Journal of Mathematical Imaging and Vision
https://doi.org/10.1007/s10851-024-01182-1

Sparse Resultant-Based Minimal Solvers in Computer Vision and Their Connection with the Action Matrix

Snehal Bhayani[1] · **Janne Heikkilä**[1] · **Zuzana Kukelova**[2]

Received: 28 October 2023 / Accepted: 22 February 2024

Abstract

Many computer vision applications require robust and efficient estimation of camera geometry from a minimal number of input data measurements. Minimal problems are usually formulated as complex systems of sparse polynomial equations. The systems usually are overdetermined and consist of polynomials with algebraically constrained coefficients. Most state-of-the-art efficient polynomial solvers are based on the action matrix method that has been automated and highly optimized in recent years. On the other hand, the alternative theory of sparse resultants based on the Newton polytopes has not been used so often for generating efficient solvers, primarily because the polytopes do not respect the constraints amongst the coefficients. In an attempt to tackle this challenge, here we propose a simple iterative scheme to test various subsets of the Newton polytopes and search for the most efficient solver. Moreover, we propose to use an extra polynomial with a special form to further improve the solver efficiency via Schur complement computation. We show that for some camera geometry problems our resultant-based method leads to smaller and more stable solvers than the state-of-the-art Gröbner basis-based solvers, while being significantly smaller than the state-of-the-art resultant-based methods. The proposed method can be fully automated and incorporated into existing tools for the automatic generation of efficient polynomial solvers. It provides a competitive alternative to popular Gröbner basis-based methods for minimal problems in computer vision. Additionally, we study the conditions under which the minimal solvers generated by the state-of-the-art action matrix-based methods and the proposed extra polynomial resultant-based method, are equivalent. Specifically, we consider a step-by-step comparison between the approaches based on the action matrix and the sparse resultant, followed by a set of substitutions, which would lead to equivalent minimal solvers.

Keywords Sparse resultants · Gröbner basis · Minimal solvers · Action matrix · Geometric computer vision

1 Introduction

The robust estimation of camera geometry, one of the most important tasks in computer vision, is usually based on solving so-called minimal problems [26, 27, 43], i.e., problems that are solved from minimal samples of input data, inside a RANSAC framework [13, 18, 45]. Since the camera geometry estimation has to be performed many times inside RANSAC [18], fast solvers to minimal problems are of high importance. Minimal problems often result in polynomial systems with the following characteristics:

1. They are usually non-square, with more polynomials than the number of variables.
2. The coefficients of the constituent polynomials usually are non-generic, i.e. algebraically constrained [15, p. 109].
3. The constituent polynomials have the same structure, i.e. the same monomials, but the coefficient values vary with the input measurements. This property has enabled an offline + online strategy, which moves as many computations as possible from the "online" stage of solving equations to an earlier pre-processing "offline" stage.

✉ Snehal Bhayani
snehal.bhayani@oulu.fi

Janne Heikkilä
janne.heikkila@oulu.fi

Zuzana Kukelova
kukelova@cmp.felk.cvut.cz

[1] Faculty of Information Technology and Electrical Engineering, Center for Machine Vision and Signal Analysis, University of Oulu, Oulu, Finland

[2] Visual Recognition Group, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Prague, Czech Republic

Published online: 23 March 2024

Most of the state-of-the-art specific minimal solvers are based on Gröbner bases and the action matrix method [15, 47]. The Gröbner basis method was popularized in computer vision by Stewenius [50]. The first efficient Gröbner basis-based solvers were mostly hand-crafted [48, 49] and sometimes very unstable [51]. However, in the last 15 years much effort has been put into making the process of constructing the solvers more automatic [27, 32, 33] stable [9, 10] and more efficient [5, 31–33, 35, 38]. Now powerful tools are available for the automatic generation of efficient solvers based on Gröbner basis [27, 32].

Note that while all these methods are in the computer vision community known as Gröbner basis methods, most of them do not generate solvers that directly compute Gröbner bases. They usually compute polynomials from a border basis [40] or go "beyond" Gröbner and border bases [35]. However, all these methods are based on computing an action matrix.[1] Therefore, for the sake of simplicity, in this paper, we will interchangeably write "action matrix method" and "Gröbner basis method" to refer to the same class of methods [5, 27, 31–33, 35, 38, 47].

The first step in such action matrix methods is to compute a linear basis B of the quotient ring $A = \mathbb{C}[X]/I$ where I denotes the ideal generated by input polynomial system. An action matrix method based on the Gröbner bases computes the basis of A by defining the division w.r.t. some Gröbner basis of the ideal I. In general, computing the Gröbner basis requires to *fix* some monomial ordering. Alternatively, a heuristic basis sampling method [35] that goes beyond Gröbner bases and monomial orderings in order to compute B can be used. The basis sampling method is related to the border basis methods [39, 40], which generalize the concept of monomial ordering and even propose a non-monomial basis of A. The second step in these action matrix methods is to compute a linear map T_f, representing the multiplication in A, w.r.t. some polynomial f. The representative matrix $\mathtt{M}_f$ of the map T_f is what we call the action matrix. Recently, [54, 55] generalized the method used for generating the basis B. The proposed methods are known as normal form methods, developed for general systems of polynomial equations. They are not tailored to problems that appear in computer vision.

While the action matrix method for generating efficient minimal solvers has been thoroughly studied in computer vision and all recently generated action-matrix-based solvers are highly optimized in terms of efficiency and stability, little attention has been paid to an alternative algebraic method for solving systems of polynomial equations, i.e. the resultant method [15, Chapter 3]. The resultant method was manually applied to several computer vision problems [20, 23, 26, 28]. However, in contrast to the action matrix method, there is no method for automatically generating sparse resultant-based minimal solvers. The most relevant works in this direction are the methods using the subdivision of the Minkowski sum of all the Newton polytopes [11, 12, 16] and the iterative method based on the Minkowski sum of a subset of Newton polytopes [21]. However, the subdivision methods are not applicable to polynomial systems with non-generic coefficients, whereas the method in [21] leads (due to linearizations) to larger and less efficient solvers than the action matrix solvers.

Our first objective in this paper is to study the theory of sparse resultants and propose an approach for generating efficient solvers for solving camera geometry problems, by using an extra polynomial with a special form. Our approach, based on the Newton polytopes and their Minkowski sums, is used to generate sparse resultant matrices, followed by a Schur complement computation, and a conversion of the resultant constraint into an eigenvalue problem.

Our approach differs from the subdivision methods [11, 16] in how the Newton polytopes are used for generating minimal solvers. The subdivision methods compute the Minkowski sum of the Newton polytopes of all the polynomials and divide its lattice interior into cells, which are used to construct the sparse resultant matrices. Whereas in this paper, we make this process iterative, computing the Minkowski sum of each subset of Newton polytopes (of the input polynomials) and avoid computing a subdivision. We directly use the lattice interior to generate a candidate sparse resultant matrix. For systems with non-generic coefficients, such an iterative approach has been crucial in generating sparse resultant matrices for the minimal problems in computer vision. The sparse resultant matrix and the eigenvalue problem together represent our minimal solver. The solvers based on our proposed approach have achieved significant improvements over the state-of-the-art sparse resultant-based solvers [11, 16, 21] and achieved comparable or better efficiency and/or accuracy to state-of-the-art Gröbner basis-based solvers [32, 35, 38]. Moreover, our proposed approach has been fully automated [2] and can be incorporated in the existing tools for automatic generation of efficient minimal solvers [27, 32, 35].

There is a similarity between the solvers obtained using our sparse resultant-based approach and the solvers based on the action matrix methods [8, 27, 32, 35, 37, 38, 47]. Therefore, the second objective in this paper is to investigate this similarity. In a step-by-step manner, we demonstrate that for a minimal solver generated based on the action matrix method, we can change the steps performed by the sparse resultant method such that it leads to an equivalent solver. Similarly, we also demonstrate that for a given minimal solver generated based on the sparse resultant method, we can change the steps performed by the action matrix method such that it leads to an equivalent solver. Specifically, our contributions are:

[1] In the field of mathematics, an action matrix is also known as a multiplication matrix.

1. A novel approach (Sect. 3), of adding an extra polynomial with a special form for generating a sparse resultant matrix whose sparse resultant constraint can be decomposed (Sect. 3.3) into an eigenvalue problem.
   - A scheme (Sect. 3.2) to iteratively test the Minkowski sum of the Newton polytopes of each subset of polynomials, searching for the most efficient minimal solver, in the presence of algebraically constrained coefficients.
   - Two procedures (Sect. 3.4) to reduce the sparse resultant matrix size, leading to comparable or better solver speeds than those generated by many state-of-the-art Gröbner basis-based methods.
   - A general method for automatic generation of efficient minimal solvers. The automatic generator is publicly available at [2].
2. A study of the constraints (Sect. 4) to be satisfied so that the solvers based on our sparse resultant method as well as the action matrix method are equivalent, i.e. they involve eigenvalue decomposition of exactly the same matrix, and the steps performed for constructing that matrix can be interchanged.

This paper is an extension of our work [3], where we proposed an extra polynomial sparse resultant method for generating efficient minimal solvers.

2 Theoretical Background and Related Work

In this section, we summarize the basic concepts and notation used in the paper. This notation is based on the book by [15], to which we refer the reader for more details.

Our goal is to compute the solutions to a system of m polynomial equations,

$$f_1(x_1, \ldots, x_n) = 0, \ldots, f_m(x_1, \ldots, x_n) = 0, \tag{1}$$

in n variables, $X = \{x_1, \ldots, x_n\}$. Let us denote the set of polynomials from (1) as $\mathcal{F} = \{f_1, \ldots, f_m\}$. The variables in X can be ordered and formed into a vector. W.l.o.g. let us assume that this vector has the form, $\mathbf{x} = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}^\top$. Then for a vector $\alpha = \begin{bmatrix} \alpha_1 & \ldots & \alpha_n \end{bmatrix}^\top \in \mathbb{N}^n$, a monomial $\mathbf{x}^\alpha$ is a product $\mathbf{x}^\alpha = \prod_{i=1}^n x_i^{\alpha_i}$. Each polynomial $f_i \in \mathcal{F}$ is expressed as a linear combination of a finite set of monomials

$$f_i(\mathbf{x}) = \sum_{\alpha \in \mathbb{N}^n} c_{i,\alpha} \mathbf{x}^\alpha. \tag{2}$$

We collect all monomials present in $\mathcal{F}$ and denote the set as $\mathrm{mon}(\mathcal{F})$. Let $\mathbb{C}[X]$ denote the set of all polynomials in unknowns X with coefficients in $\mathbb{C}$. The ideal $I = \langle f_1, \ldots, f_m \rangle \subset \mathbb{C}[X]$ is the set of all polynomial combinations of generators $f_1, \ldots, f_m$. The set V of all solutions of the system (1) is called the affine variety. Each polynomial $f \in I$ vanishes on the solutions of (1). Here we assume that the ideal I generates a zero-dimensional variety, i.e., the system (1) has a finite number of solutions (say r). Using the ideal I, we can define the quotient ring $A = \mathbb{C}[X]/I$ as the set of equivalence classes in $\mathbb{C}[X]$, where the equivalence is defined by the relation, $a \sim b \iff (a - b) \in I$. If I has a zero-dimensional variety, then the quotient ring A is a finite-dimensional vector space over $\mathbb{C}$ [15]. For an ideal I, there exist special sets of generators called Gröbner bases which have the nice property that the remainder after division is unique. Using a Gröbner basis we can define a linear basis for the quotient ring A.

Non-generic polynomial system In geometric computer vision, each coefficient, say $c_{i,\alpha}$ in (2), can be expressed as a parametric form, $c_{i,\alpha} = \phi(\mathcal{D})$ of the data measurements $\mathcal{D}$. One of the important properties of the set of the coefficients of such a polynomial system is that of non-genericity:

Definition 1 The coefficients $\{c_{i,\alpha}\}$ of $\mathcal{F}$ in (2), are defined to be non-generic, if and only if we can obtain a polynomial constraint on them by eliminating the parameters $\mathcal{D}$.

A rigorous definition of the genericity of a property of a system is given in [15, p. 109].

Vector representation of a set Given a set B, one of the steps in this paper is to express it as a column vector by arranging its elements w.r.t. to a given ordering. Let us denote this operation as

$$\mathrm{vec}(B). \tag{3}$$

Matrix form In this paper, we will need to express a given set of polynomials $\mathcal{F}$ via matrix multiplication. To achieve this, we fix an order for the polynomials in $\mathcal{F}$, and also do the same for the monomials in $B = \mathrm{mon}(\mathcal{F})$. The matrix representation of $\mathcal{F}$ has the form

$$\mathtt{C}([c_{i,\alpha}])\ \mathbf{b}, \tag{4}$$

where $\mathbf{b}$ is a column vector of the monomials in B ordered w.r.t. the given monomial ordering and $\mathtt{C}([c_{i,\alpha}])$ is the matrix of coefficients $c_{i,\alpha}$ of $\mathcal{F}$. The entries of $\mathtt{C}([c_{i,\alpha}])$ are row-indexed by the ordered set of polynomials in $\mathcal{F}$ and column-indexed by the monomials in $\mathbf{b}$.

Monomial multiples of a polynomial set We begin with a polynomial system $\mathcal{F}$ and the set of monomials in $\mathcal{F}$, i.e., $B = \mathrm{mon}(\mathcal{F})$. Let $B' \supset B$ be another set of monomials.

Then an important step in our proposed method is to extend $\mathcal{F}$ to the largest possible set, say $\mathcal{F}'$, such that $\mathrm{mon}(\mathcal{F}') \subseteq B'$. This extension is done by multiplying each $f_i \in \mathcal{F}$ with monomials. i.e., we are generating polynomials

from $I = \langle f_1, \ldots, f_m \rangle$. For this, we compute the set T_i of all possible monomials $T_i = \{\mathbf{x}^\alpha\}$ for each $f_i \in \mathcal{F}$, such that $\text{mon}(\mathbf{x}^\alpha f_i) \subset B'$, $\mathbf{x}^\alpha \in T_i$. We will use the following shorthand notation to express such an operation

$$\mathcal{F} \xrightarrow{B'} (\mathcal{F}', T), \tag{5}$$

where $T = \{T_1, \ldots, T_m\}$. Subsequently, in this operation we assume to have removed all monomials in B' which are not present in the extended set $\mathcal{F}'$ and denote the modified set as B' by abuse of notation. In other words, we will assume that $B' = \text{mon}(\mathcal{F}')$.

2.1 The Action Matrix Method

One of the well-known SOTA methods for polynomial solving is the Gröbner basis-based action matrix method [15, 47, 52]. It has been recently used to efficiently solve many minimal problems in computer vision [8, 9, 26, 27, 32, 35, 47]. It transforms the problem of finding the solutions to (1), to a problem of eigendecomposition of a special matrix. We list the steps performed by an action matrix method in "Appendix A", and note here that the algorithm can essentially be distilled in a sequence of three important steps, viz. construct the set T_j of monomial multiples for each of the input polynomials $f_j \in \mathcal{F}$, extend $\mathcal{F}$ via monomial multiplication to $\mathcal{F}'$ and linearize the resulting system as a matrix product, $\mathtt{C}\mathbf{b}$. A G–J elimination of $\mathtt{C}$ is then used to extract the required action matrix, whose eigenvalues give us the roots of $\mathcal{F}$.

2.2 Resultants

Alternative to the action matrix method, we have the method of resultants for polynomial solving. Originally, resultants were used to determine whether a system of $n+1$ polynomial equations in n unknowns has a common root or not. Let us have a system of polynomial equations as defined in (1), and in (2) and assume that $m = n + 1$.

Definition 2 The resultant $Res([\{c_{i,\alpha}\}])$ of $\mathcal{F}$ is defined to be an irreducible polynomial in the coefficients $\{c_{i,\alpha}\}$ which vanishes only if $\{c_{i,\alpha}\}$ are such that $\mathcal{F} = 0$ has non-trivial roots.

A more formal treatment of the theory of resultants can be obtained from [15, Chapter 3].

2.2.1 Polynomial Solving Using Resultants

A step in a sparse resultant method is to expand the given system of polynomials $\mathcal{F}$ to a set of linearly independent polynomials. This is usually done by adding some monomial multiples of the original polynomials, i.e., using the operation (5). The expanded set of polynomials, say $\mathcal{F}'$, can be expressed in a matrix form as

$$\mathtt{C}([c_{i,\alpha}])\ \mathbf{b}. \tag{6}$$

Usually, $\mathtt{C}([c_{i,\alpha}])$ in (6) is a tall matrix.

The resultant-based method here requires the coefficient matrix $\mathtt{C}([c_{i,\alpha}])$ to be a square invertible matrix for randomly assigned values to the coefficients, $c_{i,\alpha} \in \mathbb{C}_{\neq 0}$ i.e., $\det \mathtt{C}([c_{i,\alpha}]) \neq 0$. A matrix with such properties is called the *resultant matrix* and in this paper we will denote it as $\mathtt{M}([c_{i,\alpha}])$.[2] If $\mathtt{C}([c_{i,\alpha}])$ is a square invertible matrix, we can rewrite (6) as

$$\mathtt{M}([c_{i,\alpha}])\ \mathbf{b}. \tag{7}$$

Now, $\mathcal{F} = 0$ implies $\mathcal{F}' = 0$ and it leads to

$$\mathtt{M}([c_{i,\alpha}])\ \mathbf{b} = \mathbf{0}. \tag{8}$$

Thus, the requirement for $\mathcal{F} = 0$ to have common solutions is the following condition on the coefficients of $\mathcal{F}$,

$$\det \mathtt{M}([c_{i,\alpha}]) = 0. \tag{9}$$

We call this the *resultant constraint*. It is a polynomial with $c_{i,\alpha}$ as variables. From the definition of a resultant $Res([c_{i,\alpha}])$ [15, Theorem 2.3], we have that $Res([c_{i,\alpha}])$ is a polynomial with $c_{i,\alpha}$ as variables and that it vanishes iff the system of equations $\mathcal{F} = 0$ has common roots. This gives us the necessary condition for the existence of roots of the system $\mathcal{F} = 0$, that $\det \mathtt{M}([c_{i,\alpha}])$ must vanish if the resultant vanishes, i.e.,

$$Res([c_{i,\alpha}]) = 0 \implies \det \mathtt{M}([c_{i,\alpha}]) = 0. \tag{10}$$

This implies that given a polynomial system $\mathcal{F}$, the resultant constraint (9) is a non-trivial multiple of its resultant $Res([c_{i,\alpha}])$.

While resultants are defined for a system of one more polynomial than the number of variables, we can employ them for solving a system of n polynomials in n variables. One way to do this, is to hide one of the variables to the coefficient field (in other words, consider it to be a constant), another way is to add an extra polynomial by introducing a new variable, and then hide this variable to the coefficient field. In both these approaches, we end up with a system where we have one more polynomial than the number of variables.

[2] E.g. if $\mathtt{C}([c_{i,\alpha}])$ in (6) is a tall matrix, i.e. matrix with more rows than columns, with full column rank, then $\mathtt{M}([c_{i,\alpha}])$ can be constructed as a full rank square submatrix of $\mathtt{C}([c_{i,\alpha}])$

2.2.2 Hiding a Variable

By *hiding* one variable, say x_n, to the coefficient field, we are left with n polynomials $\mathcal{F}$ in $n-1$ variables. This gives us a way to use the concept of resultants and compute $Res([c_{i,\alpha}], x_n)$ which now becomes a function of $c_{i,\alpha}$ and x_n. In this case, (7) becomes

$$\mathtt{M}([c_{i,\alpha}], x_n)\mathbf{b}, \tag{11}$$

where the symbols have their usual meaning. For simplicity, we will denote $\mathtt{M}([c_{i,\alpha}], x_n)$ as $\mathtt{M}(x_n)$ in the rest of this paper. Its determinant $\det \mathtt{M}(x_n)$ is a multiple of the resultant $Res(x_n)$. This is known as a *hidden variable resultant* and it is a polynomial in x_n whose roots are the x_n-coordinates of the solutions of the system of polynomial equations. For theoretical details and proofs we refer to [15, Chapter 7]. Such a hidden variable approach has been used in the past to solve various minimal problems [20, 23, 26, 28].

This approach leads to computing the roots of the polynomial, $\det \mathtt{M}(x_n) = 0$. However, computing the determinant of a polynomial matrix $\det \mathtt{M}(x_n)$ and then its roots may be slow and unstable. Therefore, the most common way to solve the original system of polynomial equations is to first transform the following matrix equation

$$\mathtt{M}(x_n)\mathbf{b} = \mathbf{0}, \tag{12}$$

to a polynomial eigenvalue problem (PEP) [14], which is then expressed as,

$$(\mathtt{M}_0 + \mathtt{M}_1\, x_n + \cdots + \mathtt{M}_l\, x_n^l)\mathbf{b} = \mathbf{0}, \tag{13}$$

where l is the degree of the matrix $\mathtt{M}(x_n)$ in the hidden variable x_n and $\mathtt{M}_0, \ldots, \mathtt{M}_l$ are matrices that depend only on the coefficients $c_{i,\alpha}$ of the original system of polynomials. The PEP (13) can be easily converted to a generalized eigenvalue problem (GEP), written as,

$$\mathtt{A}\mathbf{y} = x_n \mathtt{B}\mathbf{y}, \tag{14}$$

and solved using standard efficient eigenvalue algorithms [28]. Basically, the eigenvalues give us the solutions to x_n and the rest of the variables can be extracted from the corresponding eigenvectors, $\mathbf{y}$. Such a transformation to a GEP relaxes the original problem of finding the solutions to the input polynomial system since the eigenvectors in general do not satisfy the monomial dependencies induced by the monomial vector $\mathbf{b}$ as well the monomial vector $\mathbf{y}$. Moreover, this relaxation may also introduce extra parasitic (zero) eigenvalues leading to slower polynomial solvers.

2.2.3 Adding an Extra Polynomial

Alternatively, we can add a new polynomial

$$f_{n+1} = u_0 + u_1 x_1 + \cdots + u_n x_n \tag{15}$$

to $\mathcal{F}$. We thus have an augmented polynomial system.

$$\mathcal{F}_a = \mathcal{F} \cup \{f_{m+1}\}. \tag{16}$$

We then compute the so-called *u-resultant* (see [56] and [15, Chapter 3]) by hiding $u_0, \ldots, u_n$. In general, random values are assigned to $u_1, \ldots, u_n$ (and u_0 is a new unknown). Just like in the hidden variable method, this method hides the variable u_0 to the coefficient field and generates the resultant matrix $\mathtt{M}(u_0)$. In this case, the extended system of polynomials (7) can be expressed as a matrix product as

$$\mathtt{M}([c_{i,\alpha}], u_0)\ \mathbf{b}, \tag{17}$$

where u_0 is a new unknown variable and $\mathbf{b}$ is a vector of all monomials in this extended system. We will denote $\mathtt{M}([c_{i,\alpha}], u_0)$ as $\mathtt{M}(u_0)$ in order to simplify the notation. Again, it holds that $\det \mathtt{M}(u_0)$ is a multiple of the resultant $Res(u_0)$. Thus, in order to compute the common zeros of $\mathcal{F} = 0$, we need to solve $\det \mathtt{M}(u_0) = 0$, similar to the case of the hidden variable resultant. Instead of computing the determinant of a polynomial matrix, here we also solve it as a GEP described in Sect. 2.2.2. However, as mentioned in the hidden variable method, such a method introduces spurious solutions, arising from the linearization of the original system in (17), i.e., not considering monomial dependencies in $\mathbf{b}$, as well as from transforming PEP to GEP and not considering the monomial dependencies in $\mathbf{y}$ in (14). Some of these spurious solutions (eigenvalues) can be removed by using Schur complement, described in the section below or by applying the method for removing zero eigenvalues, similar to [4, 26].

2.2.4 Schur Complement

One way to remove the spurious solutions introduced by the linearization of a hidden variable resultant matrix $\mathtt{M}(u_0)$ is to use the Schur complement of one its submatrices. Here, we briefly review this method. Let us first consider *some* partition of the set of monomials B as

$$B = B_1 \cup B_2. \tag{18}$$

Note that $\mathbf{b} = \mathrm{vec}(B) = \begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2\end{bmatrix}^\top$, where $\mathbf{b}_1 = \mathrm{vec}(B_1)$ and $\mathbf{b}_2 = \mathrm{vec}(B_2)$. The vectorization operation is defined in Eq. (3). This imposes a column partition on $\mathtt{M}(u_0)$ in (17). Moreover, we can order the rows of $\mathtt{M}(u_0)$ such that its upper

block is independent of u_0. Together, we obtain the following block partition of $\mathtt{M}(u_0)$:

$$\mathtt{M}(u_0) = \begin{bmatrix} \mathtt{M}_{11} & \mathtt{M}_{12} \\ \mathtt{M}_{21}(u_0) & \mathtt{M}_{22}(u_0) \end{bmatrix}. \tag{19}$$

Here, the upper block $\begin{bmatrix} \mathtt{M}_{11} & \mathtt{M}_{12} \end{bmatrix}$ is independent of u_0. Thus we can write (17) as

$$\begin{bmatrix} \mathtt{M}_{11} & \mathtt{M}_{12} \\ \mathtt{M}_{21}(u_0) & \mathtt{M}_{22}(u_0) \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}. \tag{20}$$

The requirement for existence of solutions of $\mathcal{F} = 0$ is that the vector in (20) should vanish. We thus obtain the following two vector equations

$$\mathtt{M}_{11}\mathbf{b}_1 + \mathtt{M}_{12}\mathbf{b}_2 = \mathbf{0} \tag{21}$$

$$\mathtt{M}_{21}(u_0)\mathbf{b}_1 + \mathtt{M}_{22}(u_0)\mathbf{b}_2 = \mathbf{0}. \tag{22}$$

If we were able to partition $\mathtt{M}(u_0)$ such that $\mathtt{M}_{12}$ is a square invertible matrix, we can eliminate $\mathbf{b}_2$ from these two equations, to obtain

$$\underbrace{(\mathtt{M}_{21}(u_0) - \mathtt{M}_{22}(u_0)\mathtt{M}_{12}^{-1}\mathtt{M}_{11})}_{\mathtt{X}(u_0)}\mathbf{b}_1 = \mathbf{0}. \tag{23}$$

The matrix $\mathtt{X}(u_0)$ is the Schur complement of the block $\mathtt{M}_{12}$ of $\mathtt{M}(u_0)$, which has the following property:

$$\det(\mathtt{M}(u_0)) = \det(\mathtt{M}_{12})\det(\mathtt{X}), \tag{24}$$

where, $\det(\mathtt{M}_{12}) \neq 0$ by our assumption. Therefore, for a generic value of u_0, $\det(\mathtt{M}(u_0)) \neq 0 \Leftrightarrow \det(\mathtt{X}(u_0)) \neq 0$, which means that $\mathtt{X}(u_0)$ is a square invertible matrix and its determinant is also a multiple of the resultant. However, $\mathtt{X}(u_0)$ corresponds to a smaller eigenvalue problem as compared to the sparse resultant matrix $\mathtt{M}(u_0)$.

Both the resultant-based methods, described in Sects. 2.2.2 and 2.2.3, are proposed for square generic systems [16] via mixed subdivision of polytopes. However, the polynomial systems studied here are usually sparse, overdetermined and consist of polynomials with non-generic coefficients (see Definition 1). The sparsity of the systems can be exploited to obtain more compact resultants using specialized algorithms. Such resultants are commonly referred to as the *sparse resultants*.

2.2.5 Polyhedral Geometry

Sparse resultants are studied via the theory of polytopes [15, Chapter 7]. Therefore, we define the important terms and notations related to polytopes, which we will later use in the text.

The *Newton polytope* $\mathrm{NP}(f)$, of a polynomial f is defined as a convex hull of the exponent vectors of all the monomials occurring in the polynomial (also known as the support of the polynomial). Therefore, we have $\mathrm{NP}(f_i) = \mathrm{Conv}(A_i)$ where A_i is the set of all integer vectors that are exponents of monomials with nonzero coefficients in f_i. A *Minkowski sum* of any two convex polytopes P_1, P_2 is defined as $P_1 + P_2 = \{p_1 + p_2 \,|\, \forall p_1 \in P_1, \forall p_2 \in P_2\}$. We demonstrate the concept of a Newton polytope using a simple example.

Example 1 Let us consider a system of two polynomials $\mathcal{F} = \{f_1(\mathbf{x}), f_2(\mathbf{x})\}$, in two variables $X = \{x_1, x_2\}$

$$\begin{aligned} f_1 = {} & c_{1,1}{x_1}^3{x_2}^3 + c_{1,2}{x_1}^2{x_2}^3 + c_{1,3}{x_1}^3{x_2}^2 \\ & + c_{1,4}{x_1}^2{x_2}^2 + c_{1,5}{x_2}^3 + c_{1,6}{x_1}^2x_2 \\ & + c_{1,7}{x_2}^2 + c_{1,8}x_1x_2 + c_{1,9}{x_1}^2 + c_{1,10}x_2 \end{aligned} \tag{25}$$

$$f_2 = c_{2,1}{x_1}^2 + c_{2,2}x_2 + c_{2,3}x_1 + c_{2,4}. \tag{26}$$

Here, $\{c_{1,1}, \ldots, c_{1,10}, c_{2,1}, \ldots, c_{2,4}\} \subset \mathbb{C}$ is the set of coefficients of $\mathcal{F}$. The supports of $\mathcal{F}$ are

$$\begin{aligned} A_1 = {} & \{[3,3],[2,3],[3,2],[2,2],[0,3], \\ & [2,1],[0,2],[1,1],[2,0],[0,1]\} \subset \mathbb{Z}^2 \end{aligned} \tag{27}$$

$$A_2 = \{[2,0],[0,1],[1,0],[0,0]\} \subset \mathbb{Z}^2. \tag{28}$$

The Newton polytopes $P_1 = \mathrm{NP}(f_1)$, $P_2 = \mathrm{NP}(f_2)$ as well as the Minkowski sum $Q = P_1 + P_2$ are depicted in Fig. 1.

2.2.6 Sparse Resultants

Consider the SOTA methods in [11, 16, 17], for computing the sparse resultant for a well-determined generic system $\mathcal{F}$. The main idea in all such methods is to compute the Minkowski sum of the Newton polytopes of all the polynomials, $f_i \in \mathcal{F}$, and divide its interior into cells. Each cell determines a multiple of one of the input polynomials, $f_i \in \mathcal{F}$ with a monomial. All such monomial multiples collectively lead to an extended polynomial system, $\mathcal{F}'$. The extended polynomial system, via linearization into a matrix product, then leads to a sparse resultant matrix $\mathtt{M}(x_n)$, if using the hidden-variable technique in Sect. 2.2.2, or $\mathtt{M}(u_0)$, if using the u-resultant of the general form in Sect. 2.2.3. As such the resulting solvers are usually quite large and not very efficient.

However, if $\mathcal{F}$ has non-generic coefficients (see Definition 1), these methods may fail, as the algebraic constraints on the coefficients lead to algebraic constraints on the elements of the sparse resultant matrices $\mathtt{M}(x_n)$ or $\mathtt{M}(u_0)$, and hence, they may not satisfy the necessary rank conditions. For such systems, [21] recently proposed an iterative approach based on the hidden variable resultant, to test and extract $\mathtt{M}(x_n)$

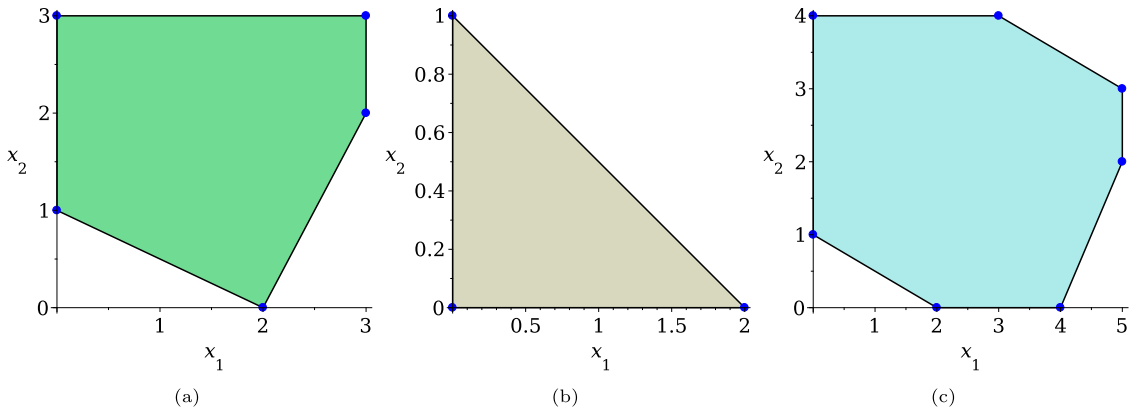


Fig. 1 An example of the Newton polytopes of two polynomials as well as their Minkowski sum: **a** $P_1 = \text{NP}(f_1)$ **b** $P_2 = \text{NP}(f_2)$ and **c** $Q = P_1 + P_2$

in (7). Thereafter, it transforms (7) to a GEP (14) and solves for eigenvalues and eigenvectors to compute the roots. Our proposed approach here, extends this iterative scheme for generating a sparse resultant matrix $\mathtt{M}(u_0)$, specifically for the extra polynomial approach in Sect. 2.2.3.

3 Proposed Extra Polynomial Resultant Method

In this paper, we propose an iterative approach based on [21] and apply it to a modified version of the u-resultant method described in Sect. 2.2.3. Specifically, we propose the following modifications to the u-resultant method.

1. Instead of the general form (15) of the extra polynomial in the u-resultant-based method, in this paper, we propose to use the following special form of the extra polynomial

$$f_{m+1} = x_k - u_0, \tag{29}$$

   where u_0 is a new variable and $x_k \in X$ is one of the input variables. In general, we can select any $x_k \in X$. However, since in practice, selecting different x_k's leads to solvers of different sizes and with different numerical stability, we test all $x_k \in X$ when generating the final solver.
2. For non-generic and overdetermined polynomial systems $\mathcal{F}$, we avoid computing the Minkowski sum of all the Newton polytopes $NP(f), \forall f \in \mathcal{F}_a$, as proposed in the methods in [11, 16, 17]. Here, $\mathcal{F}_a$ denotes the augmented polynomial system (see Eq. (16)). Instead, we iterate through each subset, $\mathcal{F}_{\text{sub}} \subset \mathcal{F}_a$, and compute the Minkowski sum of $NP(f), \forall f \in \mathcal{F}_{\text{sub}}$. Instead of dividing the Minkowski sum into cells, we simply use its lattice interior to determine the monomials B in the extended polynomial system $\mathcal{F}'$, i.e. $B = \text{mon}(\mathcal{F}')$.
3. We exploit the form of the extra polynomial (29) and propose a block partition of $\mathtt{M}(u_0)$ (17), which facilitates its decomposition using Schur complement directly into a regular eigenvalue problem. This regular eigenvalue problem, compared to GEP that arises for general u-resultant polynomial (15), leads to fewer spurious eigenvalues and hence a faster solver.

Note that our method of iteratively testing various polynomial subsets has been quite effective in generating efficient solvers, as demonstrated on many minimal problems (see Sect. 6). The generated solvers are comparable in efficiency and speed with those generated by the SOTA action matrix-based methods [32, 35, 38].

3.1 Method Outline

In the following, we first go through the important steps performed by our method.

1. Let us consider a system of m polynomial equations, $\mathcal{F} = 0$ (1), in n variables, $X = \{x_1, \ldots, x_n\}$. For all variables $x_k \in X$, we perform the steps $2-5$ in the offline phase.
2. **[Offline]** We fix the form of the extra polynomial to $f_{m+1} = x_k - u_0$ (29) and augment $\mathcal{F}$ with f_{m+1}, to obtain the augmented system (see Eq. (16)). We hide the new variable u_0 to the coefficient field which means that, $\mathcal{F}_a \subset \mathbb{C}[X \cup \{u_0\}]$.
3. **[Offline]** We execute steps 3(a)-3(c), for each subset of polynomials $\mathcal{F}_{\text{sub}} \subset \mathcal{F}_a$ and for every variable $x_k \in X$.
   (a) From the set $\mathcal{F}_{\text{sub}}$, we attempt to construct a *favourable monomial set* B, using a polytope-based method, described in Sect. 3.2.

(b) We extend the polynomial system, $\mathcal{F}_a$, using the computed monomial set B, represented as

$$\mathcal{F}_a \xrightarrow{B} (\mathcal{F}'_a, T), \tag{30}$$

where $T = \{T_1, \ldots, T_{m+1}\}$ (5).

(c) The set of equations $\mathcal{F}'_a = 0$, can be expressed in a form of a matrix equation as $\mathtt{C}(u_0)\mathbf{b} = \mathbf{0}$, where $\mathbf{b} = \text{vec}(B)$, where the vectorization operation is defined in Eq. (3).

The output of this step is described in detail in Sect. 3.2, is all favourable monomial sets B and the corresponding coefficient matrices $\mathtt{C}(u_0)$.

4. **[Offline]** For each favourable monomial set B and the corresponding coefficient matrix $\mathtt{C}(u_0)$ we perform the following:

(a) We partition $B = B_1 \cup B_2$ in two different ways,

$$B_1 = B \cap T_{m+1} \text{ or} \tag{31}$$

$$B_1 = \{\mathbf{x}^\alpha \in B \mid \frac{\mathbf{x}^\alpha}{x_k} \in T_{m+1}\}. \tag{32}$$

Note that $B_2 = B \setminus B_1$.

(b) Based on the two set-partitions of B, we attempt to partition the matrix $\mathtt{C}(u_0)$ in the following two ways.

$$\mathtt{C}(u_0)\mathbf{b} = \begin{bmatrix} \mathtt{A}_{11} & \mathtt{A}_{12} \\ \mathtt{A}_{21} - u_0\mathtt{I} & \mathtt{A}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = \mathbf{0}, \tag{33}$$

or

$$\mathtt{C}(u_0)\mathbf{b} = \begin{bmatrix} \mathtt{A}_{11} & \mathtt{A}_{12} \\ \mathtt{I} + u_0\mathtt{B}_{21} & u_0\mathtt{B}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = \mathbf{0}, \tag{34}$$

and the matrix $\mathtt{A}_{12}$ has the full column rank.

The output of this step is the coefficient matrix $\mathtt{C}(u_0)$, for which a partitioning as (33) or (34), with the full column rank matrix $\mathtt{A}_{12}$ is possible, and which corresponds to smallest set B_1. If we have more than one such choice of $\mathtt{C}(u_0)$, we select the smallest matrix $\mathtt{C}(u_0)$. This step is described in more detail in Sect. 3.3.

5. **[Offline]** In this step, we aim to reduce the size of the matrix $\mathtt{C}(u_0)$, selected in the previous step.

(a) We first try to remove a combination of rows and columns from $\mathtt{C}(u_0)$ and the corresponding monomials from the favourable monomial set B, such that the resulting monomial set is still a favourable monomial set (Sect. 3.2) and that the coefficient matrix $\mathtt{C}(u_0)$ can still be partitioned as in (33) or in (34).

(b) We next remove the extra rows from $\mathtt{C}(u_0)$ to obtain a sparse resultant matrix $\mathtt{M}(u_0)$ (7), while still respecting its block partition, as in (33) or (34).

This step is described in Sect. 3.4.

6. **[Online]** In the final online solver, we fill the precomputed sparse resultant matrix $\mathtt{M}(u_0)$ with the coefficients coming from the input data/measurements. Then we compute the Schur complement of a block of $\mathtt{M}(u_0)$, as described in 2.2.4, which is then formulated as an eigenvalue problem of the matrix $\mathtt{X}$ (50) (or (52)). Finally, the eigendecomposition of the matrix $\mathtt{X}$ gives us the solutions to the input system of polynomial equations $\mathcal{F}$, i.e. to the given instance of the minimal problem.

3.2 Computing a Favourable Monomial Set

Let the extra polynomial f_{m+1} have the form $f_{m+1} = x_k - u_0$ (29) for $x_k \in X$, and let us assume a subset $\mathcal{F}_{\text{sub}} \subset \mathcal{F}_a$ of the augmented system $\mathcal{F}_a = \mathcal{F} \cup \{f_{m+1}\}$ (16).

In the step 3(a) of our algorithm (Sect. 3.1), we attempt to generate so-called favourable monomial sets. Our method for constructing these sets is based on the polytope-based algorithm proposed in [21], where such sets were generated for the original system $\mathcal{F}$ and its subsets. Here we describe our method that constructs such favourable sets for subsets $\mathcal{F}_{\text{sub}}$ of the augmented system $\mathcal{F}_a$ (16). In this subsection, we refer to the basic terminology and notations described in Sect. 2.2.5.

In our method, we begin by computing the support $A_j = \text{supp}(f_j)$ and the Newton polytope $\text{NP}(f_j) = \text{conv}(A_j)$ for each polynomial $f_j \in \mathcal{F}_a$. We also compute the unit simplex $\text{NP}_0 \subset \mathbb{Z}^n$ and the Minkowski sum, $Q = \text{NP}_0 + \Sigma_{f \in \mathcal{F}_{\text{sub}}} \text{NP}(f)$. Let us construct a *small* displacement vector $\delta \in \mathbb{R}^n$, such that each of its elements is assigned one of the three values, $\{-10^{-1}, 0, 10^{-1}\}$. In other words,

$$\delta \in \{[\delta_1 \ \ldots \ \delta_n] \mid \delta_i \in \{-10^{-1}, 0, 10^{-1}\}; i = 1, \ldots, n\}. \tag{35}$$

Then using this value of δ, we compute the set of integer points inside $Q + \delta$, from which we compute a set of monomials, $B = \{\mathbf{x}^\alpha \mid \alpha \in \mathbb{Z}^n \cap (Q + \delta)\}$, i.e. a potential candidate for a favourable monomial set. We demonstrate this step on a system of two polynomials in Example 2.

Example 2 Let us continue with Example 1, where we computed the Newton polytopes $P_1 = \text{NP}(f_1)$ and $P_2 = \text{NP}(f_2)$ as well as their Minkowski sum $Q = P_1 + P_2$. Let us assume the displacement vector to be $\delta = [-0.1, -0.1]$. In Fig. 2, we depict the integer points in the interior of $Q + \delta$, i.e. the points in the set $\mathbb{Z}^2 \cap (Q + \delta)$. This set of points give us a favourable set of monomials B

$$\begin{aligned} B = \{&x_2, x_2^2, x_2^3, x_1^2, x_1^3, x_1x_2, x_1x_2^2, \\ &x_1x_2^3, x_1^2x_2, x_1^2x_2^2, x_1^2x_2^3, x_1^3x_2, \\ &x_1^3x_2^2, x_1^3x_2^3, x_1^4x_2, x_1^4x_2^2, x_1^4x_2^3\}. \end{aligned} \tag{36}$$

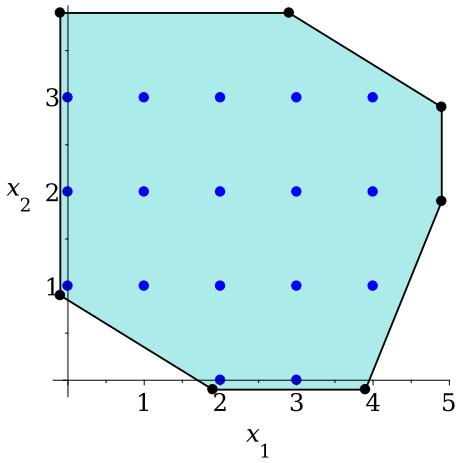


Fig. 2 The integer points (shown in blue) in the interior of the Minkowski sum of the two Newton polytopes, $Q = P_1 + P_2$ after shifting it $\delta = [-0.1, -0.1]$ (Color figure online)

Selecting the displacement vector δ

In Eq. (35), our choice of the displacement 10^{-1} in each direction is not completely random, although to the best of our knowledge there is no theory which gives us a δ leading to the smallest solver. All we know is that the displacement in each direction should be in the range $(-1.0, 1.0)$, and in this paper we have chosen a *reasonably* small value, i.e. 10^{-1}. A different choice will lead to a different favourable monomial set. To this end, we have observed, that as the number of vertices of the Minkowski sum Q increases the number of different solvers we can generate by choosing different displacements will also increase.

In the next step (see step 3(b) in Sect. 3.1), we extend the input polynomial system, $\mathcal{F}_a$, using the set of monomials B as

$$\mathcal{F}_a \overset{B}{\rightarrow} (\mathcal{F}'_a, T), \tag{37}$$

where $T = \{T_1, \ldots, T_{m+1}\}$ (see (5)). Each T_i denotes the set of monomials to be multiplied with the polynomial $f_i \in \mathcal{F}$. The extended set of polynomials $\mathcal{F}'_a$ can be written in a matrix form as

$$\mathtt{C}([c_{i,\alpha}], u_0)\ \mathbf{b}, \tag{38}$$

where $\mathbf{b}$ is a vector form of B w.r.t. some ordering of monomials. In the rest of the paper, we will denote the coefficient matrix $\mathtt{C}([c_{i,\alpha}], u_0)$ as $\mathtt{C}(u_0)$ for the sake of simplicity.

Our approach then evaluates the following three conditions:

$$\Sigma_{j=1}^{m+1} |T_j| \geq |B|, \tag{39}$$

$$\min_j |T_j| > 0, \tag{40}$$

$$\text{column rank of } \mathtt{C}(u_0) = |B|, \tag{41}$$

for a random value of $u_0 \in \mathbb{C}$. The first condition (39) ensures that we have at least as many rows as the columns in the matrix $\mathtt{C}(u_0)$. The second condition (40) ensures that there is at least one row corresponding to every polynomial $f_i \in \mathcal{F}_a$. The third condition (41) ensures that $\mathtt{C}(u_0)$ has full column rank and that we can extract a sparse resultant matrix from it. If these conditions are satisfied, we consider the set of monomials B to be a *favourable set of monomials*.

Note that we repeat this procedure for all variables $x_k \in X$, for each subset $\mathcal{F}_{\text{sub}} \subset \mathcal{F}_a$, and for each value of the displacement vector δ (35). The output of this step are all generated favourable monomial sets B and the corresponding coefficient matrices $\mathtt{C}(u_0)$.

3.3 Block Partition of $\mathtt{C}(u_0)$

In the step 4 of our method (Sec 3.1), we iterate through all favourable monomial sets B and the coefficient matrices $\mathtt{C}(u_0)$, which was the output of the previous step.

For each favourable monomial set B, we know the corresponding sets of monomial multiples T_i for each polynomial $f_i \in \mathcal{F}_a$ and the set $T = \{T_1, \ldots, T_m, T_{m+1}\}$. We have also computed the corresponding coefficient matrix $\mathtt{C}(u_0)$. Assume the extra polynomial (29) to be fixed as $f_{m+1} = x_k - u_0$, for $x_k \in X$, while computing this favourable monomial set.

Since B was constructed such that it satisfies (41), $\mathtt{C}(u_0)$ has full column rank. Let us assume, $\varepsilon = |\ B\ |$ and $p = \Sigma_{j=1}^{m+1} |T_j|$. As $\mathtt{C}(u_0)$ satisfies the condition (39), we have $p \geq \varepsilon$. Therefore, we can remove the extra $p - \varepsilon$ rows from $\mathtt{C}(u_0)$, leading to a square invertible matrix. The algorithm of row removal approach is described in B.3. The square matrix so obtained, is a sparse resultant matrix $\mathtt{M}(u_0)$ and it satisfies the resultant matrix constraint (8), if the set of equations $\mathcal{F}_a = 0$ has solutions.

Instead of directly solving the resultant constraint (9) or converting (8) to GEP, we exploit the structure of the extra polynomial f_{m+1} (29) and propose a special ordering of the rows and columns of $\mathtt{M}(u_0)$. This facilitates a block partition of $\mathtt{M}(u_0)$, such that the Schur complement of one of its block submatrices then helps us to reduce its matrix constraint (8) to a regular eigenvalue problem.[3]

Rather than block partitioning the sparse resultant matrix $\mathtt{M}(u_0)$, we first fix a block partition of the coefficient matrix $\mathtt{C}(u_0)$ and then remove its rows, while respecting this partition, to obtain the sparse resultant matrix $\mathtt{M}(u_0)$. Note that we obtain two different block partitions on $\mathbf{b}$ and $\mathtt{C}(u_0)$, in (38),

[3] Note that it may happen that neither of the two set-partitions for a given favourable monomial set B, lead to a block partition of $\mathtt{C}(u_0)$ such that $\mathtt{A}_{12}$ has full column rank. In this case, the Schur complement does not exist and our algorithm will continue to test the next favourable monomial set B.

based on our selected set-partition of the favourable monomial set $B = B_1 \cup B_2$:

$$B_1 = B \cap T_{m+1} \text{or} \tag{42}$$

$$B_1 = \left\{ \mathbf{x}^\alpha \in B \mid \frac{\mathbf{x}^\alpha}{x_k} \in T_{m+1} \right\} \tag{43}$$

Note that $B_2 = B \setminus B_1$. We consider both these partitions in Proposition 1. Thereafter, we will describe our method of removing the rows from $\mathtt{C}(u_0)$ such that we can compute a Schur complement of a block of $\mathtt{M}(u_0)$.

Proposition 1 *Consider a block partition of* $\mathtt{C}(u_0)$ *and* $\mathbf{b}$ *in* (38) *as*

$$\mathtt{C}(u_0)\,\mathbf{b} = \begin{bmatrix} \mathtt{C}_{11} & \mathtt{C}_{12} \\ \mathtt{C}_{21} & \mathtt{C}_{22} \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix}. \tag{44}$$

We can achieve this in two different ways, based on our choice the set-partition of B in (42) *or* (43)*. In either case, if* $\mathtt{C}_{12}$ *has full column rank, then the resultant matrix constraint* (8) *can be converted to an eigenvalue problem, of the form* $\mathtt{X}\,\mathbf{b}_1 = u_0\mathbf{b}_1$ *if we set-partition B as in* (42)*, and of the form* $\mathtt{X}\,\mathbf{b}_1 = -\frac{1}{u_0}\mathbf{b}_1$ *if we set-partition B as in* (43)*. The matrix* $\mathtt{X}$ *is square and its entries are the functions of the coefficients of* $\mathcal{F}$*.*

Proof Let us first consider the set-partition of B as in (42). Then, the special form of the extra polynomial (29) implies that

$$B_1 = \{\mathbf{x}^\alpha \in B \mid x_k\mathbf{x}^\alpha \in B\}. \tag{45}$$

We next order the monomials in B such that $\mathbf{b}$ can be partitioned as $\mathbf{b} = \begin{bmatrix}\text{vec}(B_1) & \text{vec}(B_2)\end{bmatrix}^T = \begin{bmatrix}\mathbf{b}_1 & \mathbf{b}_2\end{bmatrix}^T$. This induces a column partition of $\mathtt{C}(u_0)$. Moreover, we can row partition $\mathtt{C}(u_0)$ by indexing the rows in its lower block by monomial multiples of f_{m+1}. All such multiples are linear in u_0, i.e., these multiples have the form $\mathbf{x}^{\alpha_j}(x_k - u_0), \forall \mathbf{x}^{\alpha_j} \in T_{m+1}$. The upper block is row-indexed by monomial multiples of $f_1, \ldots, f_m$. Such a row and column partition of $\mathtt{C}(u_0)$ gives us a block partition as in (44). As $\begin{bmatrix}\mathtt{C}_{11} & \mathtt{C}_{12}\end{bmatrix}$ contains polynomials independent of u_0 and $\begin{bmatrix}\mathtt{C}_{21} & \mathtt{C}_{22}\end{bmatrix}$ contains polynomials of the form $\mathbf{x}^{\alpha_j}(x_k - u_0)$, we obtain

$$\begin{aligned} &\mathtt{C}_{11} = \mathtt{A}_{11}, \quad \mathtt{C}_{12} = \mathtt{A}_{12} \\ &\mathtt{C}_{21} = \mathtt{A}_{21} + u_0\mathtt{B}_{21}, \quad \mathtt{C}_{22} = \mathtt{A}_{22} + u_0\mathtt{B}_{22}, \end{aligned} \tag{46}$$

where $\mathtt{A}_{11}, \mathtt{A}_{12}, \mathtt{A}_{21}, \mathtt{A}_{22}, \mathtt{B}_{21}$ and $\mathtt{B}_{22}$ are matrices dependent only on the coefficients of input polynomials in $\mathcal{F}_a$ (16). Based on our assumption in the statement of this proposition, $\mathtt{C}_{12}$ and hence $\mathtt{A}_{12}$, have full column rank. Substituting (46) in (44) gives

$$\mathtt{C}(u_0) = \underbrace{\begin{bmatrix} \mathtt{A}_{11} & \mathtt{A}_{12} \\ \mathtt{A}_{21} & \mathtt{A}_{22} \end{bmatrix}}_{\mathtt{C}_0} + u_0 \underbrace{\begin{bmatrix} 0 & 0 \\ \mathtt{B}_{21} & \mathtt{B}_{22} \end{bmatrix}}_{\mathtt{C}_1} \tag{47}$$

We can order monomials so that $\mathbf{T}_{m+1} = \mathbf{b}_1$. Now, the block partition of $\mathtt{C}(u_0)$ implies that $\mathtt{C}_{21}$ is column-indexed by $\mathbf{b}_1$ and row-indexed by $\mathbf{T}_{m+1}$. As $\begin{bmatrix}\mathtt{C}_{21} & \mathtt{C}_{22}\end{bmatrix}$ has rows of form $\mathbf{x}^{\alpha_j}(x_k - u_0)$, $\mathbf{x}^{\alpha_j} \in T_{m+1} \implies \mathbf{x}^{\alpha_j} \in B_1$. This gives us $\mathtt{B}_{21} = -\mathtt{I}$, where $\mathtt{I}$ is an identity matrix of size $|\,B_1\,| \times |\,B_1\,|$ and $\mathtt{B}_{22}$ is a zero matrix of size $|\,B_1\,| \times |\,B_2\,|$. This also means that $\mathtt{A}_{21}$ is a square matrix of the same size as $\mathtt{B}_{21}$. Thus, we have

$$\mathtt{C}(u_0) = \begin{bmatrix} \mathtt{A}_{11} & \mathtt{A}_{12} \\ \mathtt{A}_{21} & \mathtt{A}_{22} \end{bmatrix} + u_0 \begin{bmatrix} 0 & 0 \\ -\mathtt{I} & 0 \end{bmatrix}, \tag{48}$$

where $\mathtt{C}(u_0)$ is a $p \times \varepsilon$ matrix. If $\mathtt{C}(u_0)$ is a *tall* matrix, so is $\mathtt{A}_{12}$. Therefore, we can eliminate extra rows from the upper block $\begin{bmatrix}\mathtt{A}_{11} & \mathtt{A}_{12}\end{bmatrix}$ so that we obtain a square invertible matrix $\hat{\mathtt{A}}_{12}$ while preserving the above-mentioned structure. Such a row-removal will also give us the sparse resultant matrix $\mathtt{M}(u_0)$ which satisfies the resultant matrix constraint (8). We will describe our method for removing the extra rows in Sect. 3.4. Let us now here, use the Schur complement technique 2.2.4, to reduce the size of the eigenvalue problem. From (48), we have

$$\begin{aligned} &\underbrace{\begin{bmatrix} \hat{\mathtt{A}}_{11} & \hat{\mathtt{A}}_{12} \\ \mathtt{A}_{21} & \mathtt{A}_{22} \end{bmatrix}}_{\hat{\mathtt{C}}_0} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} + u_0 \begin{bmatrix} 0 & 0 \\ -\mathtt{I} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = \mathbf{0} \\ &\implies \hat{\mathtt{A}}_{11}\mathbf{b}_1 + \hat{\mathtt{A}}_{12}\mathbf{b}_2 = \mathbf{0}, \\ &\mathtt{A}_{21}\mathbf{b}_1 + \mathtt{A}_{22}\mathbf{b}_2 - u_0\mathbf{b}_1 = \mathbf{0}. \end{aligned} \tag{49}$$

Eliminating $\mathbf{b}_2$ from the above pair of equations, we obtain

$$\underbrace{(\mathtt{A}_{21} - \mathtt{A}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11})}_{\mathtt{X}}\,\mathbf{b}_1 = u_0\mathbf{b}_1. \tag{50}$$

The matrix $\mathtt{X}$ is the Schur complement of $\hat{\mathtt{A}}_{12}$ w.r.t. $\hat{\mathtt{C}}_0$.

If we select the alternative set-partition of B, as in (43), we obtain a different block partition of $\mathbf{b}$ and $\mathtt{C}(u_0)$. Specifically, in (47), we have $\mathtt{A}_{21} = \mathtt{I}$ and $\mathtt{A}_{22} = 0$. By assumption, we have $\mathtt{A}_{12}$ has full column rank. Therefore, we can remove the extra rows from $\mathtt{C}(u_0)$ in (48) and obtain

$$\mathtt{M}(u_0) = \begin{bmatrix} \hat{\mathtt{A}}_{11} & \hat{\mathtt{A}}_{12} \\ \mathtt{I} & 0 \end{bmatrix} + u_0 \begin{bmatrix} 0 & 0 \\ \mathtt{B}_{21} & \mathtt{B}_{22} \end{bmatrix}. \tag{51}$$

Substituting the value of $\mathtt{M}(u_0)$, in (8), we get $\hat{\mathtt{A}}_{11}\mathbf{b}_1 + \hat{\mathtt{A}}_{12}\mathbf{b}_2 = \mathbf{0}$ and $u_0(\mathtt{B}_{21}\mathbf{b}_1 + \mathtt{B}_{22}\mathbf{b}_2) + \mathbf{b}_1 = \mathbf{0}$. Eliminating

$\mathbf{b}_2$ from these equations, we obtain an alternate eigenvalue formulation

$$\underbrace{(\mathtt{B}_{21} - \mathtt{B}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11})}_{\mathtt{X}}\,\mathbf{b}_1 = -(1/u_0)\mathbf{b}_1. \tag{52}$$

□

The matrix $\mathtt{X}$ here, represents an alternative representation to the one in (50), of the Schur complement.

Note that this proposition allows us to test the existence of the Schur complement $\mathtt{X}$ (50) (or (52)), for a given favourable monomial set B and the corresponding coefficient matrix $\mathtt{C}(u_0)$. This condition is tested for each B and corresponding $\mathtt{C}(u_0)$, for both choices of set-partitions (33) or (32). Out of those that succeed, we select the coefficient matrix $\mathtt{C}(u_0)$, which led to the smallest possible Schur complement $\mathtt{X}$. If we have more than one such choice, we choose the smallest coefficient matrix $\mathtt{C}(u_0)$. Note that our iterative approach is crucial in increasing the chances of obtaining a minimal solver, even when the polynomials in $\mathcal{F}$ have non-generic coefficients.

3.4 Row-Column Removal

The next step in our method is to attempt to reduce the size of the matrix $\mathtt{C}(u_0)$ selected in the previous step. For this, we employ a method, inspired by [26]. Here we select columns of $\mathtt{C}(u_0)$, one by one in a random order to test for their removal. For each such column, we select rows (say $r_1, \ldots, r_s$) that contain nonzero entries in the column and also consider all columns (say $c_1, \ldots, c_l$) that have nonzero entries in $r_1, \ldots, r_s$. Then, we can remove these s rows and l columns from $\mathtt{C}(u_0)$ only if the following conditions hold true for the resulting reduced matrix $\mathtt{C}_{\text{red}}(u_0)$.

1. After eliminating the monomials from T, we require that there is at least one monomial left in each T_i.
2. If $\mathtt{C}(u_0)$ is of size $p \times \varepsilon$, the reduced matrix $\mathtt{C}_{\text{red}}(u_0)$ would be of size $(p-s) \times (\varepsilon - l)$. Then, we require $p - s \geq \varepsilon - l$ and $\text{rank}(\mathtt{C}_{\text{red}}(u_0)) = \varepsilon - l$.
3. $\mathtt{C}_{\text{red}}(u_0)$ must be block partitioned and decomposed as in Proposition 1.

We repeat the above process until there are no more columns that can be removed. Note that the last condition is important as it ensures that at each stage, the reduced matrix can still be partitioned and decomposed into an eigenvalue formulation (50) (or alternately (52)). Now by abuse of notation, let us denote $\mathtt{C}(u_0)$ to be the reduced matrix and denote B and T to be the reduced set of monomials and the set of monomial multiples, respectively.

If $\mathtt{C}(u_0)$ still has more rows than columns, we transform it into a square matrix by removing extra rows (say $q_1, \ldots, q_j$) and the monomials from T indexing these rows, chosen in a way such that the three conditions mentioned above are still satisfied. Note that it is always possible to choose rows, such that these three conditions are satisfied. Moreover, our proposed approach first tries to remove as many rows as possible from the lower block, indexed by T_{m+1}. This is to reduce $|T_{m+1}|(= |B_1|)$ as much as possible and ensure that the matrix $\mathtt{A}_{21}$ and hence $\mathtt{X}$ (50) (or (52)) for eigenvalue problem has as small size as possible. Then, if there are more rows still to be removed, the rest are randomly chosen from the upper block indexed by $\{T_1, \ldots, T_m\}$. Algorithms for this step of matrix reduction are provided in "Appendix B".

The sparse resultant matrix $\mathtt{M}(u_0)$ is constructed offline through these three steps. In the online stage, we fill in the coefficients of $\mathtt{M}(u_0)$ from the input measurements and compute a Schur complement $\mathtt{X}$ (50) (or (52)). Then, we extract the solutions to $x_1, \ldots, x_n$ by computing the eigenvectors of $\mathtt{X}$. The speed of execution of the solver depends on the size of $\mathbf{b}_1(=|B_1|)$ as well the size of $\hat{\mathtt{A}}_{12}$, while the accuracy of the solver largely depends on the condition number as well as the size of the matrix to be inverted i.e., $\hat{\mathtt{A}}_{12}$.

4 Action Matrix vs sparse resultant

In this section, we study the connection between a solver generated using an action matrix method [8, 9, 26, 27, 32, 35, 47], and a solver based on the sparse resultant method proposed in Sect. 3. Observe that, our sparse resultant method exploits only the structure of the input polynomials, via the geometry of their Newton polytopes, whereas the SOTA action matrix method algebraically investigates the input polynomial system via the Gröbner basis. Seemingly different, the generated solvers exhibit the same properties, and hence, in this section our objective is to throw some light on the similarities. Some of the symbols used in both the methods are the same. Therefore, to distinguish them, in this section we will use the prefixes ^{a} and ^{r} to, respectively, denote the symbols used in the action matrix method in Sect. 2.1 and the sparse resultant method in Sect. 3.1. Let us assume that both these methods are used to generate a solver for a system $\mathcal{F} = 0$ (1) of m polynomial equations in n variables with r solutions. An action matrix method results in a solver that is performing G–J elimination (or alternatively LU/QR decomposition) of an elimination template $^a\mathtt{C}$ (A8) and the eigendecomposition of an action matrix $\mathtt{M}_f$ (A3) of size $r \times r$.[4] The proposed sparse resultant method, on the other hand, leads to a solver, which computes a Schur complement $\mathtt{X}$ (50) (or (52)) of size

[4] The size of the action matrix in all state-of-the art action matrix methods [9, 26, 27, 32, 35, 47] is equal to the number of solutions to the input system. The only exceptions are the methods proposed in [8] and [37], which can result in larger action matrices.

$N \times N$, from the sparse resultant matrix $\mathtt{M}(u_0)$, followed by its eigendecomposition. In general, $N \geq r$.

While the final solvers generated by these two methods are in general different, i.e., they perform different operations, at the same time, they demonstrate some similar properties, e.g., both these solvers extract solutions from eigenvectors of some matrices, and both involve computing a matrix inverse (G–J elimination can be replaced by matrix inverse). This motivates us to study the similarities of these solvers. Let us first define the *equivalence* of a solver generated using an action matrix method (**AM**) and a solver generated using the sparse resultant method from Sect. 3 (**SRes**).

Definition 3 For a given system of polynomial equations $\mathcal{F} = 0$ (1), an action matrix-based solver and a sparse resultant-based solver are defined to be equivalent, iff the following two conditions are satisfied:

1. The action matrix $\mathtt{M}_f$ (A3) is the same as the Schur complement $\mathtt{X}$ (50) (or (52)), i.e. $\mathtt{M}_f = \mathtt{X}$.
2. The size of the matrix ${}^a\hat{\mathtt{C}}$ is the same as the size of the upper block $\begin{bmatrix}\hat{\mathtt{A}}_{11} & \hat{\mathtt{A}}_{12}\end{bmatrix}$ of the sparse resultant matrix $\mathtt{M}(u_0)$ (49) (or (51)) and the step of G–J elimination of the template matrix ${}^a\hat{\mathtt{C}}$ in the online stage (step 7) of the action matrix method (Sect. 2.1) can be replaced with the matrix product $\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$, used to compute the Schur complement in the online stage (step 6) of our sparse resultant method (Sect. 3.1), or vice-versa.

Now, our goal is to define conditions under which the final online solvers generated by these two methods are *equivalent*.

We next demonstrate that if we have an action matrix-based solver (**AM**) for a given system $\mathcal{F} = 0$ (1), we can modify the steps performed by the sparse resultant method proposed in the Sect. 3.1, such that both the solvers are equivalent according to Definition 3.

4.1 $\mathtt{X}$ (SRes) from $\mathtt{M}_f$ (AM)

Let us assume that we have generated a solver for a system of polynomial equations, $\mathcal{F} = 0$ (1) using the action matrix method described in Sect. 2.1 and detailed in "Appendix A". The action polynomial is assumed to be $f = x_1$, where $x_1 \in X$.

We first propose the following changes to the steps performed by the sparse resultant method in the offline stage, in Sect. 3.1. In what follows, we will assume that step i actually means the step i in Sect. 3.1.

C1R The step 1 is skipped, as there is no need to test each variable $x_k \in X$.

C2R In the step 2, the extra polynomial is assumed to be $f_{m+1} = x_1 - u_0$.

C3R In the step $3(a)$, the favourable set of monomials ${}^rB$ is directly constructed from ${}^aB$, as ${}^rB = {}^aB$. Moreover, the set of monomial multiples is constructed as ${}^rT_i = {}^aT_i$, $i = 1, \ldots, m$ and ${}^rT_{m+1} = B_a$ (see Eq. (A5)). The step $3(b)$ is to be skipped, as the coefficient matrix ${}^r\mathtt{C}(u_0)$ will be directly obtained from the action matrix-based solver, in subsequent steps. The output of this step then is the favourable monomial set ${}^rB$.

C4R Replace the step $4(a)$ by directly determining the set partition of ${}^rB = B_1 \cup B_2$ (31)–(32), as $B_1 = B_a$ and $B_2 = \hat{B}_e \cup B_r$, where B_a, B_r and $\hat{B}_e$ are defined in "Appendix A". Moreover, the monomial ordering in ${}^a\mathbf{b}$ determines the vectors, $\mathbf{b}_1 = \mathbf{b}_a$ and $\mathbf{b}_2 = \begin{bmatrix}\hat{\mathbf{b}}_e \\ \mathbf{b}_r\end{bmatrix}$. The action matrix-based solver corresponds to the first version of set partition of ${}^rB$, considered in our sparse resultant method, i.e. the set partition as in (31).

C5R In the step $4(b)$, the upper block $\begin{bmatrix}\mathtt{A}_{11} & \mathtt{A}_{12}\end{bmatrix}$ of ${}^r\mathtt{C}(u_0)$ (33) is directly obtained from ${}^a\hat{\mathtt{C}}$, as

$$\mathtt{A}_{11} = \begin{bmatrix}\mathtt{C}_{13} \\ \mathtt{C}_{23}\end{bmatrix}, \quad \mathtt{A}_{12} = \begin{bmatrix}\hat{\mathtt{C}}_{11} & \mathtt{C}_{12} \\ \hat{\mathtt{C}}_{21} & \mathtt{C}_{22}\end{bmatrix}. \tag{53}$$

Multiplying f_{m+1} with the monomials in ${}^rT_{m+1}$, the expanded set of polynomials is obtained as $\{\mathbf{x}^\alpha\,(x_1 - u_0) \mid \mathbf{x}^\alpha \in {}^rT_{m+1}\}$. It is possible to order the polynomials in this set, such that its matrix form is

$$\begin{bmatrix}\mathtt{A}_{21} - u_0\mathtt{I} & \mathtt{A}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{b}_1 \\ \mathbf{b}_2\end{bmatrix}. \tag{54}$$

This determines the lower block of ${}^r\mathtt{C}(u_0)$. Here, $\mathtt{A}_{21}$ and $\mathtt{A}_{22}$ are binary matrices, with entries either 0 or 1. The upper and the lower blocks together give us

$${}^r\mathtt{C}(u_0)\,{}^r\mathbf{b} = \begin{bmatrix}\mathtt{A}_{11} & \mathtt{A}_{12} \\ \mathtt{A}_{21} - u_0\mathtt{I} & \mathtt{A}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{b}_1 \\ \mathbf{b}_2\end{bmatrix}. \tag{55}$$

The output of the modified step 4, is the coefficient matrix ${}^r\mathtt{C}(u_0)$.

C6R By construction, ${}^r\mathtt{C}(u_0)$ is already a square invertible matrix and has no extra rows to be removed. As we do not need to attempt row-column removal from ${}^r\mathtt{C}(u_0)$, the step 5 is to be skipped, to directly construct the sparse resultant matrix as $\mathtt{M}(u_0) = {}^r\mathtt{C}(u_0)$. In this case, the submatrices $\hat{\mathtt{A}}_{11}$ and $\hat{\mathtt{A}}_{12}$ of $\mathtt{M}(u_0)$ are equal to the submatrices $\mathtt{A}_{11}$ and $\mathtt{A}_{12}$ of ${}^r\mathtt{C}(u_0)$, respectively.

Applying these changes to the steps 1–5 in Sect. 3.1, i.e. the offline stage, we obtain the sparse resultant matrix $\mathtt{M}(u_0)$. After that, the online solver computes the Schur complement $\mathtt{X}$ of the block submatrix $\hat{\mathtt{A}}_{12}$ of $\mathtt{M}(u_0)$, followed by an eigenvalue decomposition (50) of $\mathtt{X}$ as described in the step 6 in Sect. 3.1. In the proposition C.1, we prove that the sparse resultant-based solver so obtained is equivalent to the action matrix-based solver.

4.2 $\mathtt{M}_f$ (AM) from $\mathtt{X}$ (SRes)

Let us assume that for a system of polynomial equations, $\mathcal{F} = 0$ (1), we have a sparse resultant-based solver, generated by following the steps in Sect. 3.1. In this solver the coefficient matrix ${}^r\mathtt{C}(u_0)$ is assumed to be partitioned as in (33), and the alternative partition (34) will be considered in the next Sect. 4.3. Moreover, the Schur complement $\mathtt{X}$ is assumed to have as many eigenvalues as the number of solutions to $\mathcal{F} = 0$, i.e., $N = r$.[5] The extra polynomial is supposed to be of the form $f_{m+1} = x_1 - u_0$, for $x_1 \in X$.

We first propose the following changes to the steps performed by the action matrix method in the offline stage, in "Appendix A". In the following list, we assume that step i actually means the step i in "Appendix A".

C1A There is no change to the step 1.

C2A In the step 2, set the basis of the quotient ring A, to be $\mathcal{B}_A = \{[\mathbf{x}^\alpha] \mid \mathbf{x}^\alpha \in B_1\} \subset A$. By assumption, A is an r-dimensional vector space over $\mathbb{C}$ spanned by the elements of $\mathcal{B}_A$,[6]. Construct the monomial set $B_a = B_1$.

C3A In the step 3, assume the action polynomial $f = x_1$.

C4A The monomial multiples, required in the step 4, are obtained as ${}^aT_i = {}^rT_i, i = 1, \ldots, m$. This also gives us the extended polynomial set $\mathcal{F}'$. Note that ${}^aB = {}^rB = \mathrm{mon}(\mathcal{F}')$.

C5A In the step 5, the required sets of reducible and excess monomials are determined as

$$\begin{aligned} B_r &= \{x_1\mathbf{x}^\alpha \mid \mathbf{x}^\alpha \in B_1\} \setminus B_1 \\ B_e &= B_2 \setminus B_r. \end{aligned} \tag{56}$$

Note that by (45), $\mathbf{x}^\alpha \in B_1 \implies x_1\mathbf{x}^\alpha \in {}^rB$.

C6A In the step 6, the vector of monomials is determined as ${}^a\hat{\mathbf{b}} = {}^r\mathbf{b}$. This will also determine the vectors $\mathbf{b}_a$, $\mathbf{b}_r$ and $\hat{\mathbf{b}}_e$. Thus, the monomials in the action matrix method are ordered in the same way as the monomials in the sparse resultant method.
Moreover, in the step 6, the elimination template ${}^a\hat{\mathtt{C}}$, and its block partition, are determined as

$$\begin{aligned} \begin{bmatrix} \hat{\mathtt{C}}_{11} & \mathtt{C}_{12} \\ \hat{\mathtt{C}}_{21} & \mathtt{C}_{22} \end{bmatrix} &= \hat{\mathtt{A}}_{12}, \\ \begin{bmatrix} \mathtt{C}_{13} \\ \mathtt{C}_{23} \end{bmatrix} &= \hat{\mathtt{A}}_{11}. \end{aligned} \tag{57}$$

The column partition of ${}^a\hat{\mathtt{C}}$ is determined by the partition of the vector ${}^a\mathbf{b}$. By the construction of $\mathtt{M}(u_0)$, $\hat{\mathtt{A}}_{12}$ is a square invertible matrix. Therefore, we can always find a row partition such that $\mathtt{C}_{22}$ is a square invertible matrix. As, $\hat{\mathtt{C}}_{22}$ and $\hat{\mathtt{A}}_{12}$ are square matrices, $\hat{\mathtt{C}}_{11}$ is also a square matrix. Therefore, there are no extra columns to be removed and the elimination template ${}^a\hat{\mathtt{C}}$ will be such that its G–J elimination will lead to the required form, as in (A11).

Applying these changes to the steps 1–6 in Sect. 2.1, i.e. the offline stage, we obtain the elimination template ${}^a\hat{\mathtt{C}}$. Subsequently, the online action matrix-based solver (step 7 in Sect. 2.1) computes the G–J elimination of ${}^a\hat{\mathtt{C}}$, from which the entries of the action matrix $\mathtt{M}_f$ are read-off. This implies that the action matrix-based solver is equivalent to the sparse resultant-based solver, which is proved in Proposition C.2.

4.3 $\mathtt{M}_f$ (AM) from $\mathtt{X}$ (SRes) with Alternative Form

Let us assume that for a system of polynomial equations, $\mathcal{F} = 0$ (1), we have used the alternative partition of ${}^rB$ (32) and ${}^r\mathtt{C}(u_0)$ (34), and generated a sparse resultant-based solver by following the steps in Sect. 3.1. This means that we need to assume that no solution of $\mathcal{F} = 0$, has $x_1 = 0$. The other assumptions remain the same as in Sect. 4.2.

The alternative partition of the favourable monomial set ${}^rB$ (32) implies that the Schur complement $\mathtt{X}$ (52) gives us a representation of each coset $\left[\frac{\mathbf{x}^{\alpha_\mathbf{j}}}{x_1}\right], \mathbf{x}^{\alpha_\mathbf{j}} \in B_1$ as a linear combination of the cosets $[\mathbf{x}^{\alpha_\mathbf{j}}], \forall \mathbf{x}^{\alpha_\mathbf{j}} \in B_1$. However, the approach in Sect. 4.2 does not work, because in this case we need to set the action polynomial f to be $1/x_1$.

Therefore, we can use the so-called *Rabinowitsch trick* [15], and propose the following changes to the offline stage of the action matrix method in Sect. 2.1. In the following list, we will assume that step i actually means the step i in Sect. 2.1.

[5] If the Schur complement $\mathtt{X}$ has more eigenvalues N than the number of solutions r, we can still change the steps performed by the action matrix method such that it leads to a solver equivalent to the sparse resultant-based solver. However, such a case would be too complicated, and is not discussed here. Recently, [37] have proposed an action matrix-based automatic generator which attempts to generate solvers with an eigenvalue problem larger than the number of roots of the system.

[6] It was proved in [15, Theorem 6.17] that if $\mathcal{F}$ is a generic polynomial system, then A would be spanned by the elements of the set $\mathcal{B}_A$. However, even if the polynomial system is not generic, we can still follow the steps performed in the same proof.

C1A′ In the step 1, the polynomial system $\mathcal{F}$, is augmented with an extra polynomial ${}^{a}f_{m+1} = x_1\lambda - 1$, where λ is a new variable. The augmented polynomial system is denoted as ${}^{a}\mathcal{F}_a$, and the augmented set of variables as ${}^{a}X_a = X \cup \{\lambda\}$. Consider the ideal $I_a = \langle {}^{a}\mathcal{F}_a \rangle$ and the quotient ring $A_a = \mathbb{C}[{}^{a}X_a]/I_a$.

C2A′ Note that the number of solutions to $\mathcal{F} = 0$ is the same as that of ${}^{a}\mathcal{F}_a = 0$, because we have assumed $x_1 \neq 0$. Therefore, A_a is an r-dimensional vector space over $\mathbb{C}$. Its basis is constructed as $\mathcal{B}_A = \{[\mathbf{x}^{\alpha_j}] \mid \mathbf{x}^{\alpha_j} \in B_1\}$. Construct the monomial set $B_a = B_1$.

C3A′ In the step 3, the action polynomial is assumed to be $f = \lambda$.

C4A′ The sets of monomial multiples required in the step 4, are constructed as ${}^{a}T_i = {}^{r}T_i, i = 1, \ldots, m+1$. From (32), $\mathbf{x}^{\alpha_j} \in T_{m+1}$ implies $x_1\mathbf{x}^{\alpha_j} \in B_1$, which implies that $\lambda x_1 \mathbf{x}^{\alpha_j} \in B_r$. The extended polynomial set ${}^{a}\mathcal{F}'_a$ is obtained by multiplying each $f_i \in {}^{a}\mathcal{F}_a$ with the monomials in ${}^{a}T_i$. In the step 4, the set of monomials is obtained ${}^{a}B = \text{mon}({}^{a}\mathcal{F}'_a)$.

C5A′ In the step 5, the required monomial sets are

$$\begin{aligned} B_r &= \{\lambda \mathbf{x}^{\alpha} \mid \mathbf{x}^{\alpha} \in B_1\} \\ B_e &= B_2. \end{aligned} \tag{58}$$

Note that the set of monomials ${}^{a}B = B_e \cup B_r \cup B_a$, and $B_a \cap B_r = \emptyset$.

C6A′ In the step 6, the monomial vectors are set as $\mathbf{b}_a = \mathbf{b}_1, \mathbf{b}_r = \text{vec}(B_r)$ and $\mathbf{b}_e = \mathbf{b}_2$. This will also fix the vector of monomials ${}^{a}\mathbf{b} = \begin{bmatrix} \mathbf{b}_e \\ \mathbf{b}_r \\ \mathbf{b}_a \end{bmatrix}$. The monomials in the vector ${}^{a}\mathbf{b}$ in the action matrix method are ordered in the same way as the monomials in the sparse resultant method in Sect. 3.1. Moreover, as the monomials in B_r are in a one-to-one correspondence w.r.t. the monomials in B_1, the monomial ordering in $\mathbf{b}_1$ fixes the order of the monomials in $\mathbf{b}_r$ as well.

Applying these changes to the steps 1–6 in Sect. 2.1, i.e. the offline stage, we obtain the elimination template ${}^{a}\hat{\mathtt{C}}$. Then, the action matrix-based solver (step 7 in Sect. 2.1) computes the G–J elimination of ${}^{a}\hat{\mathtt{C}}$, from which the entries of the action matrix $\mathtt{M}_f$ are read-off. This implies that the action matrix-based solver is equivalent to the sparse resultant-based solver, which is proved in Proposition C.3.

5 Comparison w.r.t. SOTA Sparse Resultant Methods

Our extra polynomial algorithm is based on the hidden variable method [21]. It also extends the method in [11] for square systems, by doing away with the computation of the mixed subdivision of the Minkowski sum of all the Newton polytopes. We consider these two methods to be the current SOTA for generating sparse resultant-based minimal solvers.

We select a subset of interesting minimal problems and demonstrate the importance of our algorithm in improving solver speeds compared to those generated by the SOTA resultant methods. Moreover, our algorithm has two important properties, viz. the special form of the extra polynomial and the row-column removal procedure. In this section, we demonstrate via an ablation study, how these two properties are critical in achieving substantially improved solver sizes. Refer to Table 1, where we list the sizes of the solvers generated using the SOTA sparse resultant methods [11, 21] (**cols 3** & **4**), the sizes of the solvers generated with an extra polynomial of a general form (see Sect. 2.2.3) (**col 5**), and the sizes of the solvers generated without the row-column removal step (see Sect. B.2) (**col 6**). Finally, we list the sizes of the solvers generated by our approach (**col 7**).

Note that the size of a solver based on the hidden variable method [21] is reported as the size of the underlying generalized eigenvalue problem (GEP) (14). If the matrix $\mathtt{A}$ (and $\mathtt{B}$) in GEP is of dimensions $r \times r$, the solver size can be considered to be equivalent to the size of an inverse of a matrix of dimensions $r \times r$ and an eigenvalue computation of a matrix of size $r \times r$. On the other hand, the size of the solver based on any one of the other methods is reported as a matrix of size $s \times t$, which is equivalent to the size of an inverse of a matrix of dimensions $s \times s$ and an eigenvalue computation of a matrix of size $(t-s) \times (t-s)$. The difference in the time taken for inverting matrices of comparable sizes is minute, as the sparsity of the matrices enable highly optimized software implementation for matrix inverse. Therefore the speed of the solver is primarily governed by the size of the eigenvalue computation. With this discussion in mind, we observe from Table 1 that the solvers based on the GEP (**col 1**) will have worse or comparable speed to that of the solvers based on our approach (**col 7**). Observe that out of the five problems the approach by [11] failed to generate any solver as the systems are non-square, and hence, the subdivision methods cannot be applied. For all the problems, the method of u-resultant using an extra polynomial of a general form leads to larger elimination templates, whereas using an extra polynomial with a special form improves the template size. Additionally, the row-column removal step ultimately leads to the best possible solver sizes among all sparse resultant-based methods.

6 Experiments

In this section, we evaluate the performance of the minimal solvers, generated using our sparse resultant method,

Table 1 A comparison of the sizes of the solvers for some minimal problems, generated by the iterative algorithm [21] (**col 3**), the polytope subdivision approach for square systems in [11] (**col 4**), the u-resultant method using an extra polynomial with a generic form (**col 5**), our proposed algorithm (see Sect. 3) without (**col 6**) and with row-column removal (**col 7**)

#	Problem	[21] **GEP**	[11]	Our		
				u-resultant	No row-col removal	Full algorithm
1	Rel. pose F+λ 8pt	12×12	15×24	15×24	$\mathbf{7 \times 16}$	$\mathbf{7 \times 16}$
2	Rel. pose E+λ 6pt	30×30	–	42×60	$\mathbf{14 \times 40}$	$\mathbf{14 \times 40}$
3	Stitching $f\lambda$+R+$f\lambda$ 3pt	24×24	6×30	31×49	18×36	$\mathbf{8 \times 31}$
4	Rel. pose λ_1+F+λ_2 9pt	68×68	210×240	–	120×147	$\mathbf{77 \times 104}$
5	Abs. pose quivers	43×43	104×161	–	69×94	$\mathbf{36 \times 70}$
6	Abs. pose refractive P5P	36×36	49×73	61×77	61×77	$\mathbf{39 \times 65}$

Note that the solvers in **col 3** consist of **GEP** (14). E.g., a solver of size 68×68 can be considered to be competing with a solver having two matrix operations, viz. matrix inverse of size 68×68 and matrix eigenvalue of size 69×68. –: The algorithm failed to generate any solver. The best solver sizes are highlighted in bold

proposed in Sect. 3. We compare the stability as well as the computational complexities of these solvers with the state-of-art Gröbner basis-based solvers for many interesting minimal problems. The minimal problems selected for comparison represent a huge variety of relative and absolute pose problems and correspond to those studied in [35]. Note that for the two minimal problems, Rel. pose f+E+f 6pt and Abs. pose refractive P5P, we generated solvers from a simplified polynomial system as described in [38], instead of the original formulation.

6.1 Computational Complexity

The biggest contributor towards computational complexity of a minimal solver is the size of the matrices that undergo crucial numerical operations. The solvers based on our sparse resultant method in Sect. 3, the state-of-the-art Gröbner basis solvers as well as in the original solvers proposed by the respective authors (see column 4) involve two crucial operations, a *matrix inverse*[7] and an *eigenvalue decomposition*. This is indicated by the size of the solver in the table, e.g. a sparse resultant-based solver of size 11×20 means computing a matrix inverse $\hat{\mathtt{A}}_{12}^{-1}$ of size 11×11, ultimately leading to a Schur complement $\mathtt{X}$ of size $20 - 11 = 9$, and an eigenvalue decomposition of this matrix $\mathtt{X}$. Similarly an action matrix-based solver of size 11×20 means performing a G–J elimination of a 11×20 matrix $\mathtt{C}$ and then an eigenvalue decomposition of an action matrix $\mathtt{M}_f$ of size $20-11 = 9$. So in Table 2, we compare the size of the upper block $\begin{bmatrix}\hat{\mathtt{A}}_{11} & \hat{\mathtt{A}}_{12}\end{bmatrix}$ of the sparse resultant matrix $\mathtt{M}(u_0)$ in our extra polynomial sparse resultant-based solvers, with the size of the elimination template $\mathtt{C}$ used in state-of-the-art Gröbner basis solvers as well as in the original solvers proposed by the respective authors (column 5).

The Gröbner basis-based solvers used for comparison include the solvers generated using the approach in [32] (column 6), the Gröbner fan-based and the heuristic-based approaches in [35] (columns 7 and 9, respectively), and the template optimization approach using a greedy parameter search in [38] (column 10). Out of the two methods for generating minimal solvers, proposed in [38], we consider the smallest solver for each minimal problem.

As we can see from Table. 2, for most minimal problems, our sparse resultant method leads to smaller solvers compared to the Gröbner basis solvers based on [32, 35] and of exactly the same size as the solvers based on [38]. The smallest solvers where size of the eigenvalue decomposition problem is the same as the number of solutions are written in bold in Table 2.

Moreover, for some minimal problems, our sparse resultant method leads to a larger eigenvalue problem than the number of solutions, written in italic in Table 2. For three such minimal problems, i.e. Rel. pose E+$f\lambda$ 7pt, Unsynch. Rel. pose, and Optimal pose 2pt v2, our sparse resultant method leads to solvers with the smaller size as compared to the solvers based on the Gröbner basis-based methods [32, 35], whereas for the problem, Abs. pose quivers, our sparse resultant method leads to a smaller solver than the solvers based on [32, 35] and of comparable size w.r.t. the solver based on [38].

Note that for the two minimal problems, Optimal pose 2pt v2 and Rel. pose E angle+4pt, the elimination template $\mathtt{C}$ in the solvers based on the approach in [38], underwent an extra Schur complement-based step in the offline stage. Therefore, the template sizes reported for these two solvers are smaller than those in the solvers based on the other Gröbner basis methods in [32, 35]. For the problem, Rel. pose E angle+4pt, our sparse resultant method failed to generate a solver. This implies that none of the subsets $\mathcal{F}_{\text{sub}}$ for this problem lead to a favourable monomial set (see Sect. 3.2). The primary reason for this failure is the non-genericity of the coefficients (see

[7] G–J elimination is usually performed by a computing a matrix inverse.

Table 2 Size comparison of solvers based on [32] (**col 6**), Gröbner fan [35] (**col 7**), heuristic sampling [35] (**col 9**) and greedy parameter search [38] (**col 10**)

#	Problem	Our	Author	Original	Syzygy	GFan	(#GB)	Heuristic	Greedy
1	Rel. pose F+λ 8pt(‡) (8 sols.)	*7 × 16*	[25]	12 × 24	11 × 19	11 × 19	(10)	**7 × 15**	**7 × 15**
2	Rel. pose E+f 6pt (9 sols.)	**11 × 20**	[6]	21 × 30	19 × 28	**11 × 20**	(66)	**11 × 20**	**11 × 20**
3	Rel. pose f+E+f 6pt* (15 sols.)	**11 × 26**	[27]	31 × 46	25 × 40	31 × 46	(218)	25 × 40	**11 × 26**
4	Rel. pose E+λ 6pt (26 sols.)	**14 × 40**	[25]	48 × 70	34 × 60	34 × 60	(846)	**14 × 40**	**14 × 40**
5	Stitching fλ+R+fλ 3pt (18 sols.)	*8× 31*	[42]	54 × 77	48 × 66	48 × 66	(26)	**18 × 36**	**18 × 36**
6	Abs. Pose P4Pfr (16 sols.)	**52 × 68**	[7]	136 × 152	140 × 156	54 × 70	(1745)	54 × 70	**52 × 68**
7	Abs. Pose P4Pfr (elim. f) (12 sols.)	**28 × 40**	[34]	**28 × 40**	58 × 70	**28 × 40**	(699)	**28 × 40**	**28 × 40**
8	Rel. pose λ+E+λ 6pt(‡) (52 sols.)	*39 × 95*	[27]	238 × 290	154 × 206	–	?	53 × 105	*39 × 95*
9	Rel. pose λ_1+F+λ_2 9pt (24 sols.)	*77 × 104*	[27]	179 × 203	165 × 189	105 × 129	(6896)	**73 × 97**	76 × 100
10	Rel. pose E+fλ 7pt (19 sols.)	*34 × 56*	[25]	200 × 231	181 × 200	69 × 88	(3190)	65 × 84	55 × 74
11	Rel. pose E+fλ 7pt (elim. λ) (19 sols.)	**22 × 41**	–	52 × 71	37 × 56	(332)	24 × 43	**22 × 41**	**22 × 41**
12	Rel. pose E+fλ 7pt (elim. fλ) (19 sols.)	**51 × 70**	[30]	**51 × 70**	**51 × 70**	**51 × 70**	(3416)	**51 × 70**	**51 × 70**
13	Rolling shutter pose (8 sols.)	*40 × 52*	[46]	48 × 56	**47 × 55**	**47 × 55**	(520)	**47 × 55**	**47 × 55**
14	Triangulation from satellite im. (27 sols.)	**87 × 114**	[57]	93 × 120	88 × 115	88 × 115	(837)	88 × 115	**87 × 114**
15	Abs. pose refractive P5P* (16 sols.)	*39 × 65*	[19]	280 × 399	57 × 73	57 × 73	(8659)	57 × 73	57 × 73
16	Abs. pose quivers (20 sols.)	*36 × 70*	[24]	372 × 386	156 × 185	–	?	68 × 88	**65 × 85**
17	Unsynch. Rel. pose(‡) (16 sols.)	*59 × 80*	[1]	633 × 649	159 × 175	–	?	139 × 155	139 × 155
18	Optimal PnP (Hesch) (27 sols.)	**87 × 114**	[22]	93 × 120	88 × 115	88 × 115	(837)	88 × 115	**87 × 114**
19	Optimal PnP (Cayley) (40 sols.)	**118 × 158**	[41]	**118 × 158**	**118 × 158**	**118 × 158**	(2244)	**118 × 158**	**118 × 158**
20	Optimal pose 2pt v2 (24 sols.)	*112 × 146*	[53]	192 × 216	192 × 216	–	?	192 × 216	139 × 163 †
21	Rel. pose E angle+4pt (20 sols.)	–	[36]	270 × 290	266 × 286	–	?	183 × 203	99 × 119†

The smallest solver sizes are written in bold, and the solver sizes with larger eigenvalue problem than the number of solutions is written in italic. Missing entries are when we failed to generate a solver. (∗): Using a simplified input system of polynomials, as in [38]. (‡): Solved using the alternate eigenvalue formulation (52). (†): In the offline stage in [38], the elimination template was reduced using a Schur complement

Definition 1). This points to a future research direction for such minimal problems. We need to look beyond Newton Polytopes of the subsets $\mathcal{F}_{\text{sub}}$ and test other convex objects.

6.2 Stability Comparison

We evaluate and compare the stability of our solvers from Table 2 with the Gröbner basis-based solvers. As it is not feasible to generate scene setups for all considered problems, we instead evaluated the stability of minimal solvers using 5K instances of random data points. Stability measures include *mean* and *median* of Log_{10} of the normalized equation residuals for computed solutions, as well as the solver failures. The solver failures are an important measure of stability for real-world applications and are computed as the % of 5K instances for which at least one solution has a normalized equation residual $> 10^{-3}$. These measures on randomly generated inputs have been shown to be sufficiently good indicators of solver stability [32]. Also, Table 3 shows the stability of the solvers for all the minimal problems from Table 2. We observe that for most of the minimal problems, our proposed sparse resultant-based solvers have no failures. Among those with some failures, for all except two minimal problems our sparse resultant-based solvers have fewer failures, as compared to the Gröbner basis-based solvers.

In general, our new method generates solvers that are stable with only very few failures. For a selected subset of nine minimal problems from Table 2, we computed the Log_{10} of the normalized equation residuals and depicted their histograms in Fig. 4. The histograms agree with our observation from Table 3, that our proposed sparse resultant method leads to solvers with only few failures.

Note that as our new solvers are solving the same formulations of problems as the existing state-of-the-art solvers, the performance on noisy measurements and real data would be the same as the performance of the state-of-the-art solvers. The only difference in the performance comes from numerical instabilities that already appear in the noise-less case and are detailed in Table 3 (fail%). For performance of the solvers in real applications, we refer the reader to papers where the original formulations of the studied problems were presented (see Table 2, column 2). Here we select two interesting problems, i.e., one relative and one absolute pose problem, and perform experiments on synthetically generated scenes and on real images, respectively.

E+ $f\lambda$ solver on synthetic scenes We study the numerical stability of our sparse resultant-based solver for the problem of estimating the relative pose of one calibrated camera, and one camera with unknown focal length and radial distortion from 7-point correspondences, i.e. the Rel. pose E+$f\lambda$ 7pt problem from Table 2. We consider the formulation "elim. λ" proposed in [35] that leads to the smallest solvers. We study the performance on noise-free data and compare it to the results of Gröbner basis solvers from Table 2.

We generated 10K scenes with 3D points drawn uniformly from a $[-10, 10]^3$ cube. Each 3D point was projected by two cameras with random feasible orientation and position. The focal length of the first camera was randomly drawn from the interval $f_{gt} \in [0.5, 2.5]$ and the focal length of the second camera was set to 1, i.e. the second camera was calibrated. The image points in the first camera were corrupted by radial distortion following the one-parameter division model. The radial distortion parameter λ_{gt} was drawn at random from the interval $[-0.7, 0]$ representing distortions of cameras with a small distortion up to slightly more than GoPro-style cameras.

Figure 3 shows Log_{10} of the relative error of the distortion parameter λ (left) and the focal length f (right), obtained by selecting the real root closest to the ground truth. All tested solvers provide stable results with only a small number of runs with larger errors. The solver based on our sparse resultant method (green) is not only smaller, but also slightly more stable than the heuristic-based solver [35] (red) and the solver based on [32] (black). For most minimal problems, the solver based on our proposed sparse resultant method has fewer failures, as compared to the solver based on the greedy parameter approach in [38] (blue) (Fig. 4).

P4Pfr solver on real images We evaluated our sparse resultant-based solver for a practical problem of estimating the absolute pose of camera with unknown focal length and radial distortion from four 2D-to-3D point correspondences, i.e. the P4Pfr solver, on real data. We consider the *Rotunda* dataset, which was proposed in [29], and in [34] it was used for evaluating P4Pfr solvers. This dataset consists of 62 images captured by a GoPro Hero4 camera. Example of an input image from this dataset (left) as well as undistorted (middle) and registered image (right) using our new solver, is shown in Fig. 5 (top). The Reality Capture software [44] was used to build a 3D reconstructions of this scene. We used the 3D model to estimate the pose of each image using the new P4Pfr resultant-based solver (28×40) in a RANSAC framework. Similar to [34], we used the camera and distortion parameters obtained from [44] as ground truth for the experiment. Figure 5 (bottom) shows the errors for the focal length, radial distortion, and the camera pose. Overall, the errors are quite small, e.g. most of the focal lengths are within 0.1% of the ground truth and almost all rotation errors are less than 0.1 degrees, which shows that our new solver works well for real data. We have summarized these results in Table 4 where we present the errors for the focal length, radial distortion, and the camera pose obtained using our proposed solver and for the sake of comparison we also list the errors, which were reported in [34], where the P4Pfr (40×50) solver was tested on the same dataset. Overall the errors are quite small, e.g. most of the focal lengths are within 0.1% of

Table 3 Stability comparison for solvers for the minimal problems from Table 2, generated by our new method, solvers generated using [32] (**col 4**), heuristic-based solvers [35] (**col 5**) and greedy parameter search [38] (**col 6**)

#	Problem	Our			Syzygy			Heuristic			Greedy		
		Mean	Med	Fail (%)	Mean	Med	Fail (%)	Mean	Med	Fail (%)	Mean	Med	Fail (%)
1	Rel. pose F+λ 8pt(‡)	−15.35	−15.38	**0**	−14.31	−14.63	**0**	−14.18	−14.48	**0**	−15.07	−14.46	**0**
2	Rel. pose E+f 6pt	−13.99	−14.26	**0**	−13.36	−13.64	**0**	−13.05	−13.34	**0**	−13.43	−13.79	**0**
3	Rel. pose f+E+f 6pt*	−14.38	−14.60	**0**	−13.60	−14.00	0.04	−13.87	−14.07	**0**	−14.15	−14.41	**0**
4	Rel. pose E+λ 6pt	−14.07	−14.27	**0**	−13.75	−13.94	**0**	−13.13	−13.34	0.04	−13.85	−14.08	**0**
5	Stitching fλ+R+fλ 3pt	−14.30	−14.46	**0**	−14.60	−14.37	**0**	−13.20	−13.46	**0**	−14.30	−14.51	**0**
6	Abs. Pose P4Pfr (elim. f)	−13.62	−13.82	**0**	−13.53	−13.77	**0**	−12.73	−13.00	**0**	−13.87	−14.13	**0**
7	Rel. pose λ+E+λ 6pt	−11.86	−12.33	0.06	−7.50	−7.78	5.16	−10.53	−10.89	0.40	−13.50	−13.95	**0**
8	Rel. pose λ_1+F+λ_2 9pt	−11.91	−12.14	**0.02**	−10.63	−11.22	1.94	−12.67	−13.08	0.10	−12.56	−13.16	0.40
9	Rel. pose E+fλ 7pt	−14.05	−14.23	**0**	−12.90	−13.11	**0**	–	–	–	−14.45	−14.75	**0**
10	Rel. pose E+fλ 7pt (elim. λ)	−12.53	−12.95	**0.02**	−12.59	−12.92	0.04	−12.53	−12.76	**0.02**	−12.52	−12.80	**0.02**
11	Rel. pose E+fλ 7pt (elim. fλ)	−13.37	−13.66	0.04	−12.60	−12.98	0.06	−13.59	−13.78	**0**	−13.10	−13.66	0.1
12	Rolling shutter pose	−13.67	−13.83	**0**	−14.52	−14.72	**0**	−12.43	−12.65	**0**	–	–	–
13	Triangulation from satellite im	−13.12	−13.03	**0**	−12.67	−13.06	**0**	−11.61	−11.93	0.50	−13.25	−13.50	**0**
14	Abs. pose refractive P5P*	−13.54	−13.77	**0**	−13.42	−14.05	0.48	−12.82	−13.26	0.20	−14.35	−14.55	**0**
15	Abs. pose quivers	−13.35	−13.57	**0**	−13.29	−13.62	0.32	−13.74	−13.95	**0**	−13.20	−13.83	0.24
16	Unsynch. Rel. pose(‡)	−13.59	−13.78	**0**	−10.53	−11.29	3.43	−12.78	−13.36	0.36	−12.28	−12.84	0.44
17	Optimal PnP (Hesch)	−13.14	−13.32	**0**	−12.89	−13.23	**0**	−13.20	−13.48	**0**	−14.60	−14.70	**0**
18	Optimal PnP (Cayley)	−12.36	−12.56	**0**	−11.33	−11.69	**0**	−12.88	−13.08	**0**	−14.30	−14.52	**0**
19	Optimal pose 2pt v2	−10.68	−11.06	0.52	−12.01	−12.32	**0**	−12.44	−12.54	0.05	−12.01	−12.21	**0**

For each minimal problem, the solvers that led to minimum number of failures are written in bold. Mean and median are computed from Log_{10} of normalized equation residuals. (†): In the offline stage in [38], the elimination template was reduced using a Schur complement. (−): Failed to extract solutions to all variables. (∗): Using a simplified input system of polynomials, as in [38]

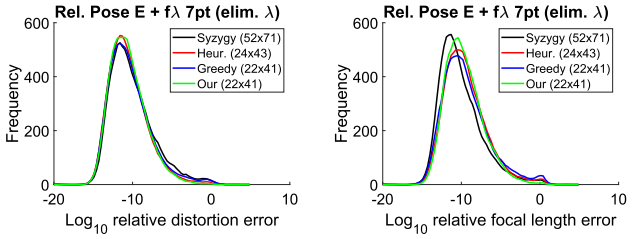


Fig. 3 Histograms of Log_{10} relative error in radial distortion (**left**) and focal length (**right**) for Rel. pose E+$f\lambda$ 7pt (elim λ) problem for 10K randomly generated synthetic scenes. These scenes represent cameras with different radial distortions, poses and focal lengths. For comparison, we generated solvers using our sparse resultant method (green), Gröbner basis method [32] (black), heuristic method [35] (red) and greedy parameter search method [38] (blue) (Color figure online)

the ground truth and almost all rotation errors are less than 0.1 degrees, which shows that our new solver as well as the original solver work well for real data. The results of both solvers are very similar. However, note that the slightly different results reported in [34] are due to RANSAC's random nature and a slightly different P4Pfr formulation (40×50) used in [34].

7 Conclusion

In this paper, we have proposed a sparse resultant method for generating efficient solvers for minimal problems in computer vision. It uses a polynomial with a special form to augment the initial polynomial system $\mathcal{F}$ in (1) and construct a sparse resultant matrix $\mathtt{M}(u_0)$ (7), using the theory of polytopes [15]. The special form enables us to decompose the resultant matrix constraint (8), into a regular eigenvalue problem (50) (or (52)) using the Schur complement. As demonstrated in Sect. 6, our sparse resultant method leads to minimal solvers with comparable speed and stability w.r.t. to the SOTA solvers based on the action matrix method [26, 32, 35, 38]. Note that the way our sparse resultant method is designed, for some minimal problems, it leads to solvers with larger eigenvalue problems but performing a smaller matrix inverse and with comparable or better stability compared to that of the Gröbner basis-based solvers.

While the action matrix method and the sparse resultant method are based on different mathematical theories, we have observed that the resulting solvers involve similar numerical operations, such as eigenvalue computation. This raises the question, "Under what conditions for a given minimal problem (a given system of polynomial equations), are the two solvers the same?" In Sect. 4, we have attempted to answer this question. Specifically, if we begin with an action matrix-based solver for a given minimal problem, then we propose a list of changes to be made to the steps in our sparse resultant method so that it leads to an equivalent sparse resultant-based solver. In the opposite direction, we also study the case when we begin with a sparse resultant-based solver and establish a list of changes to the steps performed by an action matrix method such that it leads to an equivalent solver. In other words, if we begin with an action matrix-based solver, it determines the extra polynomial f_{m+1} and the favourable monomial set, to be used in the sparse resultant method. Or, if we begin with a sparse resultant-based solver, it determines the monomial ordering, the extended polynomial set $\mathcal{F}'$, and the basis of the quotient ring to be used for computing an action matrix-based solver.

We hope that this discussion paves a path towards unifying the two approaches for generating minimal solvers, i.e., the action matrix methods and the sparse resultant method based on an extra polynomial. It is known that both these approaches are examples of the normal form methods for a given polynomial system $\mathcal{F}$, recently studied in [40, 54, 55]. To the best of our knowledge, normal forms have yet not been incorporated in the automatic generators for minimal solvers in computer vision. This may be an interesting future topic to investigate.

Acknowledgements Zuzana Kukelova was supported by the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 'Research Center for Informatics' and the Czech Science Foundation (GAČR) JUNIOR STAR Grant No. 22-23183M.

Funding Open Access funding provided by University of Oulu (including Oulu University Hospital).

Appendix A Action Matrix

Here, we describe the important steps typically performed by an action matrix method based on the theory of the Gröbner basis. Note that for a given minimal problem, the **offline** steps are performed only once, while the **online** steps are repeated for each instance of the input data/measurements for the given minimal problem.

1. **[Offline]** We begin with a system of m polynomial equations $\mathcal{F} = 0$ (1) in n variables. Let us denote the ideal generated by $\mathcal{F}$, as I. Let us also assume that $\mathcal{F} = 0$

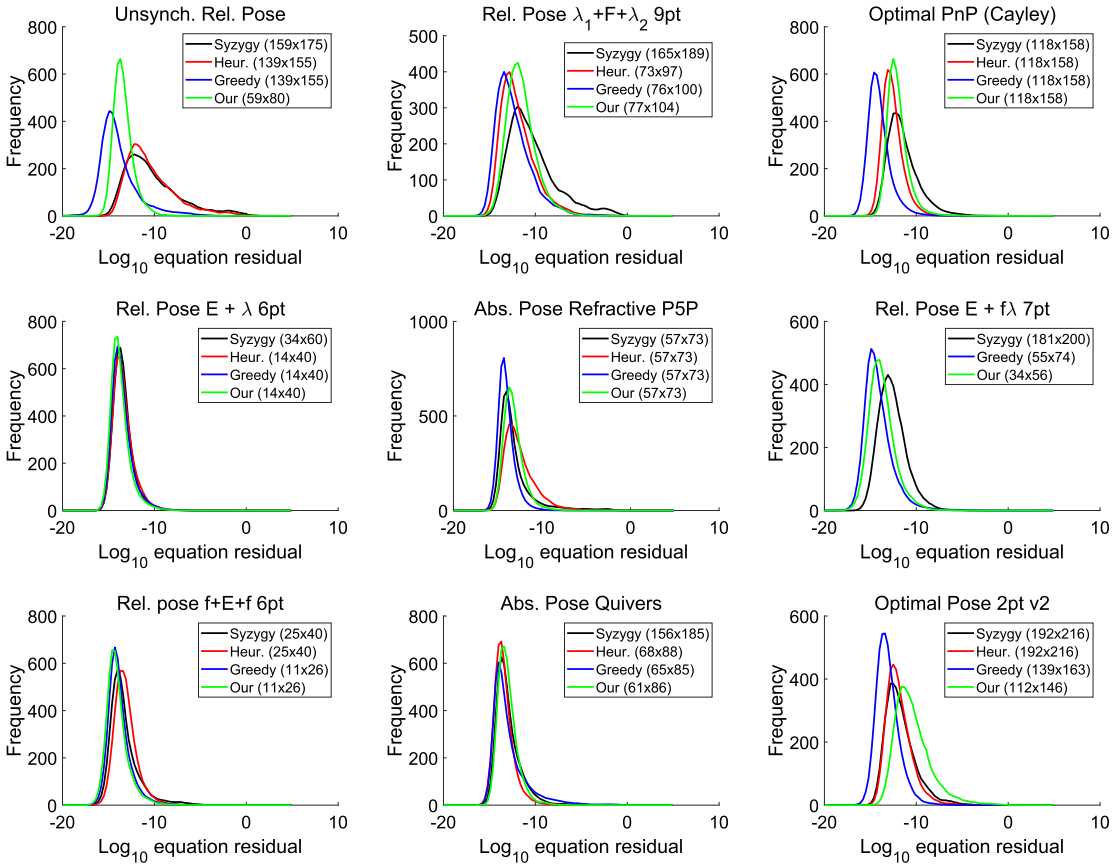


Fig. 4 Histograms of Log_{10} of normalized equation residual error for nine selected minimal problems. The methods tested for comparison are based on Gröbner basis [32], heuristics [32], greedy parameter search [38] and our sparse resultant method

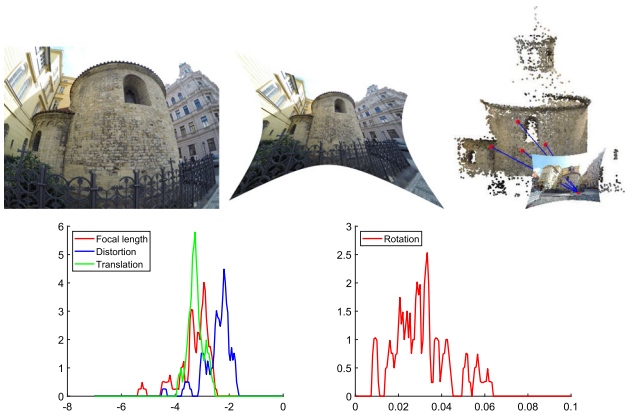


Fig. 5 Top row: Example of an input image (left). Undistorted image using the proposed resultant-based P4Pfr solver (middle). Input 3D point cloud and an example of registered camera (right). Bottom row: Histograms of errors for 62 images. The measured errors are (left) the $Log10$ relative focal length $|f - f_{GT}|/f_{GT}$, radial distortion $|k - k_{GT}|/|k_{GT}|$, and the relative translation error $\|\vec{t} - \vec{t}_{GT}\|/\|\vec{t}_{GT}\|$, and (right) the rotation error in degrees

has r solutions. It is well known that the quotient ring $A = \mathbb{C}[X]/I$ has the structure of a finite-dimensional vector space over the field $\mathbb{C}$ (see [15] for details). Let A be a l-dimensional vector space. Then, if I is a radical ideal, i.e., $I = \sqrt{I}$, we have $l = r$.

2. **[Offline]** This method computes a linear basis of A, denoted as $\mathcal{B}_A = \{[\mathbf{x}^{\alpha_1}], \ldots, [\mathbf{x}^{\alpha_r}]\}$. Here, *some* monomial ordering is assumed, to define the division of a polynomial $f \in \mathbb{C}[X]$ w.r.t. the ideal I. This polynomial division is denoted with the operator $[f]$. The set of monomials, corresponding to the elements in $\mathcal{B}_A$, are $B_a = \{\mathbf{x}^{\alpha_1}, \ldots, \mathbf{x}^{\alpha_r}\}$.
3. **[Offline]** Some $f \in \mathbb{C}[X]$ is assumed as an *action polynomial*, which sends each $[\mathbf{x}^{\alpha}] \in A$ to $[f\mathbf{x}^{\alpha}] \in A$. We thus have the following linear map

$$T_f : A \to A,\ T_f([\mathbf{x}^{\alpha}]) = \left[f\mathbf{x}^{\alpha}\right]. \tag{A1}$$

Fixing a linear basis $\mathcal{B}_A$ of A, allows us to represent the linear map T_f, with an $r \times r$ matrix, $\mathtt{M}_f = (m_{ij})$. Thus, for each $\mathbf{x}^{\alpha_j} \in B_a$, we have

$$T_f([\mathbf{x}^{\alpha_j}]) = [f\mathbf{x}^{\alpha_j}] = \sum_{i=1}^{r} m_{ij}[\mathbf{x}^{\alpha_i}]. \tag{A2}$$

In other words, the matrix $\mathtt{M}_f$ represents, what we call a multiplication or an action matrix. Assuming $B_{fa} = \{f\mathbf{x}^{\alpha_\mathbf{j}} \mid \mathbf{x}^{\alpha_\mathbf{j}} \in B_a\}$, we have

$$\mathtt{M}_f \mathbf{b}_a = \mathbf{b}_{fa}, \tag{A3}$$

where $\mathbf{b}_a = \text{vec}(B_a)$ and $\mathbf{b}_{fa} = \text{vec}(B_{fa})$. We can find polynomials $q_j \in I$, such that (A2) becomes

$$f\mathbf{x}^{\alpha_\mathbf{j}} = \sum_{i=1}^{r} m_{ij}\mathbf{x}^{\alpha_\mathbf{i}} + q_j. \tag{A4}$$

Note that $q_j \in I \implies q_j = \sum_{i=1}^{m} h_{ij} f_i$. Various action matrix methods in the literature adopt different approaches for computing these polynomials $q_j \in \mathbb{C}[X]$. Here, we assume the action polynomial $f = x_k$, for some variable $x_k \in X$.

4. **[Offline]** In all action matrix methods, we basically need to compute a set T_j of monomial multiples for each of the input polynomials $f_j \in \mathcal{F}$. This gives us an expanded set of polynomials, which we denote as $\mathcal{F}' = \{\mathbf{x}^\alpha f_i \mid f_i \in \mathcal{F}, \mathbf{x}^\alpha \in T_i\}$. This extended set of polynomials $\mathcal{F}'$, is constructed such that each q_j can be computed as a linear combination of the polynomials $\mathcal{F}'$.[8]
5. **[Offline]** Let $B = \text{mon}(\mathcal{F}')$, which is partitioned [8] as

$$B = B_e \cup B_r \cup B_a, \tag{A5}$$
$$B_r = \{x_k\mathbf{x}^\alpha \mid \mathbf{x}^\alpha \in B_a\} \setminus B_a, \tag{A6}$$
$$B_e = B \setminus (B_r \cup B_a), \tag{A7}$$

where B_r and B_e are, respectively, what we call the *reducible* monomials and the *excess* monomials.

6. **[Offline]** The set of equations $\mathcal{F}' = 0$ is expressed in a matrix form, as

$$\mathtt{C}\,\mathbf{b} = \begin{bmatrix} \mathtt{C}_{11} & \mathtt{C}_{12} & \mathtt{C}_{13} \\ \mathtt{C}_{21} & \mathtt{C}_{22} & \mathtt{C}_{23} \end{bmatrix} \begin{bmatrix} \mathbf{b}_e \\ \mathbf{b}_r \\ \mathbf{b}_a \end{bmatrix} = \mathbf{0}, \tag{A8}$$

where $\mathbf{b}_e = \text{vec}(B_e)$ and $\mathbf{b}_r = \text{vec}(B_r)$. The rows of $\mathtt{C}$ are assumed to be partitioned such that $\mathtt{C}_{22}$ is a square invertible matrix. The matrix $\mathtt{C}$ is known as an *elimination template*. As $\mathtt{C}$ represents a coefficient matrix of $\mathcal{F}'$ constructed as described in step 4, if we perform a G–J elimination of $\mathtt{C}$, we obtain the following

$$\begin{bmatrix} \mathtt{C}'_{11} & \mathtt{0} & \mathtt{C}'_{13} \\ \mathtt{0} & \mathtt{I} & \mathtt{C}'_{23} \end{bmatrix}. \tag{A9}$$

[8] The coefficients of each polynomial q_j are found through a G–J elimination of the matrix $\mathtt{C}$ representing the extended set of polynomials $\mathcal{F}'$.

Table 4 Errors for the real Rotunda dataset

Solver	Our P4Pfr 28 × 40			P4Pfr 40 × 50 SOTA		
	Avg	Med	Max	Avg	Med	Max
Focal (%)	0.080	0.063	0.266	0.08	0.07	0.29
Distortion (%)	0.522	0.453	1.651	0.51	0.45	1.85
Rotation (°)	0.031	0.029	0.062	0.03	0.03	0.10
Translation (%)	0.066	0.051	0.210	0.07	0.07	0.26

The errors are relative to the ground truth for all except rotation which is shown in degrees. The results for the SOTA P4Pfr solver of size 40 × 50, are taken from [34]

Note that, $\mathtt{C}_{22}$ is a square invertible matrix. The submatrix $\mathtt{C}'_{11}$ may not be square. This implies that some columns in $\begin{bmatrix} \mathtt{C}_{11} \\ \mathtt{C}_{21} \end{bmatrix}$ are linearly dependent, which can be removed [26], along with the corresponding monomials from the set of excess monomials B_e. Let the resulting monomial set be denoted as $\hat{B}_e$ and the reduced column block as $\begin{bmatrix} \hat{\mathtt{C}}_{11} \\ \hat{\mathtt{C}}_{21} \end{bmatrix}$. Then, (A8) becomes

$$\hat{\mathtt{C}}\,\hat{\mathbf{b}} = \begin{bmatrix} \hat{\mathtt{C}}_{11} & \mathtt{C}_{12} & \mathtt{C}_{13} \\ \hat{\mathtt{C}}_{21} & \mathtt{C}_{22} & \mathtt{C}_{23} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{b}}_e \\ \mathbf{b}_r \\ \mathbf{b}_a \end{bmatrix} = \mathbf{0}, \tag{A10}$$

where $\hat{\mathtt{C}}$ and $\hat{\mathbf{b}}$, respectively, denote the reduced elimination template and the monomial vector.

7. **[Online]** A Gauss–Jordan (G–J) elimination[9] of $\hat{\mathtt{C}}$ leads to

$$\begin{bmatrix} \mathtt{I} & \mathtt{0} & \hat{\mathtt{C}}'_{13} \\ \mathtt{0} & \mathtt{I} & \mathtt{C}'_{23} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{b}}_e \\ \mathbf{b}_r \\ \mathbf{b}_a \end{bmatrix} = \mathbf{0}. \tag{A11}$$

The lower block row is rewritten as

$$\begin{bmatrix} \mathtt{I} & \mathtt{C}'_{23} \end{bmatrix} \begin{bmatrix} \mathbf{b}_r \\ \mathbf{b}_a \end{bmatrix} = \mathbf{0}. \tag{A12}$$

The entries of the action matrix $\mathtt{M}_f$ can be then read off from the entries of the matrix $\mathtt{C}'_{23}$ [8, 26]. For $\mathbf{x}^{\alpha_\mathbf{j}} \in B_a$, if $x_k\mathbf{x}^{\alpha_\mathbf{j}} \in B_a$ for some $\mathbf{x}^{\alpha_{\mathbf{i}_1}} \in B_a$, then j-th row of $\mathtt{M}_f$ contains 1 in i_1-th column and 0 in the remaining columns. But if $x_k\mathbf{x}^{\alpha_\mathbf{j}} \notin B_a$, then $x_k\mathbf{x}^{\alpha_\mathbf{j}} \in B_r$ for some $\mathbf{x}^{\alpha_{\mathbf{i}_2}} \in B_r$. In this case, j-th row of $\mathtt{M}_f$ is the i_2-th row of $-\mathtt{C}'_{23}$.
We recover solutions to $\mathcal{F} = 0$, from the eigenvalues (and eigenvectors) of the action matrix $\mathtt{M}_f$. Specifically, $u_0 \in \mathbb{C}$ is an eigenvalue of the matrix $\mathtt{M}_f$, iff u_0 is a value of the function f on the variety V. We refer to the book

[9] Usually, G–J elimination is performed through a step of LU or QR factorization [8].

[15] for further details. In other words, if $f = x_k$, then the eigenvalues of $\mathtt{M}_f$ are the x_k-coordinates of the solutions of (1). The solutions to the remaining variables can be obtained from the eigenvectors of $\mathtt{M}_f$. This means that after finding the multiplication matrix $\mathtt{M}_f$, we can recover the solutions by its eigendecompostion, for which efficient algorithms exist.

The output of the offline stage is the elimination template $\hat{\mathtt{C}}$ in step 6. In the online stage, step 7, for a given instance of the minimal problem, we fill in the coefficients of the elimination template $\hat{\mathtt{C}}$ using the input data, and perform its G–J elimination to compute the action matrix $\mathtt{M}_f$. Eigendecomposition of $\mathtt{M}_f$, then gives us the solutions to the given instance of the minimal problem.

The first automatic approach for generating elimination templates and Gröbner basis solvers was presented in [27]. Recently, an improvement to the automatic generator [27] was proposed in [32]. It exploits the inherent relations between the input polynomial equations and it results in more efficient solvers than [27]. The automatic method from [32] was later extended by a method for dealing with saturated ideals [33] and a method for detecting symmetries in polynomial systems [31].

Appendix B Algorithms

Here, we provide the algorithms for our sparse resultant method in Sect. 3.

B.1 Extracting a Favourable Set of Monomials

Algorithm 1 computes a favourable set of monomials B and a block partition of the corresponding coefficient matrix $\mathtt{C}(u_0)$, for an instance of a minimal problem, i.e., a system of m polynomial equations $\mathcal{F} = 0$, in n unknown variables X (1). The output of the algorithm also contains a set of monomial multiples T. Our approach for computing a favourable monomial set is described in Sect. 3.2, and our approach for partitioning the coefficient matrix is described in Sect. 3.3. In Sect. 3.3, we have considered two ways of partitioning the favourable monomial set B, i.e., as in (42) and in (43), in order to block partition $\mathtt{C}(u_0)$. However, in Algorithm 1, we consider the first one. For the alternate partition, all steps remain the same, except the step 15, where we use the alternative partition in (43).

$$B'_1 \leftarrow \{\mathbf{x}^\alpha \in B' \mid \frac{\mathbf{x}^\alpha}{x_i} \in T_{m+1}\},\ B'_2 \leftarrow B' \setminus B'_1.$$

Algorithm 1 Computing a favourable set of monomials B and block partitioning the coefficient matrix $\mathtt{C}(u_0)$

Input : $\mathcal{F} = \{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\}$, $\mathbf{x} = [x_1, \ldots, x_n]$
Output : $B, T, \mathtt{C}(u_0)$
1: $B \leftarrow \phi, T \leftarrow \phi$
2: **for** $k \in \{1, \ldots, n\}$ **do**
3: $\mathcal{F}_a \leftarrow \{f_1, \ldots, f_{m+1}\}$, $f_{m+1} = x_k - u_0$
4: Calculate the support of the input polynomials: $A_j \leftarrow \text{supp}(f_j)$, $j = 1, \ldots, m+1$
5: Construct Newton polytopes: $NP_j \leftarrow \text{conv}(A_j)$, $j = 1, \ldots, m+1$ as well as a unit simplex $NP_0 \subset \mathbb{Z}^n$.
6: Enumerate combinations of indices of all possible sizes: $K \leftarrow \{\{k_0, \ldots, k_i\} \mid \forall 0 \leq i \leq (m+1); k_0, \ldots, k_i \in \{0, \ldots, m+1\}; k_j < k_{j+1}\}$
7: Let $\Delta \leftarrow \{\{\delta_1, \ldots, \delta_{n+1}\} \mid \delta_i \in \{-0.1, 0, 0.1\}; i = 1, \ldots, (n+1)\}$
8: **for** $I \in K$ **do**
9: Compute the Minkowski sum: $Q \leftarrow \sum_{j \in I}(NP_j)$
10: **for** $\delta \in \Delta$ **do**
11: $B' \leftarrow \{\mathbf{x}^\alpha \mid \alpha \in \mathbb{Z}^n \cap (Q + \delta)\}$
12: $\mathcal{F}_a \xrightarrow{B'} (\mathcal{F}'_a, T')$
13: T' contains $\{T'_1 \ldots T'_{m+1}\}$
14: Compute $\mathtt{C}(u_0)'$ from B' and T'
15: $B'_1 \leftarrow B' \cap T'_{m+1}$, $B'_2 \leftarrow B' \setminus B'_1$
16: **if** $\Sigma_{j=1}^{m+1}|T'_j| \geq |B'|$ and $\min_j |T'_j| > 0$ and $rank(\mathtt{C}(u_0)') = |B'|$ **then**
17: $\mathtt{A}_{12} \leftarrow$ submatrix of $\mathtt{C}(u_0)'$ column-indexed by B'_2 and row-indexed by $T'_1 \cup \cdots \cup T'_m$
18: **if** $\text{rank}(\mathtt{A}_{12}) = |B'_2|$ and $|B| \geq |B'|$ **then**
19: $B \leftarrow B', T \leftarrow T'$
20: **end if**
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: Compute $\mathtt{C}(u_0)$ from B and T

Algorithm 2 Row-column removal

Input : B, T
Output : $B_{\text{red}}, T_{\text{red}}, \mathtt{C}_{\text{red}}(u_0)$

1: $B' \leftarrow B, T' \leftarrow T$
2: **repeat**
3: stopflag $\leftarrow$ TRUE
4: Compute $\mathtt{C}(u_0)'$ from B' and T'
5: **for** column c in $\mathtt{C}(u_0)'$ **do**
6: Copy $\mathtt{C}(u_0)'$ to $\mathtt{C}(u_0)''$
7: Remove rows $r_1, \ldots, r_s$ containing c from $\mathtt{C}(u_0)''$
8: Remove columns $c_1, \ldots, c_l$ of $\mathtt{C}(u_0)''$ present in $r_1, \ldots, r_s$
9: **if** $\mathtt{C}(u_0)''$ satisfies Prop. 1 **then**
10: Remove monomials from B' indexing columns $c_1, \ldots, c_l$
11: Remove monomials from T' indexing rows $r_1, \ldots, r_s$
12: stopflag $\leftarrow$ FALSE
13: **break**
14: **end if**
15: **end for**
16: **until** stopflag is TRUE
17: $B_{\text{red}} \leftarrow B', T_{\text{red}} \leftarrow T'$
18: Compute $\mathtt{C}_{\text{red}}(u_0)$ from B_{red} and T_{red}

B.2 Row-Column Removal

The next step in the proposed method is to reduce the favourable monomial set B, by removing columns from $\mathtt{C}(u_0)$ along with a corresponding set of rows, described in Sect. 3.4. Algorithm 2 achieves this. Its input is the favourable monomial set B, the corresponding set of monomial multiples T, computed by Algorithm 1 and the output is a reduced monomial set B_{red} and a reduced set of monomial multiples, T_{red} that index the columns and rows of the reduced matrix $\mathtt{C}_{\text{red}}(u_0)$, respectively. We note that this algorithm is the same irrespective of the version of partition of the monomial set B, i.e. (42) or (43).

B.3 Row Removal

It may happen that the reduced matrix $\mathtt{C}_{\text{red}}(u_0)$ still has more rows than columns. Therefore, we propose an approach to remove the excess rows from $\mathtt{C}_{\text{red}}(u_0)$, to transform it into a square matrix, which will be our sparse resultant matrix $\mathtt{M}(u_0)$. Towards this, we provide Algorithm 3 to remove the extra rows from $\mathtt{C}_{\text{red}}(u_0)$ by removing some monomial multiples from T_{red}. It accepts the favourable monomial set B_{red} and its corresponding monomial multiples T_{red}, as input and returns a reduced set of monomial multiples, T_{red} such that, along with the basis B_{red}, leads to a square matrix. If we partitioned B_{red} using the alternative approach (43), we just need to change step 17 in Algorithm 3 to

$$B'_1 \leftarrow \{\mathbf{x}^\alpha \in B' \mid \frac{\mathbf{x}^\alpha}{x_i} \in T_{m+1}\},\ B'_2 \leftarrow B' \setminus B'_1.$$

Algorithm 3 Removal of the extra rows

Input B_{red}, T_{red}
Output T_{red}, $\mathtt{M}(u_0)$
1: T_{red} contains $\{T'_1, \ldots, T'_{m+1}\}$
2: $B_N \leftarrow |B_{\text{red}}|$, $T_N \leftarrow \Sigma_{j=1}^{m+1}|T'_j|$, $t_{\text{chk}} \leftarrow \phi$
3: **while** $T_N > B_N$ **do**
4: $B' \leftarrow B_{\text{red}}$, $T' \leftarrow T_{\text{red}}$
5: T' contains $\{T'_1, \ldots, T'_{m+1}\}$
6: Randomly select $t \in \{t_m \in T'_{m+1} \mid (t_m, m+1) \notin t_{\text{chk}}\}$
7: **if** t **then**
8: $T'_{m+1} \leftarrow T'_{m+1} \setminus \{t\}$
9: $T' \leftarrow \{T'_1, \ldots, T'_{m+1}\}$
10: $t_{\text{chk}} \leftarrow t_{\text{chk}} \cup \{(t, m+1)\}$
11: **else**
12: Randomly select $i \in \{1, \ldots, m\}$
13: Randomly select $t \in \{t_i \in T'_i \mid (t_i, i) \notin t_{\text{chk}}\}$
14: $T'_i \leftarrow T'_i \setminus \{t\}$, $T' \leftarrow \{T'_1, \ldots, T'_{m+1}\}$
15: $t_{\text{chk}} \leftarrow t_{\text{chk}} \cup \{(t, i)\}$
16: **end if**
17: $B'_1 \leftarrow B' \cap T'_{m+1}$, $B'_2 \leftarrow B' \setminus B'_1$
18: Compute $\mathtt{C}(u_0)'$ from B' and T'
19: **if** $\mathtt{C}(u_0)'$ satisfies Prop. 1 **then**
20: $T_{\text{red}} \leftarrow T'$, $T_N \leftarrow \Sigma_{j=1}^{m+1}|T'_j|$
21: **end if**
22: **end while**
23: Compute $\mathtt{M}(u_0)$ from B_{red} and T_{red}

Appendix C Proofs

In this appendix, we provide proofs for the propositions discussed in Sect. 4, regarding the equivalence of an action matrix-based solver with a sparse resultant-based solver, in three situations.

Proposition C.1 *For a given system of polynomial equations $\mathcal{F} = 0$ (1), let us assume to have generated an action matrix-based solver using the method in Sect. 2.1. The action matrix-based solver is assumed to be generated for the action variable, $f = x_1$. Also, let us consider a sparse resultant-based solver, generated after applying the changes C1R-C6R in Sect. 4.1, to the steps 1–5 in Sect. 3.1, i.e. the offline stage. Then, the two solvers are equivalent, as defined in Definition 3.*

Proof Note from (53) that the elimination template can be written as

$${}^a\hat{\mathtt{C}} = \begin{bmatrix} \hat{\mathtt{C}}_{11} & \mathtt{C}_{12} & \mathtt{C}_{13} \\ \hat{\mathtt{C}}_{21} & \mathtt{C}_{22} & \mathtt{C}_{23} \end{bmatrix} = \begin{bmatrix} \hat{\mathtt{A}}_{12} & \hat{\mathtt{A}}_{11} \end{bmatrix}. \tag{C13}$$

Therefore, G–J elimination of ${}^a\hat{\mathtt{C}}$ can be considered as computing the matrix product, $\hat{\mathtt{A}}_{12}^{-1}\,{}^a\hat{\mathtt{C}} = \begin{bmatrix} \mathtt{I} & \hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11} \end{bmatrix}$. Comparing it with the G–J eliminated form of the matrix ${}^a\hat{\mathtt{C}}$ in (A11), we have

$$\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11} = \begin{bmatrix} \hat{\mathtt{C}}'_{13} \\ \mathtt{C}'_{23} \end{bmatrix}. \tag{C14}$$

Note that for each $\mathbf{x}^{\alpha_\mathbf{j}}$ in ${}^rT_{m+1} (= B_1)$, the j-th row of $\begin{bmatrix} \mathtt{A}_{21} - u_0\mathtt{I} & \mathtt{A}_{22} \end{bmatrix}{}^r\mathbf{b}$ (55) represents the multiple $\mathbf{x}^{\alpha_\mathbf{j}} f_{m+1} = \mathbf{x}^{\alpha_\mathbf{j}} x_1 - \mathbf{x}^{\alpha_\mathbf{j}} u_0$. Then, we have the following two cases, for each $\mathbf{x}^{\alpha_\mathbf{j}} \in B_1$:

Case 1: If $x_1\mathbf{x}^{\alpha_\mathbf{j}} = \mathbf{x}^{\alpha_{\mathbf{i_1}}} \in B_1$, the j-th row of $\mathtt{A}_{22}$ is $\mathbf{0}$. The j-th row of $\mathtt{A}_{21}$, and hence that of $\mathtt{X}$ has 1 in i_1-th column and 0 in the remaining columns. Moreover, in this case, the j-th row of the action matrix $\mathtt{M}_f$ should also have 1 in i_1-th column and 0 in the remaining columns (see step 7 of the action matrix-based method in Sect. 2.1).

Case 2: If $x_1\mathbf{x}^{\alpha_\mathbf{j}} \notin B_1$, then $x_1\mathbf{x}^{\alpha_\mathbf{j}} = \mathbf{x}^{\alpha_{\mathbf{i_2}}}$, for some $\mathbf{x}^{\alpha_{\mathbf{i_2}}} \in B_2$. Then the j-th row of $\mathtt{A}_{21}$ is $\mathbf{0}$, and j-th row of $\mathtt{A}_{22}$ has 1 in i_2-th column and 0 in the remaining columns. Therefore, the j-th row of the

matrix product $-\mathtt{A}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$, and hence that of the matrix $\mathtt{X}$, is actually the i_2-th row of $-\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$. From (C14), this is actually a row from the lower block,[10] $-\mathtt{C}'_{23}$. However, from the discussion in the step 7 in Sect. 2.1, this is the same as the j-th row of the action matrix $\mathtt{M}_f$.

Thus in both cases, the rows of the matrices $\mathtt{M}_f$ and $\mathtt{X}$ are the same, and therefore, $\mathtt{X} = \mathtt{M}_f$. Moreover, (C14) implies that computing the matrix product $\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$ can be replaced by the step of G–J elimination of the matrix ${}^a\hat{\mathtt{C}}$. Therefore, as both the conditions in Definition 3 are satisfied, the action matrix-based solver is equivalent to the sparse resultant-based solver. □

Proposition C.2 *For a given system of polynomial equations $\mathcal{F} = 0$ (1), let us assume to have generated a sparse resultant-based solver. Let us also consider an action matrix-based solver, generated after applying the changes C1A-C6A in Sect. 4.2 to the offline stage, i.e. steps 1–6 in Sect. 2.1. Then the two solvers are equivalent as defined in Definition 3.*

Proof From (57), note that the elimination template ${}^a\hat{\mathtt{C}}$ can be expressed as ${}^a\hat{\mathtt{C}} = \begin{bmatrix}\hat{\mathtt{C}}_{11} & \mathtt{C}_{12} & \mathtt{C}_{13}\\ \hat{\mathtt{C}}_{21} & \mathtt{C}_{22} & \mathtt{C}_{23}\end{bmatrix} = \begin{bmatrix}\hat{\mathtt{A}}_{12} & \hat{\mathtt{A}}_{11}\end{bmatrix}$. Therefore a G–J elimination of ${}^a\hat{\mathtt{C}}$ can be achieved by computing the matrix product $\hat{\mathtt{A}}_{12}^{-1}\,{}^a\hat{\mathtt{C}} = \begin{bmatrix}\mathtt{I} & \hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}\end{bmatrix}$. Comparing it with the G–J eliminated form of the matrix ${}^a\hat{\mathtt{C}}$ in (A11), we can write

$$\begin{bmatrix}\hat{\mathtt{C}}'_{13}\\ \mathtt{C}'_{23}\end{bmatrix} = \hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}. \tag{C15}$$

This is exactly the same situation, as in Proposition (C.1). Specifically, the Schur complement $\mathtt{X}$, and the action matrix $\mathtt{M}_f$, are exactly the same matrices, with each row, either being a row from $-\mathtt{C}'_{23}$ or a row containing 1's or 0's. Moreover (C15) implies that G–J elimination of ${}^a\hat{\mathtt{C}}$ can be replaced by the matrix product, $\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$. Therefore, as both the conditions in Definition 3 are satisfied, the action matrix-based solver is equivalent to the sparse resultant-based solver. □

Proposition C.3 *For a given system of polynomial equations $\mathcal{F} = 0$ (1), let us assume to have generated a sparse resultant-based solver using the alternative partition of ${}^rB$ (32) and ${}^r\mathtt{C}(u_0)$ (34). Let us also consider an action matrix-based solver, generated after applying the changes C1A′-C6A′ in Sect. 4.3 to the steps 1–6 in Sect. 2.1, i.e. the offline stage. Then the two solvers are equivalent as defined in Definition 3.*

Proof In the step C4A′, the monomial multiples are set to be the same as those in the sparse resultant method, i.e. ${}^aT_i = {}^rT_i, i = 1, \ldots, m+1$. Therefore, the upper block of the elimination template ${}^a\mathtt{C}$ in (A8), is fixed by setting

$$\mathtt{C}_{11} = \hat{\mathtt{A}}_{12}, \mathtt{C}_{12} = 0, \mathtt{C}_{13} = \hat{\mathtt{A}}_{11}, \tag{C16}$$

where $\begin{bmatrix}\hat{\mathtt{A}}_{11} & \hat{\mathtt{A}}_{12}\end{bmatrix}$ (34) is the upper block of the sparse resultant matrix $\mathtt{M}(u_0)$.

Note that each monomial $\mathbf{x}^{\alpha_\mathtt{j}}$ in ${}^aT_{m+1}$ provides the j-th row in the lower block of ${}^a\mathtt{C}$, as a vector form of the polynomial $\mathbf{x}^{\alpha_\mathtt{j}}\,{}^af_{m+1} = \mathbf{x}^{\alpha_\mathtt{j}}(x_1\lambda - 1)$. Similarly, each monomial $\mathbf{x}^{\alpha_\mathtt{j}}$ in ${}^rT_{m+1}$ provides the j-th row in the lower block of ${}^r\mathtt{C}(u_0)$ in (34), as a vector form of the polynomial $\mathbf{x}^{\alpha_\mathtt{j}}\,{}^rf_{m+1} = \mathbf{x}^{\alpha_\mathtt{j}}(x_1 - u_0)$. The lower block of ${}^a\mathtt{C}$ in (A8) is of the form $\begin{bmatrix}\mathtt{C}_{21} & \mathtt{I} & \mathtt{C}_{23}\end{bmatrix}$, and the lower block of ${}^r\mathtt{C}(u_0)$ in (34) is of the form $\begin{bmatrix}\mathtt{I} + u_0\mathtt{B}_{21} & u_0\mathtt{B}_{22}\end{bmatrix}$. Note that the columns of $\mathtt{C}_{21}$ and $\mathtt{C}_{23}$ are, respectively, indexed by $\mathbf{b}_e$ and $\mathbf{b}_a$, whereas the columns of $\mathtt{B}_{21}$ and $\mathtt{B}_{22}$ are, respectively, indexed by $\mathbf{b}_1$ and $\mathbf{b}_2$. From the change C6A′, note that $\mathbf{b}_a = \mathbf{b}_1$, $\mathbf{b}_e = \mathbf{b}_2$ and ${}^aT_{m+1} = {}^rT_{m+1}$. Therefore, $\mathtt{C}_{21} = \mathtt{B}_{22}$ and $\mathtt{C}_{23} = \mathtt{B}_{21}$.

Substituting the upper and the lower blocks of ${}^a\mathtt{C}$, in (A10), we obtain

$${}^a\mathtt{C}\,{}^a\mathbf{b} = \begin{bmatrix}\hat{\mathtt{A}}_{12} & 0 & \hat{\mathtt{A}}_{11}\\ \mathtt{B}_{22} & \mathtt{I} & \mathtt{B}_{21}\end{bmatrix}\begin{bmatrix}\mathbf{b}_e\\ \mathbf{b}_r\\ \mathbf{b}_a\end{bmatrix} = \mathbf{0}. \tag{C17}$$

Note that, in the sparse resultant approach, the matrix $\hat{\mathtt{A}}_{12}$ has to be square invertible. Therefore, the matrix $\begin{bmatrix}\hat{\mathtt{A}}_{12} & 0\\ \mathtt{B}_{22} & \mathtt{I}\end{bmatrix}$ is also invertible and there are no extra columns to be removed in the step 6 in Sect. 2.1. This means that we have ${}^a\hat{\mathtt{C}} = {}^a\mathtt{C}$, ${}^a\hat{\mathbf{b}} = {}^a\mathbf{b}$, ${}^a\hat{\mathbf{b}}_e = {}^a\mathbf{b}_e$, $\hat{\mathtt{C}}_{11} = \hat{\mathtt{A}}_{12}$ and $\hat{\mathtt{C}}_{21} = \mathtt{B}_{22}$, in (A10). Note that

$$\begin{bmatrix}\hat{\mathtt{A}}_{12} & 0\\ \mathtt{B}_{22} & \mathtt{I}\end{bmatrix}^{-1} = \begin{bmatrix}\hat{\mathtt{A}}_{12}^{-1} & 0\\ -\mathtt{B}_{22}\hat{\mathtt{A}}_{12}^{-1} & \mathtt{I}\end{bmatrix}. \tag{C18}$$

Therefore, a G–J elimination of the matrix ${}^a\hat{\mathtt{C}}$ is of the form:

$$\begin{bmatrix}\mathtt{I} & 0 & \hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}\\ 0 & \mathtt{I} & \mathtt{B}_{21} - \mathtt{B}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}\end{bmatrix}. \tag{C19}$$

Comparing it with the G–J eliminated form of the matrix ${}^a\hat{\mathtt{C}}$ in (A11), the submatrix $\mathtt{C}'_{23}$ is

$$\mathtt{C}'_{23} = \mathtt{B}_{21} - \mathtt{B}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}. \tag{C20}$$

From the lower block (A12), we have $\mathbf{b}_r = -\mathtt{C}'_{23}\mathbf{b}_a$. In this case $B_r \cap B_a = \emptyset$, which means that $-\mathtt{C}'_{23}$ is the action matrix $\mathtt{M}_f$ (A3), for $f = \lambda$. However, from (52), $\mathtt{X} = \mathtt{B}_{21} - \mathtt{B}_{22}\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$. Thus, the action matrix $\mathtt{M}_f$ is equal

[10] i_2-th row of $\begin{bmatrix}\hat{\mathtt{C}}'_{13}\\ \mathtt{C}'_{23}\end{bmatrix}$ will always be a row from the lower block, indexed by the monomials in $\mathbf{b}_r$. This is because $x_1\mathbf{x}^{\alpha_\mathtt{j}} = \mathbf{x}^{\alpha_{\mathtt{i}_2}} \in B_2 \implies \mathbf{x}^{\alpha_{\mathtt{i}_2}} \in B_r$, and from the change C4R, $B_2 = \hat{B}_e \cup B_r$.

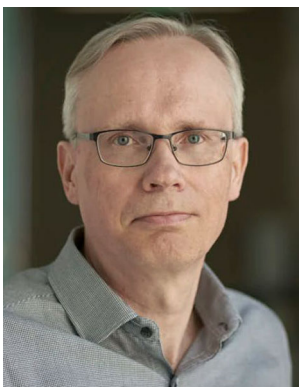
to the Schur complement $\mathtt{X}$. Moreover from (C19), it can be seen that a step of G–J elimination of the matrix ${}^{a}\hat{\mathtt{C}}$ can be replaced by the step of computing the matrix product $\hat{\mathtt{A}}_{12}^{-1}\hat{\mathtt{A}}_{11}$. Therefore, as both the conditions in Definition 3 are satisfied, the action matrix-based solver is equivalent to the sparse resultant-based solver. □

References

1. Albl, C., Kukelova, Z., Fitzgibbon, A.W., Heller, J., Smíd, M., Pajdla, T.: On the two-view geometry of unsynchronized cameras. CoRR arXiv:1704.06843 (2017)
2. Bhayani, S., Kukelova, Z., Heikkilä, J.: Automatic generator for sparse resultant solvers. https://github.com/snehalbhayani/aut_gen_sparse_res_solver (2020)
3. Bhayani, S., Kukelova, Z., Heikkilä, J.: A sparse resultant based method for efficient minimal solvers. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1767–1776 (2020)
4. Bhayani, S., Kukelova, Z., Heikkilä, J.: Computing stable resultant-based minimal solvers by hiding a variable. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 6104–6111 (2021)
5. Bujnak, M., Kukelova, Z., Pajdla, T.: Making minimal solvers fast. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012), pp. 1506–1513 (2012)
6. Bujnak, M., Kukelova, Z., Pajdla, T.: 3D reconstruction from image collections with a single known focal length. In: International Conference on Computer Vision (ICCV), pp. 1803–1810. IEEE (2009)
7. Bujnak, M., Kukelova, Z., Pajdla, T.: New efficient solution to the absolute pose problem for camera with unknown focal length and radial distortion. In: Asian Conference on Computer Vision (ACCV), pp. 11–24. Springer (2010)
8. Byröd, M., Josephson, K., Åström, K.: Fast and stable polynomial equation solving and its application to computer vision. Int. J. Comput. Vis. **84**, 237–256 (2009)
9. Byröd, M., Josephson, K., Åström, K.: Improving numerical accuracy of Gröbner basis polynomial equation solvers. In: International Conference on Computer Vision (ICCV). IEEE (2007)
10. Byröd, M., Josephson, K., Åström, K.: A column-pivoting based strategy for monomial ordering in numerical Gröbner basis calculations. In: European Conference on Computer Vision (ECCV). Springer, Berlin (2008)
11. Canny, J.F., Emiris, I.Z.: A subdivision-based algorithm for the sparse resultant. J. ACM **47**(3), 417–451 (2000)
12. Checa, C., Emiris, I.Z.: Mixed subdivisions suitable for the Canny–Emiris formula. Math. Comput. Sci. **18**, 1–18 (2022)
13. Chum, O., Matas, J., Kittler, J.: Locally optimized RANSAC. In: Michaelis, B., Krell, G. (eds.) Pattern Recognition, pp. 236–243. Springer, Berlin (2003)
14. Cox, D., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. An Introduction to Computational Algebraic Geometry and Commutative Algebra. Springer, New York (2007)
15. Cox, D.A., Little, J.B., O'Shea, D.: Using Algebraic Geometry. Graduate Texts in Mathematics, vol. 185, 1st edn. Springer, Berlin (1998)
16. Emiris, I.Z.: A general solver based on sparse resultants. CoRR arXiv:1201.5810 (2012)
17. Emiris, I.Z., Canny, J.F.: A practical method for the sparse resultant. In: Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation, ISSAC '93, Kiev, Ukraine, July 6–8, 1993, pp. 183–192 (1993)
18. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
19. Haner, S., Åström, K.: Absolute pose for cameras under flat refractive interfaces. In: Computer Vision and Pattern Recognition (CVPR), pp. 1428–1436 (2015)
20. Hartley, R.I., Li, H.: An efficient hidden variable approach to minimal-case camera motion estimation. IEEE Trans. Pattern Anal. Mach. Intell. **34**, 2303–2314 (2012)
21. Heikkilä, J.: Using sparse elimination for solving minimal problems in computer vision. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017, pp. 76–84 (2017)
22. Hesch, J.A., Roumeliotis, S.I.: A direct least-squares (dls) method for pnp. In: 2011 International Conference on Computer Vision, pp. 383–390 (2011)
23. Kasten, Y., Galun, M., Basri, R.: Resultant based incremental recovery of camera pose from pairwise matches. In: IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7–11, 2019, pp. 1080–1088 (2019)
24. Kuang, Y., Åström, K.: Pose estimation with unknown focal length using points, directions and lines. In: International Conference on Computer Vision (ICCV), pp. 529–536 (2013)
25. Kuang, Y., Solem, J.E., Kahl, F., Åström, K.: Minimal solvers for relative pose with a single unknown radial distortion. In: Computer Vision and Pattern Recognition (CVPR), pp. 33–40. IEEE (2014)
26. Kukelova, Z.: Algebraic methods in computer vision. Ph.D. thesis, Czech Technical University in Prague (2013)
27. Kukelova, Z., Bujnak, M., Pajdla, T.: Automatic generator of minimal problem solvers. In: European Conference on Computer Vision (ECCV 2008), Proceedings, Part III, volume 5304 of Lecture Notes in Computer Science (2008)
28. Kukelova, Z., Bujnak, M., Pajdla, T.: Polynomial eigenvalue solutions to minimal problems in computer vision. IEEE Trans. Pattern Anal. Mach. Intell. **34**, 1381–1393 (2012)
29. Kukelova, Z., Heller, J., Bujnak, M., Fitzgibbon, A., Pajdla, T.: Efficient solution to the epipolar geometry for radially distorted cameras. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2309–2317 (2015)
30. Kukelova, Z., Kileel, J., Sturmfels, B., Pajdla, T.: A clever elimination strategy for efficient minimal solvers. In: Computer Vision and Pattern Recognition (CVPR). IEEE (2017)
31. Larsson, V., Åström, K.: Uncovering symmetries in polynomial systems. In: European Conference on Computer Vision (ECCV). Springer (2016)
32. Larsson, V., Åström, K., Oskarsson, M.: Efficient solvers for minimal problems by syzygy-based reduction. In: Computer Vision and Pattern Recognition (CVPR) (2017)
33. Larsson, V., Åström, K., Oskarsson, M.: Polynomial solvers for saturated ideals. In: International Conference on Computer Vision (ICCV) (2017)
34. Larsson, V., Kukelova, Z., Zheng, Y.: Making minimal solvers for absolute pose estimation compact and robust. In: International Conference on Computer Vision (ICCV) (2017)
35. Larsson, V., Oskarsson, M., Åström, K., Wallis, A., Kukelova, Z., Pajdla, T.: Beyond grobner bases: Basis selection for minimal solvers. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp. 3945–3954 (2018)
36. Li, B., Heng, L., Lee, G.H., Pollefeys, M.: A 4-point algorithm for relative pose estimation of a calibrated camera with a known relative rotation angle. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1595–1601. IEEE (2013)
37. Martyushev, E., Bhayani, S., Pajdla, T.: Automatic solver generator for systems of Laurent polynomial equations (2023)

38. Martyushev, E., Vrablikova, J., Pajdla, T.: Optimizing elimination templates by greedy parameter search (2022)
39. Mourrain, B., Telen, S., Van Barel, M.: Truncated normal forms for solving polynomial systems: generalized and efficient algorithms. J. Symb. Comput. **102**, 63–85 (2021)
40. Mourrain, B., Trébuchet, P.: Stable normal forms for polynomial system solving. Theor. Comput. Sci. **409**(2), 229–240 (2008)
41. Nakano, G.: Globally optimal DLS method for pnp problem with Cayley parameterization. In: Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7–10, 2015, pp. 78.1–78.11 (2015)
42. Naroditsky, O., Daniilidis, K.: Optimizing polynomial solvers for minimal geometry problems. In: International Conference on Computer Vision (ICCV). IEEE (2011)
43. Nistér, D.: An efficient solution to the five-point relative pose problem. IEEE Trans. Pattern Anal. Mach. Intell. **26**(6), 756–770 (2004)
44. RealityCapture. www.capturingreality.com (2016)
45. Raguram, R., Chum, O., Pollefeys, M., Matas, J., Frahm, J.-M.: USAC: a universal framework for random sample consensus. IEEE Trans. Pattern Recognit. Mach. Intell. **35**(8), 2022–2038 (2013)
46. Saurer, O., Pollefeys, M., Lee, G.H.: A minimal solution to the rolling shutter pose estimation problem. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 1328–1334. IEEE (2015)
47. Stetter, H.J.: Matrix eigenproblems are at the heart of polynomial system solving. SIGSAM Bull. **30**(4), 22–25 (1996)
48. Stewenius, H., Engels, C., Nistér, D.: Recent developments on direct relative orientation. ISPRS J. Photogramm. Remote Sens. **60**, 284–294 (2006)
49. Stewenius, H., Nister, D., Kahl, F., Schaffalitzky, F.: A minimal solution for relative pose with unknown focal length. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005) (2005)
50. Stewenius, H.: Gröbner basis methods for minimal problems in computer vision. Ph.D. thesis, Lund University, Sweden (2005)
51. Stewenius, H., Schaffalitzky, F., Nister, D.: How hard is 3-view triangulation really? In: International Conference on Computer Vision (ICCV). IEEE (2005)
52. Sturmfels, B.: Solving systems of polynomial equations. In: American Mathematical Society, CBMS Regional Conferences Series, No. 97 (2002)
53. Svärm, L., Enqvist, O., Kahl, F., Oskarsson, M.: City-scale localization for cameras with known vertical direction. IEEE Trans. Pattern Anal. Mach. Intell. **39**(7), 1455–1461 (2017)
54. Telen, S., Mourrain, B., Van Barel, M.: Solving polynomial systems via truncated normal forms. SIAM J. Matrix Anal. Appl. **39**(3), 1421–1447 (2018)
55. Telen, S., Van Barel, M.: A stabilized normal form algorithm for generic systems of polynomial equations. J. Comput. Appl. Math. **342**, 119–132 (2018)
56. van der Waerden, B.L., Artin, E., Noether, E., Benac, T.J.: Modern Algebra, vol. II. F. Ungar, New York (1950)
57. Zheng, E., Wang, K., Dunn, E., Frahm, J.: Minimal solvers for 3D geometry from satellite imagery. In: 2015 IEEE International Conference on Computer Vision (ICCV), pp. 738–746 (2015)

Snehal Bhayani is a PostDoctoral researcher with the Center for Machine Vision and Signal Analysis at the Faculty of Information Technology and Electrical Engineering, University of Oulu. He received the Doctor of Science in Computer Science and Engineering, University of Oulu. His research interests include 3D computer vision, multiple view geometry, and algebraic methods for solving equations.



Janne Heikkilä received the Doctor of Science in Technology degree in Information Engineering from the University of Oulu, Finland, in 1998. He is a professor of computer vision and digital video processing in the Faculty of Information Technology and Electrical Engineering, University of Oulu, and the head of the Center for Machine Vision and Signal Analysis. He is an IAPR Fellow, and a senior member of the IEEE. He has served as a member of editorial boards of several scientific journals and program committees of various international conferences. His research interests include 3D computer vision, machine learning, digital image and video processing, and biomedical image analysis. He has published more than 200 peer reviewed scientific articles in international journals and conferences.

Zuzana Kukelova is an Assistant Professor at Czech Technical University in Prague (CTU), working in the fields of 3D Computer Vision and Algebraic Geometry. Her research focuses on algebraic methods for solving systems of polynomial equations with applications in camera geometry estimation and robotics. She received her Ph.D. from CTU in 2013 and was a Postdoctoral Researcher at Microsoft Research Cambridge. Zuzana is the co-author of the first automatic generator of efficient polynomial equation solvers based on Grobner bases. She has worked on absolute and relative camera pose estimation problems for (partially) uncalibrated, semi-generalized and rolling shutter cameras, as well as on solvers based on SIFT correspondences. Her work was awarded the Best Paper Award at ACCV'18. Zuzana is the laureate of the 2023 Neuron Award for Promising Scientists in Computer Science and the winner of the 2015 Cor Baayen Young Researcher Award.