of both \mathbf{H}_{A_1} and \mathbf{H}_{A_2} are 7 according to Eq. 23 and Eq. 24. FLOPs of calculating \mathbf{H}_C is 28 according to Eq. 28, Eq. 30 and Eq. 32. The multiplication operation of these sub-transformations costs 43 FLOPs. Without consideration of the final normalization (which has 12 FLOPs), ACA can compute a homography up to a scale with only 85 FLOPs and do not involve any division operations. Compared to the previous 4-point homography methods NDLT-SVD, HOSVD, GPT-LU and RHO-GE shown in Table 1, the total 97 FLOPs of ACA is about 0.35%, 5.4%, 5.0% and 43% of their FLOPs, respectively. These theoretical speedups are also reflected by the experimental results running on CPU and GPU shown in Sec. 6.

Here we only analyze the FLOPs of ACA for the mapping between four general points on source and target planes. When the quadrilateral on the source plane is a square or rectangle (often seen in deep homography pipelines), the calculation process is further simplified, which will be illustrated in Sec. 5.3.

4.5. Polynomial Expression of Homography

This subsection describes another algebraic advantage of the proposed ACA decomposition. Owing to the extremely simple expression of each sub-transformation, we can represent each element of a homography by the input variables (16 coordinates provided by four point correspondences) in polynomial form.

See Eq. 23 and Eq. 25. It is clear that \mathbf{H}_{A_1} and $\mathbf{H}_{A_2}^{-1}$ can be directly expressed with the input 16 coordinates. Substituting \mathbf{H}_C in Eq. 26 into the ACA decomposition in Eq. 27, after some manipulations, the column vector form of \mathbf{H} is given by

$$\begin{bmatrix} h_{11} \\ h_{21} \\ h_{31} \\ h_{31} \\ h_{11} \\ h_{22} \\ h_{33} \\ h_{33} \end{bmatrix} = \begin{bmatrix} N_2 \cdot x (P_1 \cdot y - M_1 \cdot y) & P_2 \cdot x (M_1 \cdot y - N_1 \cdot y) & M_2 \cdot x (N_1 \cdot y - P_1 \cdot y) \\ N_2 \cdot y (P_1 \cdot y - M_1 \cdot y) & P_2 \cdot y (M_1 \cdot y - N_1 \cdot y) & M_2 \cdot y (N_1 \cdot y - P_1 \cdot y) \\ (P_1 \cdot y - M_1 \cdot y) & (M_1 \cdot y - N_1 \cdot y) & (N_1 \cdot y - P_1 \cdot y) \\ N_2 \cdot x (M_1 \cdot x - P_1 \cdot x) & P_2 \cdot x (N_1 \cdot x - M_1 \cdot x) & M_2 \cdot x (P_1 \cdot x - N_1 \cdot x) \\ N_2 \cdot x (M_1 \cdot x - P_1 \cdot x) & P_2 \cdot y (N_1 \cdot x - M_1 \cdot x) & M_2 \cdot x (P_1 \cdot x - N_1 \cdot x) \\ N_2 \cdot x \cdot f_1 & P_2 \cdot x \cdot f_2 & M_2 \cdot x \cdot f_{A_1} \\ N_2 \cdot y \cdot s \cdot f_1 & P_2 \cdot y \cdot s \cdot f_2 & M_2 \cdot x \cdot s \cdot f_{A_1} \\ f_1 & f_2 & f_{A_1} \end{bmatrix}$$
 (33)

where f_1 and f_2 , as well as f_{A_1} in Eq. 24, are second degree polynomials of input coordinates, which are expressed by,

$$f_1 = P_1.x * M_1.y - P_1.y * M_1.x,$$

$$f_2 = M_1.x * N_1.y - M_1.y * N_1.x.$$
(34)

Meanwhile, it can be seen from Eq. 28 and Eq. 30 that $Q_3.x$, $Q_3.y$, $Q_4.x$, $Q_4.y$, t_1 and t_2 are also second degree polynomials. Therefore, c_{11} , c_{22} , c_{33} in Eq. 32 are sixth degree polynomials. Thus it is straightforward that each element in the vector form of a homography is a polynomial, with 8, 8, 7, 8, 8, 7, 9, 9, 8 degree respectively, which is expressed by

$$\mathbf{H} = \begin{bmatrix} \mathcal{F}_{11}^8 & \mathcal{F}_{12}^8 & \mathcal{F}_{13}^9 \\ \mathcal{F}_{21}^8 & \mathcal{F}_{22}^8 & \mathcal{F}_{23}^9 \\ \mathcal{F}_{31}^7 & \mathcal{F}_{32}^7 & \mathcal{F}_{33}^8 \end{bmatrix}, \tag{35}$$

where \mathcal{F}^i denotes an *i*-th degree polynomial.

To our knowledge, this is the first time to obtain the polynomial expression of a homography directly with the input 16 coordinates. An interesting observation is that if there is a division operation in the calculation process, it may be impossible to obtain the polynomial expression of a homography.

5. EXTENSION AND APPLICATIONS

The above two sections describe the proposed SKS and ACA methods respectively, in both of which 4-point homography is computed with clear geometric meaning and fast calculation. In this section, we will introduce their further extension and applications. Specifically, how SKS and ACA extend the existing SAP decomposition is illustrated in Sec. 5.1. The SKS and ACA decomposition for a 2D affine transformation is given in Sec. 5.2. The tensorized ACA (TensorACA) used in end-to-end deep homography estimation is proposed in Sec. 5.3.

5.1. Extending SAP Decomposition

Consider our SKS decomposition shown in Eq. 18 and Eq. 21. Assuming \mathbf{H}_{S_1} is an identity matrix, one equivalent decomposition is derived by us,

$$\mathbf{H} = \mathbf{H}_{S_{2}}^{-1} \mathbf{H}_{K}$$

$$= \mathbf{H}_{S_{2}}^{-1} \begin{bmatrix} 1 & u'_{K} \\ v'_{K} \\ & 1 \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 \\ 1 - a'_{K} & b'_{K} & a'_{K} \end{bmatrix}, (36)$$

where it can be seen that SAP is just one special form of the proposed SKS decomposition.

Furthermore, if combining the essence similarity and affine transformations into a general affine transformation, the SAP decomposition will have one fixed form,

$$\mathbf{H} = \begin{bmatrix} h_{11} - h_{31} \frac{h_{13}}{h_{33}} & h_{12} - h_{32} \frac{h_{13}}{h_{33}} & \frac{h_{13}}{h_{33}} \\ h_{21} - h_{31} \frac{h_{23}}{h_{33}} & h_{22} - h_{32} \frac{h_{23}}{h_{33}} & \frac{h_{23}}{h_{33}} \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$
(37)

where this expression is also equivalent to the proposed ACA decomposition, if we assume \mathbf{H}_{A_1} is an identity matrix and make an adequate transformation.

The fixed forms of Eq. 5 and Eq. 37 greatly limit the practical use of the SAP decomposition. However, the proposed SKS and ACA decomposition can be regarded as an extension of SAP, enjoying the notable merits of dealing with general primitives and fast calculation.

5.2. Decomposition of 2D Affine Transformation

This subsection describes how to decompose an affine transformation using the proposed SKS and ACA methods. For SKS, the kernel transformation \mathbf{H}_K will be affine if the homography becomes an affine transformation. Thus the parameters \mathbf{b}_K and \mathbf{v}_K denoting the projective components in Eq. 11 (or the parameters $1-\mathbf{a}_K'$ and \mathbf{b}_K' in Eq. 21) should be zero. Then an affine transformation denoted by \mathbf{H}_A can be decomposed as

$$\mathbf{H}_A = \mathbf{H}_{S_0}^{-1} \mathbf{H}_K \mathbf{H}_{S_1}, \tag{38}$$

with a new form of \mathbf{H}_K

$$\mathbf{H}_K \equiv \left[\begin{array}{cc} 1 & u_K \\ & c_K \\ & & 1 \end{array} \right]. \tag{39}$$

Then the upper left 2*2 sub-matrix of \mathbf{H}_A (which denotes a 2D linear transformation) can be given by

$$\mathbf{H}_{A}^{2*2} = (\mathbf{H}_{S_{2}}^{-1})^{2*2} \mathbf{H}_{K}^{2*2} \mathbf{H}_{S_{1}}^{2*2}, \tag{40}$$

where this expression is actually equivalent to twice using QR factorization [34] [p. 246].

For the ACA decomposition, it is obvious that the core transformation \mathbf{H}_C will be an identity matrix if the homography degenerates to an affine transformation with 6 DOF. Thus the affine transformation \mathbf{H}_A is expressed by

$$\mathbf{H}_{A} = \mathbf{H}_{A_{2}}^{-1} \mathbf{H}_{A_{1}} = \mathbf{H}_{A_{2}}^{-1} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & 1 \end{bmatrix} \mathbf{H}_{A_{1}}$$

$$= \begin{bmatrix} N_{2}.x & P_{2}.x & M_{2}.x \\ N_{2}.y & P_{2}.y & M_{2}.y \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} N_{1}.x & P_{1}.x & M_{1}.x \\ N_{1}.y & P_{1}.y & M_{1}.y \\ 1 & 1 & 1 \end{bmatrix}^{-1}.$$

$$(41)$$

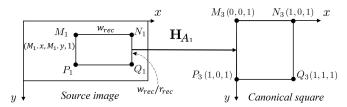


Fig. 5. Affine transformation mapping a rectangle (with four parameters) in source image to the canonical square.

The second row in the above equation is actually the solution of \mathbf{H}_A directly utilizing the constraints of the three point correspondences [42] [Eq. 10.73 in p. 237]. However, both the first and second matrices in the second row are projective transformations, which should not appear in an affine decomposition. Another advantage of the proposed ACA decomposition in the first row is that affine transformations can be computed in the same way as projective transformations. Moreover, referring to Table 4, it is not difficult to deduce that the FLOPs of $\mathbf{H}_A = \mathbf{H}_{A_2}^{-1} \mathbf{H}_{A_1}$ is only 33, which is obviously less than the FLOPs of the second row. Compared to the OpenCV's function getAffineTransform which solves a linear system of inhomogeneous equations with 6 variables utilizing the LU factorization (~1000 FLOPs), the ACA decomposition for solving an affine transformation with three point correspondences is much more efficient.

5.3. Tensorized ACA

This subsection illustrates how to tensorize the proposed ACA decomposition for being used in deep homography pipelines. The tensorized ACA is called TensorACA, which inherits the nomination of the previous method TensorDLT [63] internally implementing the tensorized GPT-LU [4]. Although ACA only uses basic arithmetic operations which are naturally differential, two adjustments should be made to promote running efficiency.

The first adjustment is that when writing Python algorithms under deep learning frameworks, such as PyTorch and TensorFlow, it is better to take advantage of tensor structures (such as vectors, 2D matrices and 3D tensors) and operations. By doing this, although the number of FLOPs of the adjusted algorithms increases, the runtime decreases as the cost of function calling and memory access is reduced, especially when running on GPU.

The second adjustment is for the simple quadrangle selected in source image. All previous 4-point offsets based deep homography methods compute the homography mapping a square [63, 47, 75, 66, 21] or rectangle [65] [54] in source image to a general quadrangle in target image. However, none of these methods explore how to simplify the 4-point homography computation under this rectangle scenario. For a square or rectangle on source image, we find that 4-point homography computation can be simplified as follows.

Following the image coordinate system, we assume M_1

is the upper left vertex of an arbitrary rectangle and N_1 , Q_1 , P_1 are other vertices in a clockwise order. As shown in Fig. 5, the affine transformation \mathbf{H}_{A_1} induced by three anchor points $\{M_1, N_1, P_1\}$ transforms the rectangle to the canonical square. Denote the rectangle by four common parameters $\{M_1.x, M_1.y, w_{rec}, r_{rec}\}$, where w_{rec} and r_{rec} are the width and the aspect ratio respectively. The affine transformation \mathbf{H}_{A_1} in Eq. 23 is simplified to

$$\mathbf{H}_{A_1} \equiv \begin{bmatrix} 1 & & & \\ & r_{rec} & \\ & & 1 \end{bmatrix} \begin{bmatrix} 1 & & -M_1.x \\ & 1 & -M_1.y \\ & & w_{rec} \end{bmatrix}, \quad (42)$$

and the point Q_3 in Eq. 28 is directly obtained without any calculation,

$$\begin{bmatrix} Q_3.x \\ Q_3.y \\ f_{A_1} \end{bmatrix} \equiv \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{43}$$

Substituting into Eq. 30 and Eq. 32, the core transformation \mathbf{H}_C is expressed by

$$\mathbf{H}_{C} \equiv \begin{bmatrix} Q_{4}.x & & & \\ Q_{4}.y & & & \\ Q_{4}.x+t_{2} & Q_{4}.y+t_{2} & -t_{2} \end{bmatrix}. \tag{44}$$

After some manipulations, the homography ${\bf H}$ between a rectangle and its corresponding quadrangle is expressed by

$$\mathbf{H} \equiv \begin{bmatrix} N_{2}.x & P_{2}.x & M_{2}.x \\ N_{2}.y & P_{2}.y & M_{2}.y \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} Q_{4}.x & -Q_{4}.xM_{1}.x \\ & r_{rec}Q_{4}.y & -r_{rec}Q_{4}.yM_{1}.y \\ t_{2} & r_{rec}t_{2} & -t_{2}(M_{1}.x + r_{rec}M_{1}.y + w_{rec}) \end{bmatrix},$$

$$(45)$$

where the expression will be further simplified when the rectangle is a square (i.e., $r_{rec} = 1$).

It is clear that all calculation steps related to \mathbf{H}_{A_1} in Table 3 are removed and matrix multiplication is also simplified. FLOPs of homography computation for a rectangle and square on source plane is only 47 and 44, respectively. As a result, the runtime of TensorACA applied in deep homography pipeline will further decrease. The complete steps of the tensorized ACA for a rectangle are illustrated in Algorithm 1 with only 15 vector operations. It is also straightforward to see that the first two columns of homography constraining intrinsic parameters used in camera calibration [92] are only related to the aspect ratio of the source rectangle, rather than its position and width.

It is mentioned in Sec. 2.1 that RHO-GE without pivoting chosen may not be robust. This problem becomes more apparent when facing the four vertices of a rectangle in source image. When the four point correspondences are given in an improper order, RHO-GE will fail to compute the correct homography. A detailed discussion of failure cases is given in Appendix B.

Algorithm 1: Pseudo-code of TensorACA for a rectangle

```
Input: rectangle's upper left vertex M_1, width w_{rec} and aspect
           ratio r_{rec} in source image;
           four projection corners M_2, N_2, P_2, Q_2 in target image
           with homogeneous vector representations \mathbf{m}_2, \mathbf{n}_2, \mathbf{p}_2, \mathbf{q}_2.
Output: 2D homography matrix \mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3] up to a scale.
1. Calculate two difference vectors:
       \mathbf{d}_1 = [N_2.x - M_2.x; P_2.x - M_2.x; Q_2.x - M_2.x], 
 \mathbf{d}_2 = [N_2.y - M_2.y; P_2.y - M_2.y; Q_2.y - M_2.y].
2. Calculate cross-product:
       \mathbf{c} = \text{Cross}(\mathbf{d}_1, \mathbf{d}_2).
                                                \% \mathbf{c} = [Q_4.x; Q_4.y; -f_{A_2}]
3. Calculate a temporary vector by scaling \mathbf{m}_2:
        \mathbf{b} = \operatorname{Sum}(\mathbf{c}) \cdot * \mathbf{m}_2. \qquad \% \operatorname{Sum}(\mathbf{c}) = -t_2
4. Compute the first two columns of H:
        \mathbf{h}_1 = Q_4.x. * \mathbf{n}_2 - \mathbf{b},
       \mathbf{h}_2 = r_{rec} \cdot * (Q_4 \cdot y \cdot * \mathbf{p}_2 - \mathbf{b}).
                                                                 \% r_{rec} = 1 for a square
5. Compute the last column of H:
       \mathbf{h}_3 = w_{rec} \cdot * \mathbf{b} - M_1 \cdot x \cdot * \mathbf{h}_1 - M_1 \cdot y \cdot * \mathbf{h}_2
Notation: '.*' represents a vector multiplied by a scalar.
```

6. EXPERIMENTS

This section shows the experimental results. As the proposed SKS and ACA deal with the task of homography computation under the minimal condition, our experiments focus on verifying the runtime of 4-point homography solvers under various scenarios. Specifically, Sec. 6.1 tests the runtime of single homography on CPU. Sec. 6.2 further gives the changes of runtime after integrating the proposed method into the traditional feature-based RANSAC pipelines on synthetic and real datasets. Sec. 6.3 tests the runtime of different 4-point homography solvers implemented by Python, as well as the performance in deep homography pipelines. Sec. 6.4 further gives the runtime of parallel computation of multiple 4-point homographies on GPU utilizing the CUDA toolkit [8]. Detailed hardware and software configurations are listed below.

CPU: Intel i7-10700 (8 physical cores)
GPU: NVIDIA RTX-3090 (10496 CUDA cores)
Memory: 64G

Software Environment (C++): Win10 & Visual Studio 2019 & OpenCV 4.5.1 & CUDA 11.8

Software Environment (Python): Ubuntu 20.04 & Py-Torch 1.13.0 & CUDA 11.1

All the shown results can be reproduced with our source codes, which have been released in https://github.com/cscvlab/SKS-Homography.

6.1. Runtime of Single Homography on CPU

In this subsection, four simulated point correspondences are used to test the runtime of homography solvers on a single core of CPU. Except the proposed SKS and ACA methods, there are other four algorithms participating in the comparison of 4-point homography computation. The implementation functions and files in OpenCV 4.5.1 of these four algorithms are listed below:

Table 5. Average runtime (μ s) of different 4-point homography algorithms running on a single CPU core under three compiler optimization options. The suffix (*) and (**) denote the implementations with single-precision and double-precision floating-point numbers, respectively. The 1st and 2nd fastest results are highlighted in red and blue, respectively.

Compiler		Previous M	Our Methods					
Optimization	NDLT-SVD(**) [40]	HO-SVD(**) [38]	GPT-LU(**) [4]	RHO-GE(*) [17]	SKS(*)	SKS(**)	ACA(*)	ACA(**)
Od	13.6	13.5	0.885	0.169	0.0595	0.0901	0.0393	0.0561
O1	12.7	12.2	0.735	0.0327	0.0253	0.0263	0.0146	0.0178
O2	12.5	12.2	0.732	0.0287	0.0252	0.0256	0.0145	0.0171

- NDLT-SVD [40]: runKernel in 'fundam.cpp'.
- HO-SVD [38]: homographyHO in 'ippe.cpp'.
- GPT-LU [4]: getPerspectiveTransform in 'imgwarp.cpp'.
- GE-RHO [17]: hFuncRefC in 'rho.cpp'.

Similar to the function *hFuncRefC* in the RHO-GE method, we implement SKS and ACA without any externally defined functions or data structures. Meanwhile, two versions of our methods are implemented using single-precision floating point numbers (as RHO-GE uses) and double-precision floating point numbers (as NDLT-SVD, HO-SVD and GPT-LU use), respectively. These two implementations are denoted with the suffix (*) and (**), respectively.

The release program runs on a single core of CPU with the compiler optimization options Od, O1 and O2, respectively [9]. Optimization options O1 and O2 set a combination of optimizations including vectorization [5] [p. 18-83] that generate minimum size code and optimize code for maximum speed, respectively. Vectorization is actually a special case of single instruction multiple data (SIMD), operating on multiple data in parallel [5][p. 3-34]. On the contrary, the Od option turns off all compiler optimizations and executes program code one by one.

The average runtime is shown in Table 5, where we run all algorithms 10M times. Several observations are achieved. First, the runtime difference of using single-precision and double-precision floating point numbers is not significant, especially when enabling compiler optimization (O1 or O2). Second, the influence of compiler optimization for the algorithms differs obviously. For the first three algorithms from left to right based on OpenCV data structures and functions, the influence of enabling compiler optimization is trivial. However, for the last five algorithms (RHO-GE and ours) including only arithmetic operations, doing this brings several times speedup. Of these algorithms, RHO-GE benefits the most from enabling compiler optimizations as it is pointed out that 'This neatly fits in half of a vector register file with 16 4-lane registers, a common configuration in most modern architectures.' [17]. Third, compared to the three robust methods (NDLT-SVD, HO-SVD and GPT-LU), SKS(**) and ACA(**) represents $\{488x, 477x, 29x\}$ and $\{731x, 713x,$ 43x} respectively under O2 optimization. These values of practical speedup are significantly larger than the theoretical FLOPs comparison shown in Table 1. This is because the implementations of the three robust methods include more or less conditional branch judgments, data copy or exchange, OpenCV data structures, etc., which severely influence the speed. This phenomenon will be alleviated when we run these algorithms on GPU shown in Sec. 6.4 as all external data structures and functions are discarded except the CUDA toolkit. Fourth, SKS(*) is obviously faster than RHO-GE, while ACA(*) takes only half runtime of RHO-GE under O2 optimization, representing \sim 70M calculations per second. The runtime performance of SKS and ACA is basically consistent with their FLOPs. This speedup is a significant improvement for estimating homography between images, whether in the traditional feature-based RANSAC pipelines tested in Sec. 6.2, or in deep homography pipelines tested in Sec. 6.3.

6.2. Runtime in Feature-based RANSAC Pipelines

In this subsection, SKS and ACA are integrated as a plug-in module into the traditional feature-based RANSAC pipelines to estimate homography between two images. The main purpose of this test is to observe how our homography decomposition methods accelerate practical applications. For the comprehensive comparison of feature-based RANSAC pipelines, interested readers could refer to the literature focusing on feature points extraction and matching, such as the benchmark [45].

The first experiment is conducted on the widely used synthetic MS-COCO dataset [51]. Following the early work [29], 5000 images are randomly chosen from MS-COCO's test set. Each image is resized to grayscale 640*480, and a pair of 256*256 image patches is generated by a synthetic homography, which matches four corners of a source square patch to a target quadrangle patch with four disturbed corners. The whole process is roughly divided into five stages, which are tested separately. The first two stages accomplish the feature points extraction and coarse matching. In the first stage, the traditional SIFT [56] and ORB [72] algorithms, as well as the deep learning method SuperPoint [30], are used to extract feature points whose maximum number is set to 1000. In the second stage, coarse descriptor matching based on 2 nearest neighbors search and ratio test (with the threshold τ =0.8)

Table 6. Average runtime (μ s) of five stages for six combinations of feature points extraction and matching frameworks on MS-COCO dataset. The 1st and 2nd fastest results of computing 4-point homography are highlighted in red and blue, respectively. The notations ' \mathcal{R} ' and ' \mathcal{M} ' denote the default solver in RANSAC and MAGSAC++ respectively.

Stage		SIFT [56] & RANSAC [33]	SIFT [56] & MAGSAC++ [14]	ORB [72] & RANSAC [33]	ORB [72] & MAGSAC++ [14]	SuperPoint [30] & RANSAC [33]	SuperPoints [30] & MAGSAC++ [14]	
1) Feature Poi	nts Extraction	19	9.7K	4.	71K	12.2K		
2) Descriptor	2) Descriptor Matching w. Ratio Test		680	1.	20K	17.2K		
	NDLT-SVD [40] (\mathcal{R})	109	76.8	204	154	397	314	
3) Loop of	HO-SVD [38]	79.7	56.3	144	98.7	279	215	
4-point	-point $GPT-LU[4](\mathcal{M})$		4.72	14.7	8.46	29.8	16.3	
Homography	Iomography RHO-GE [17]		0.217	0.503	0.400	1.24	0.771	
Computation	SKS (Ours)	0.238	0.191	0.422	0.331	0.887	0.676	
	ACA (Ours)	0.174	0.144	0.302	0.281	0.690	0.517	
4) Loop of Ve	rification	4.34	25.6	6.41	42.0	19.9	153	
5) Global Opt	imization	101	54.3	82.0	79.4	108	103	
	Inlier Number	204	202	137	135	224	225	
Metrics	Inlier Ratio	0.919	0.912	0.846	0.833	0.731	0.735	
Menies	Homography Loop	5.68	3.51	9.97	7.27	18.8	15.4	
	Total Runtime w. R/M	20.6K	20.5K	6.20K	6.04K	29.9K	29.7K	

is performed. Image pairs with matching feature points less than 8 are removed. Then, RANSAC [33] and the SOTA MAGSAC++ [14] frameworks, covering the third to fifth stages, are adopted to remove outliers. The third stage is the loop computation of 4-point homography, where the previous four solvers and our two solvers participate in the comparison. Notice that the default 4-point homography solver in RANSAC and MAGSAC++ is NDLT-SVD and GPT-LU respectively. The fourth stage includes loop of 4-point subset sample, inliers verification, and other possible operations (e.g., shape check of 4-point, homography scoring and local optimization in MAGSAC++). The number of loops here is calculated based on inlier ratio with a probability 99% that all sampled four points are inliers. At the same time, the maximum number of loops is set to 1000, roughly corresponding to the minimum inlier ratio of 26%. The fifth stage is the global optimization with all inliers, in which the Levenberg-Marquardt (LM) algorithm [60] and a series of manipulations are adopted in RANSAC and MAGSAC++ respectively. All the above methods are implemented based on OpenCV's C++ procedures and tested on CPU, except SuperPoint which is implemented using PyTorch and tested on GPU. The average runtime of six combinations of three feature points methods and two outlier-removal frameworks is shown in Table 6.

Overall speaking, similar to the results shown in Table 5, SKS and ACA significantly reduce runtime of the stage of 4-point homography computation. However, compared to the default robust solvers NDLT-SVD and GPT-LU, the speedup brought by SKS or ACA for the whole process is not obvious as the first two stages of feature points extraction and coarse matching are relatively time-consuming. Another reason is that the synthetic image pairs have high inlier ratios (shown in the second row of 'Metrics') and small numbers of homogra-

phy loops (shown in the third row of 'Metrics'). For example, the maximum speedup happens in the combination of the real-time feature point method ORB and the classical RANSAC framework. Replacing the default solver NDLT-SVD with our ACA will save about 204 microseconds, which is accumulated from 7.27 loops of 4-point homography computation. Compared to the original process with the total runtime 6.20K, the revised process represents about 3% speedup and 97% runtime ratio. Another minor factor affecting the runtime is the number of involved points in each stage. Take the runtime of the fifth stage as an instance. Both in the RANSAC and MAGSAC++ frameworks, SuperPoint is the most time-consuming feature point method as it has the largest number of inlier points (shown in the first row of 'Metrics').

Most image pairs in the Warped MS-COCO dataset are ideal in terms of lighting, texture, and no foreground, which result in their generally high inlier ratios. Thus the second experiment is conducted on one more difficult real dataset: dynamic scenes [47]. From the real dynamic dataset, we randomly select 33 videos, containing 2024 image pairs. The combination 'ORB & RANSAC' adopted by ORB-SLAM3 [22] is performed to compare this real dataset and MS-COCO. The runtime performance is evaluated through the processes integrating ACA and the default 4-point homography solver NDLT-SVD. The distribution of inlier ratio and runtime ratio (percentage of image pairs for ten levels of ratio) are shown in Fig. 6. It is clear that the dynamic real dataset has significantly lower inlier ratio and runtime ratio than MS-COCO. For the other two combinations containing the RANSAC framework, their runtime ratios will increase due to the increasing runtime of feature extraction and numbers of loops. For the combinations containing the MAGSAC++ framework, the speedup brought by ACA will

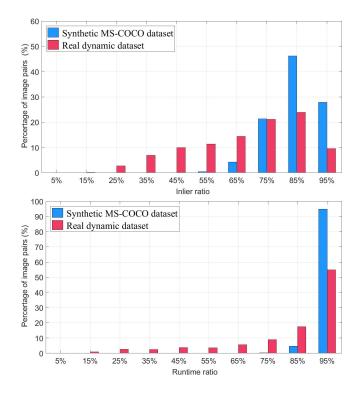


Fig. 6. Comparison between the synthetic MS-COCO dataset and the dynamic real dataset. Upper: inlier ratio; Lower: runtime ratio.

be further insignificant, since the default 4-point homography solver GPT-LU already saves most of the runtime of the third stage.

6.3. Runtime in Deep Homography Pipelines

In this section, we test the runtime of deep homography networks using either the default 4-point homography solver TensorDLT or the proposed TensorACA. Different from the traditional feature-based RANSAC pipelines, deep homography pipelines do not suffer the inconsistent runtime mainly caused by the undetermined number of loops. TensorDLT (which actually is the tensorized GPT-LU) is implemented differently in recent deep homography networks, including UDHN [63], CA-UDHN [54] and 1-scale IDHN [21]. In the official implementation using PyTorch, UDHN and CA-UDHN use the same code (represented by TensorDLT-1) to compute a 4-point homography, although the resolutions of their input gray images are 128*128 and 560*315 respectively. IDHN uses another implementation code (represented by TensorDLT-2) and its input images are color with the resolution 128*128. TensorACA is implemented in two ways, which are called TensorACA-vanilla and TensorACA-rect. TensorACA-vanilla is written using Python statements to imitate our C++ program, which only contains arithmetic operations of floating-point numbers. TensorACA-rect is implemented (as shown in Algo. 1) for the mapping from a source rectangle to a target quadrangle. Meanwhile, we also

use Python statements to re-implement the RHO-GE method named TensorGE to participate in the comparison. For TensorGE, we manually select an order of four points analyzed in Appendix. B to avoid division by zero. The two methods NDLT-SVD and HO-SVD are not re-implemented using Python as it is pointed out that 'taking the gradients in SVD has high time complexity and has practical implementation issues.' [63].

The experiments are conducted to test the inference time of the individual modules of five 4-point homography solvers, the original networks and the revised networks both on CPU and GPU, with the PyTorch library. The average runtime of 50K trials for modules and 5K trials for networks are shown in Table 7. Several observations are illustrated below.

First, the individual module of each 4-point homography solver runs about twice as slowly on GPU than on CPU. However, the whole network runs faster on GPU than on CPU, since GPU is suitable for large-scale parallel computing, such as convolution operations.

Second, there exists obvious difference of runtime for two ways of tensorizing GPT-LU (TensorDLT-1 and TensorDLT-2). For Python programs, it is not strange because that different operation ways for high-dimensional arrays will bring about large speed differences. Moreover, the performance of 4-point homography solver in individual module test is a little different from integrating it in networks. This is because GPU executions are commonly asynchronous, but the stable runtime test requires synchronization [7].

Third, TensorGE is the slowest in the tests of both individual module and the three integrated networks, because it is inefficient for Python statements to execute floating-point arithmetic operations. TensorACA-vanilla performs much better than TensorGE due to less FLOPs and parameters. Compared with TensorDLT-2, both of the two TensorACA methods run faster in individual module and the three integrated networks. Compared with TensorDLT-1, TensorACA-rect runs about 18% and 15% faster in individual module test on CPU and GPU respectively.

Fourth, with TensorACA-rect as the new 4-point homography solver, the revised UDHN, CA-UDHN and IDHN can reduce 1.3%, 0.34%, and 17% runtime respectively on GPU (shown in the right part of the last row of Table 7), compared to their original networks with TensorDLT inside (shown in the right part of the third and fourth rows of Table 7). Both the absolute time reduction and largest percentage speedup occurs the revised IDHN, as the adopted TensorDLT-2 is not efficient and its network iteratively predicts 4-point offsets six times to further refine the homography.

Compared with the CPU runtime of these 4-point homography solvers implemented by C++ (shown in Table 5 and Table 6), the performance of these solvers' Python procedures drops dramatically. For example, TensorDLT-1, TensorGE and TensorACA-vanilla converting C++ statements to Python statements are about $172(\approx 126.2/0.732)$,

Table 7. Average runtime (ms) of homography computation from 4-point offsets in deep homography inference. In UDHN and CA-UDHN, the 4-point homography solver TensorDLT-1 is executed only once, while in IDHN, its solver TensorDLT-2 is iteratively executed six times.

Methods		Runr	ning on CPU		Running on GPU				
Wethods	Module	UDHN [63]	CA-UDHN [54]	IDHN [21]	Module	UDHN [63]	CA-UDHN [54]	IDHN [21]	
TensorDLT-1 (GPT-LU [4])	0.1262	18.52	1229.26		0.2509	4.726	13.52		
TensorDLT-2 (GPT-LU [4])	0.4458			46.86	0.8286			23.84	
TensorGE (RHO-GE [17])	1.372	20.04	1231.93	53.35	2.522	7.227	15.72	34.00	
TensorACA-vanilla (Ours)	0.2976	18.77	1229.44	45.97	0.6972	5.186	13.82	23.16	
TensorACA-rect (Ours)	0.1034	18.50	1229.23	44.72	0.2145	4.666	13.47	19.91	

 $47.8\text{K}(\approx 1372/0.0287)$ and $17.4\text{K}(\approx 297.6/0.0171)$ times slower, respectively. TensorDLT-1 gains the fewest runtime drop, as its Python implementation is suitable for tensorization (only involving stack of coefficient matrix and solving an inhomogeneous linear system with the form $\mathbf{A}_{8*8}\mathbf{x}_{8*1} = \mathbf{b}_{8*1}$). Thus, to avoid runtime drop of Python programs and pursue fast running in practice, one possible solution is to implement networks directly with the C++ CUDA toolkit².

6.4. Runtime of Parallel Homographies on GPU

In this subsection, we test the runtime of multiple homographies computation in parallel on GPU. This is meaningful for both the feature-based RANSAC pipelines and the deep homography pipelines. In the traditional feature-based RANSAC pipelines, multiple 4-point subsets can be sampled simultaneously from extracted point correspondences, followed by parallel computation of multiple homographies in the next stage. In the deep homography pipelines implemented by Python in Sec. 6.3, one and a number of (batch size) 4-point homographies are calculated in network inference and training, respectively. Therefore, compared to sequential execution on CPU and parallel execution with Python on GPU, direct implementation of multiple homographies computation in parallel using CUDA on GPU will greatly reduce the runtime. Specifically, each 4-point homography is assigned to one thread of GPU for computation and the program statements will be sequentially executed by a GPU CUDA core. We sample $\{1, 10, 100, 1000, 10K, 100K, 1M\}$ 4-point subsets respectively, followed by sending them to GPU for parallel homography computation. These sampling numbers correspond to the approximate outlier proportions $\{0\%, 22\%, 54\%, 74\%, 85\%, 92\%, 95\%\}$ respectively, with a probability 99% to ensure at least one 4-point sample only consists of inliers. For running on GPU, the previous four algorithms and our two methods are re-implemented using CUDA. There are only double-precision floating-point numbers adopted and all OpenCV data structures are discarded in all implementations.

Table 8. Average runtime (µs) of computing multiple 4-point homographies in parallel on GPU. All methods are reimplemented using CUDA. Here the listed sampling numbers correspond to different outlier ratios with a probability 99% that all four points are inliers in standard RANSAC.

Samp. Num.		Our Methods				
/ Outlier Ratio	NDLT-SVD	HO-SVD	GPT-LU	RHO-GE	SKS	ACA
/ Outner Rano	[40]	[38]	[4]	[17]	SKS	ACA
1 / 0%	469	55.1	29.6	4.69	4.20	3.11
$10 / \sim 22\%$	617	65.5	30.7	4.69	4.26	3.16
100 / ~ 54%	794	79.3	31.0	4.74	4.31	3.19
1K / ~74%	807	80.8	30.8	6.17	4.83	3.20
10K / ~85%	1.35K	135	50.7	10.1	7.45	5.26
100K / ∼92%	15.0K	1.19K	845	66.7	49.9	29.3
1M / ~95%	151K	11.2K	8.39K	589	436	245

The average runtime of computing multiple 4-point homographies in parallel on GPU are shown in Table 8, where we run the six algorithms 10K to 1M times, according to roughly the same total execution time (10 seconds). Overall speaking, for all sample numbers (or outlier ratios), the proposed ACA algorithm is fastest among the compared methods, followed by the proposed SKS algorithm. The comparison of RHO-GE, SKS and ACA are basically similar to running on CPU (shown in Table 5). SKS is always obviously faster than RHO-GE, and ACA is about twice as fast as RHO-GE. For NDLT-SVD, HO-SVD and GPT-LU re-implemented without external data structures running on GPU, their performances are more consistent with comparison of FLOPs than them on CPU. Taking the fastest ACA method as the basis for comparison, the ratios of FLOPs, CPU runtime (with 'O2' compiler optimization) and GPU runtime (with 100K homographies in parallel) of all algorithms are depicted in Fig. 7. It is observed that the ratios of GPU runtime are more consistent with ratios of FLOPs for all algorithms than the ratios of CPU runtime.

Another observation is also expected that the total runtime of all algorithms for small numbers ($\leq 10 \text{K}$) of homographies in Table 8 increases slightly with the increase of the numbers. Meanwhile, the average runtime of computing one homography is almost inversely proportional to the small sampling number. This is because the parallel computation of small numbers of homographies don't trigger all 10496

²One famous example we familiar is the 3D implicit representation neural network Instant-NGP [61], which can be trained within a number of seconds and accomplish new view synthesis inference in real time.

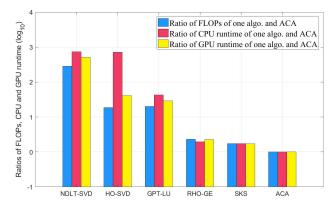


Fig. 7. Ratios (log_{10}) of FLOPs, CPU runtime and GPU runtime of one algorithm and the proposed ACA as a comparison basis. Since the programs of NDLT-SVD, HO-SVD and GPT-LU running on CPU include conditional branch judgments, data copy or exchange, OpenCV data structures, their ratios of CPU runtime deviate the ratios of FLOPs much more than the ratios of GPU runtime.

CUDA cores and the slight increase of the runtime is mainly due to more data accesses. Thus we further conduct an experiment with more sampling numbers ranging from 1K to 20K. The results of these six methods are shown in Fig. 8. It can be seen that the stationary point of GPU runtime happens with the sampling number 10K, which is the approximately same to the number of CUDA cores (10496). When the sampling number is higher than 10K, the runtime of computing one homography becomes stable and the total runtime of computing multiple homographies begins to increase linearly with the increase of the number.

The efficiency difference for 4-point homography computation on CPU and GPU can be seen from Table 5 and Table 8. When the number of homographies needed to compute is less than 100 (outlier ratio is lower than $\sim\!54\%$), the GPU runtime for one homography is actually larger than the CPU runtime. When the number of homographies in parallel computation increases above 100, the efficiency advantage of GPU starts to show. For example, the average runtime of one homography in the parallel computation of 1M homographies on GPU is only 0.245ns (shown in the last row of Table 8), representing $\sim\!70$ x speedup relative to the CPU runtime 17.1ns (shown in the last row of Table 5).

The efficiency difference between C++ program with the CUDA toolkit and Python program with the PyTorch library both on GPU can be seen from Table 7 and Table 8. Take the runtime of ACA as an instance. For deep homography inference on GPU, computing a 4-point homography with Python statements costs $697.2\mu s$, as shown in the penultimate row of Table 7. However, computing a 4-point homography using CUDA only costs $3.11\mu s$, as shown in the last column of Table 8. This 224x speedup demonstrates the speed advantage of directly implementing deep homography networks with the C++ CUDA toolkit.

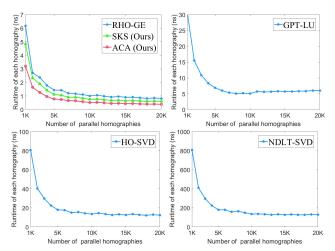


Fig. 8. Average runtime (ns) of computing one homography on GPU. The stationary point of runtime happens with the sampling number 10K, which is approximately equal to the number of CUDA cores (10496) of NVIDIA RTX3090 GPU.

7. CONCLUSION AND FUTURE WORK

In this paper, two novel homography decomposition methods with high computational efficiency and clear geometrical interpretation are presented. In particular, the kernel transformation in the SKS decomposition with only four parameters indicates the minimal expression of projective distortion between two normalized images. The ACA decomposition further significantly improves the computational efficiency, requiring only 97 FLOPs (85 FLOPs and no division operations for homographies up to a scale). Experiments conducted on a consumer desktop computer show that our ACA algorithm can be run about 70M times on CPU and 4G times on GPU. This huge computational advantage enables the proposed algorithm to be integrated into both the traditional feature-based RANSAC pipeline and the deep homography pipeline to replace their default 4-point homography solvers. In addition, the proposed anchor points based method is a unified solution for planar primitives, which has been illustrated in the extension of SAP decomposition and the computation of 2D affine transformation.

Practical applications of other advantages, such as the direct expression of each element in homography via a polynomial of input coordinates and geometric parameterization, are worth pursuing in future work. We are also willing to explore the significance of SKS and ACA in other vision problems, such as pose estimation and image stitching.

Appendix

A. PROOF OF EXPRESSION OF KERNEL TRANSFORMATION

Assume that the kernel transformation is expressed by

$$\mathbf{H}_K = \begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix}, \tag{A.1}$$

where \mathbf{H}_K should satisfy

$$\begin{bmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{bmatrix} \begin{bmatrix} \mp 1 \\ 0 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \mp 1 \\ 0 \\ 1 \end{bmatrix}. \tag{A.2}$$

Thus we have

$$-k_{21} + k_{23} = 0, \quad k_{21} + k_{23} = 0,$$

 $k_{11} + k_{13} = k_{31} + k_{33}, \quad -k_{11} + k_{13} = k_{31} - k_{33}.$ (A.3)

Solving the above equations, we have

$$k_{21} = 0$$
, $k_{23} = 0$, $k_{11} = k_{33}$, $k_{13} = k_{31}$. (A.4)

Removing the homogeneous equality (let $k_{22}=1$), the kernel transformation can be expressed by

$$\mathbf{H}_K \equiv \begin{bmatrix} a_K & u_K & b_K \\ & 1 \\ b_K & v_K & a_K \end{bmatrix} . \tag{A.5}$$

B. FAILURE CASES OF RHO-GE

In RHO-GE [17], since no pivoting is adopted, division by zero will occur for some specific configurations of four points on source plane. Still denoting the four points on source plane and their correspondences on target plane by $\{M_1,N_1,P_1,Q_1\}$ and $\{M_2,N_2,P_2,Q_2\}$ respectively. In RHO-GE, the order of four points is fixed and the third point P_1 is used to be subtracted for the first time of row operation. After some row interchange operations, the first six columns of \mathbf{A}_{8*9} in Eq. 3 is expressed by

s expressed by
$$\begin{bmatrix} M_1.x - P_1.x & M_1.y - P_1.y & 1 & 0 & 0 & 0\\ N_1.x - P_1.x & N_1.y - P_1.y & 1 & 0 & 0 & 0\\ P_1.x & P_1.y & 1 & 0 & 0 & 0\\ Q_1.x - P_1.x & Q_1.y - P_1.y & 1 & 0 & 0 & 0\\ 0 & 0 & 0 & M_1.x - P_1.x & M_1.y - P_1.y & 1\\ 0 & 0 & 0 & N_1.x - P_1.x & N_1.y - P_1.y & 1\\ 0 & 0 & 0 & P_1.x & P_1.y & 1\\ 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - P_1.y & 1\\ 0 & 0 & 0 & 0 & 0 & Q_1.x - P_1.x & Q_1.y - Q_1.y -$$

In the subsequent row reduction operations, two fixed items will be selected as the divisors, which are expressed by

$$d_1 = M_1.x - P_1.x,$$

$$d_2 = (M_1.x - P_1.x) * (N_1.y - P_1.y) - (M_1.y - P_1.y) * (N_1.x - P_1.x).$$
(B.7)

Obviously, the first point M_1 and the third point P_1 on the source plane cannot be on a vertical line, otherwise the divisor d_1 will be zero. However, the implementation of RHO-GE in OpenCV (the fastest competitor among the four previous methods in our C++ experiments), does not contain program statements (which would degrade its speed performance) to exclude such failure cases. For a rectangle on source plane, among 24 orders of 4 vertices, 8 orders will divide by 0. Therefore, a manual ordering of four vertices is done by us to avoid this failure case in our deep learning experiments. For the other divisor d_2 , although it cannot be zero unless the first three points are collinear or two of them are same (both of which are degenerate configurations), d_2 may cause a decrease in numerical precision when it is close to zero.

C. REFERENCES

- [1] https://github.com/Tyrant1337/Lu-Decomposition-in-Parallel.
- [2] https://github.com/xintaoding/CSAC.
- [3] https://en.wikipedia.org/wiki/FLOPS.
- [4] OpenCV's function: getPerspectiveTransform. https://docs.opencv.org/4.5.1/da/d54/group__imgproc__transform.html. 2, 3, 4, 5, 9, 13, 15, 16, 18
- [5] Intel 64 and IA-32 architectures optimization reference manual. https://www.intel.com/content/ www/us/en/developer/articles/technical/ intel-sdm.html. 7, 15
- [6] http://www.vision.caltech.edu/bouguetj/ calib_doc/.
- [7] Cuda c++ programming guide. https://docs.nvidia. com/cuda/cuda-c-programming-guide/index. html. 17
- [8] Cuda toolkit. https://developer.nvidia.com/ cuda-toolkit. 14
- [9] Microsoft c++, c, and assembler documentation. https://docs.microsoft.com/en-us/cpp/ build/reference/o-options-optimize-code? view=msvc-160. 7, 15
- [10] S. Baker, A. Datta, and T. Kanade. Parameterizing homographies. *Robotics Institute, Pittsburgh, PA, Technical Report CMU-RI-TR-06-11*, 2006. 1, 5
- [11] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International journal of computer vision* (*IJCV*), 56(3):221–255, 2004. 5
- [12] A. Bandera and J. M. Pérez. Mean shift based clustering of hough domain for fast line segment detection. *Pattern Recognition Letters*, 27(6):578–586, 2006.
- [13] D. Barath, J. Matas, and J. Noskova. MAGSAC: marginalizing sample consensus. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 4
- [14] D. Barath, J. Noskova, M. Ivashechkin, and J. Matas. MAGSAC++, a fast, reliable and accurate robust estimator. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020. 2, 4, 5, 16

- [15] C. A. Basca, M. Talos, and R. Brad. Randomized hough transform for ellipse detection with result clustering. In *International Conference on Computer As A Tool*, pages 1397–1400, 2006
- [16] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision and Image Understanding* (CVIU), 110(3):404–417, 2006. 1, 4
- [17] H. Bazargani, O. Bilaniuk, and R. Laganiere. A fast and robust homography scheme for real-time planar target detection. *Journal of Real-Time Image Processing*, 15(4):739–758, 2018. 2, 3, 4, 5, 10, 15, 16, 18, 20
- [18] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. Dsac-differentiable ransac for camera localization. In *IEEE Conference on Computer Vision* and Pattern Recognition (CVPR), volume 3, 2017.
- [19] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The euroc micro aerial vehicle datasets. 2016.
- [20] S. Cai, Z. Zhao, L. Huang, and Y. Liu. Camera calibration with enclosing ellipses by an extended application of generalized eigenvalue decomposition. *Machine Vision and Applications* (MVA), 24(3):513–520, 2013. 2, 6
- [21] S.-Y. Cao, J. Hu, Z. Sheng, and H.-L. Shen. Iterative deep homography estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 1, 5, 13, 17, 18
- [22] J. J. G. R. J. M. M. M. Carlos Campos, Richard Elvira and J. D. Tardós. Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam. *IEEE Transactions* on Robotics, 37(6):1874–1890, 2021. 1, 5, 16
- [23] C.-H. Chang, C.-N. Chou, and E. Y. Chang. Clkn: Cascaded lucas-kanade networks for image alignment. In *IEEE/CVF* Conference on Computer Vision and Pattern Recognition (CVPR), 2017. 5
- [24] Q. Chen, H. Wu, and T. Wada. Camera calibration with two arbitrary coplanar circles. In *European Conference on Computer Vision (ECCV)*, 2004.
- [25] O. Chum and J. Matas. Matching with prosac progressive sample consensus. In *IEEE Conference on Computer Vision* and Pattern Recognition (CVPR), 2005. 4
- [26] O. Chum and J. Matas. Matching with prosac-progressive sample consensus. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 220–226. IEEE, 2005.
- [27] O. Chum, J. Matas, and J. Kittler. Locally optimized ransac. In Joint Pattern Recognition Symposium, 2003. 10
- [28] T. Collins and A. Bartoli. Infinitesimal plane-based pose estimation. *International Journal of Computer Vision (IJCV)*, 109(3):252–286, 2014. 1, 4, 5
- [29] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. arXiv preprint: 1606.03798, 2016. 1, 5, 15
- [30] D. DeTone, T. Malisiewicz, and A. Rabinovich. Superpoint: Self-supervised interest point detection and description. In Conference on Computer Vision and Pattern Recognition Workshop (CVPRW) on Deep Learning for Visual SLAM, 2018. 1, 4, 15, 16
- [31] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *IEEE Conference on Computer Vision and Pattern Recognition* (*CVPR*), pages 2790–2797, 2009.

- [32] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022. 7
- [33] M. Fischler and R. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 2, 4, 16
- [34] G. H. Golub and C. F. V. Loan. *Matrix computations, 4th ed.* The Johns Hopkins University Press, 2013. 1, 2, 4, 6, 7, 13
- [35] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(4):722–732, 2010. 2
- [36] J. Guo, S. Cai, Z. Wu, and Y. Liu. A versatile homography computation method based on two real points. *Image and Vision Computing (IVC)*, 64(C):23–33, 2017. 2, 6, 7, 8, 9
- [37] P. Gurdjos, J. S. Kim, and I. S. Kweon. Euclidean structure from confocal conics: Theory and application to camera calibration. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006. 2, 6
- [38] M. J. Harker and P. L. O'Leary. Computation of homographies. In *British Computer Vision Conference (BMVC)*. 1, 3, 4, 5, 9, 10, 15, 16, 18
- [39] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.
- [40] R. Hartley and A. Zisserman. Multiple view geometry in computer vision, 2nd ed. Cambridge University Press, 2003. 1, 2, 3, 4, 5, 6, 7, 9, 10, 15, 16, 18
- [41] M. Hong, Y. Lu, N. Ye, C. Lin, Q. Zhao, and S. Liu. Unsupervised homography estimation with coplanarity-aware gan. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2022. 5
- [42] J. Hughes, A. Van Dam, M. McGuire, J. Foley, D. Sklar, S. Feiner, and K. Akeley. *Computer Graphics: Principles and Practice, Third Edition*. The systems programming series. Addison-Wesley, 2014. 13
- [43] J. Illingworth and J. Kittler. A survey of the hough transform. Computer Vision, Graphics, and Image Processing, 44(1):87– 116, 1988.
- [44] G. Jiang and L. Quan. Detection of concentric circles for camera calibration. In *International Conference on Computer Vision (ICCV)*, 2005.
- [45] Y. Jin, D. Mishkin, A. Mishchuk, J. E. S. Matas, P. Fua, K. M. Yi, and E. Trulls. Image matching across wide baselines: From paper to practice. *International Journal of Computer Vision (IJCV)*, 129:517–547, 2021. 15
- [46] D. C. Lay, S. R. Lay, and J. J. McDonald. *Linear Algebra and Its Applications, Fifth Edition*. Pearson Publishers Inc., 2014.
- [47] H. Le, F. Liu, S. Zhang, and A. Agarwala. Deep homography estimation for dynamic scenes. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1, 5, 13, 16
- [48] S. Li, C. Xu, and M. Xie. A robust o(n) solution to the perspective-n-point problem. *IEEE Transactions on Pattern*

- Analysis and Machine Intelligence (PAMI), 34(7):1444–1450, 2012.
- [49] Y. Li and N. R. Gans. Predictive ransac: Effective model fitting and tracking approach under heavy noise and outliers. *Computer Vision and Image Understanding*, 161:99–113, 2017.
- [50] D. Liebowitz and A. Zisserman. Metric rectification for perspective images of planes. In *IEEE/CVF Conference on Com*puter Vision and Pattern Recognition (CVPR). 2, 6, 7
- [51] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollr, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014. 15
- [52] Y. Lin, S. L. Pintea, and J. C. van Gemert. Deep houghtransform line priors. In *European Conference on Computer Vision (ECCV)*, 2020.
- [53] R. Litman, S. Korman, A. Bronstein, and S. Avidan. Inverting ransac: Global model detection via inlier rate estimation. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 5243–5251, 2015.
- [54] S. Liu, N. Ye, C. Wang, K. Luo, J. Wang, and J. Sun. Content-aware unsupervised deep homography estimation and its extensions. In *European Conference on Computer Vision (ECCV)*, 2020. 1, 5, 13, 17, 18
- [55] C. G. Looney. A new approach to fuzzy clustering. In Computers and Their Applications, pages 268–273, 2000.
- [56] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. 1, 4, 15, 16
- [57] L. Magri and A. Fusiello. Multiple structure recovery with t-linkage. *Journal of Visual Communication and Image Repre*sentation, 49:57–77, 2017.
- [58] J. Matas and O. Chum. Randomized ransac with sequential probability ratio test. In *International Conference on Computer Vision (ICCV)*, volume 2, pages 1727–1732, 2005.
- [59] F. M. Mcmahon. The livermore fortran kernels: A computer test of numerical performance range. *Technical Report*, *Lawrence Livermore National Laboratory*, 15. 7
- [60] J. More. *Numerical Analysis, Lecture Notes in Mathematics* 630. Springer-Verlag, 1978. 16
- [61] T. Müller, A. Evans, C. Schied, and A. Keller. Instant neural graphics primitives with a multiresolution hash encoding. ACM Transactions on Graphics (ToG), 41(4):102:1–102:15, July 2022. 18
- [62] A. Méler, M. Decrouez, and J. L. Crowley. Betasac: A new conditional sampling for ransac. In *British Machine Vision Conference (BMVC)*, pages 1–11, 2010.
- [63] T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, and V. Kumar. Unsupervised deep homography: A fast and robust homography estimation model. *IEEE Robotics and Automation Letters (RAL)*, 3(3):2346–2353, 2018. 1, 2, 5, 13, 17, 18
- [64] K. Ni, H. Jin, and F. Dellaert. Groupsac: Efficient consensus in the presence of groupings. In *International Conference on Computer Vision (ICCV)*, pages 2193–2200, 2009.
- [65] L. Nie, C. Lin, K. Liao, S. Liu, and Y. Zhao. Unsupervised deep image stitching: Reconstructing stitched features to images. *IEEE Transactions on Image Processing (TIP)*, 30:6184–6197, 2021. 1, 5, 13

- [66] L. Nie, C. Lin, K. Liao, S. Liu, and Y. Zhao. Depth-aware multi-grid deep homography estimation with contextual correlation. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 32(7):4460–4472, 2022. 1, 5, 13
- [67] F. E. Nowruzi, R. Laganiere, and N. Japkowicz. Homography estimation from image pairs with hierarchical convolutional networks. In *International Conference on Computer Vision Workshops (ICCVW)*, 2017. 5
- [68] D. A. Patterson and J. L. Hennessy. Computer Organization and Design, Revised Fourth Edition. Morgan Kaufmann Publishers Inc., 2011. 7
- [69] R. Pautrat, J.-T. Lin, V. Larsson, M. R. Oswald, and M. Pollefeys. Sold2: Self-supervised occlusion-aware line description and detection. In *IEEE/CVF Conference on Computer Vision* and Pattern Recognition (CVPR), 2021. 2
- [70] R. Raguram, O. Chum, M. Pollefeys, J. Matas, and J. M. Frahm. Usac: A universal framework for random sample consensus. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(8):2022–2038, 2013. 2, 4, 5
- [71] P. Rangarajan and P. Papamichalis. Estimating homographies without normalization. In *International Conference on Image Processing (ICIP)*, pages 3517–3520, 2009.
- [72] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: an efficient alternative to sift or surf. In *International Conference* on Computer Vision (ICCV), 2011. 1, 4, 15, 16
- [73] J. G. Semple and G. T. Kneebone. Algebraic projective geometry. Clarendon Press, 1952. 2, 6
- [74] H. Shao, T. Svoboda, and L. V. Gool. Zubud ± zurich buildings database for image based recognition. 2003.
- [75] R. Shao, G. Wu, Y. Zhou, Y. Fu, L. Fang, and Y. Liu. Localtrans: A multiscale local transformer network for crossresolution homography estimation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 1, 5, 13
- [76] N. Simond and P. Rives. Homography from a vanishing point in urban scenes. In *International Conference on Intelligent Robots and Systems*, volume 1, pages 1005–1010, 2003.
- [77] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354–356, 1969.
- [78] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.
- [79] P. Sturm and S. Maybank. On plane-based camera calibration: A general algorithm, singularities, applications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 432–437, 1999.
- [80] Y. Tian, X. Yu, B. Fan, F. Wu, H. Heijnen, and V. Balntas. Sosnet: Second order similarity regularization for local descriptor learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 4
- [81] R. Toldo and A. Fusiello. Robust multiple structures estimation with j-linkage. In *European Conference on Computer Vision* (*ECCV*), pages 537–547, 2008.
- [82] P. H. S. Torr, A. Zisserman, and S. J. Maybank. Robust detection of degenerate configurations while estimating the fundamental matrix. *Computer Vision and Image Understanding*, 71(3):312–333, 1998.

- [83] R. Vidal. Subspace clustering. *Signal Processing Magazine*, 28(2):52–68, 2011.
- [84] W. Wang, J. Yang, and R. R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *International Conference on Very Large Data Bases*, pages 186–195, 1997.
- [85] Z. Wei. Nonparametric estimation of multiple structures with outliers. In *International Conference on Dynamical Vision*, pages 60–74, 2006.
- [86] Y. Wu, X. Li, F. Wu, , and Z. Hu. Coplanar circles, quasi-affine invariance and calibration. *Image and Vision Computing (IVC)*, 24(4):319–326, 2006. 2, 6
- [87] N. Ye, C. Wang, H. Fan, and S. Liu. Motion basis learning for unsupervised deep homography estimation with subspace projection. In *International Conference on Computer Vision* (*ICCV*), 2021. 5
- [88] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. Lift: Learned invariant feature transform. In *European Conference on Computer Vision (ECCV)*, 2016. 1, 4
- [89] S. Yue, M. Wei, J. S. Wang, and H. Wang. A general gridclustering approach. *Pattern Recognition Letters*, 29(9):1372– 1384, 2008.
- [90] J. Zaragoza, T.-J. Chin, M. S. Brown, and D. Suter. Asprojective-as-possible image stitching with moving dlt. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.
- [91] R. Zeng, S. Denman, S. Sridharan, and C. Fookes. Rethinking planar homography estimation using perspective fields. In Asian Conference on Computer Vision (ACCV), 2018. 5
- [92] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 22(11):1330–1334, 2000. 1, 14
- [93] Y. Zhao, X. Huang, and Z. Zhang. Deep lucas-kanade homography for multimodal image alignment. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 5