A sparse resultant based method for efficient minimal solvers

Snehal Bhayani1 Zuzana Kukelova2 Janne Heikkilä 1
1Center for Machine Vision and Signal Analysis, University of Oulu, Finland
2Visual Recognition Group, Faculty of Electrical Engineering, Czech Technical University in Prague

Abstract

Many computer vision applications require robust and efficient estimation of camera geometry. The robust estimation is usually based on solving camera geometry problems from a minimal number of input data measurements, i.e., solving minimal problems in a RANSAC framework. Minimal problems often result in complex systems of polynomial equations. Many state-of-the-art efficient polynomial solvers to these problems are based on Gröbner bases and the action-matrix method that has been automated and highly optimized in recent years. In this paper we study an alternative algebraic method for solving systems of polynomial equations, i.e., the sparse resultant-based method and propose a novel approach to convert the resultant constraint to an eigenvalue problem. This technique can significantly improve the efficiency and stability of existing resultant-based solvers. We applied our new resultant-based method to a large variety of computer vision problems and show that for most of the considered problems, the new method leads to solvers that are the same size as the best available Gröbner basis solvers and of similar accuracy. For some problems the new sparse-resultant based method leads to even smaller and more stable solvers than the state-of-the-art Gröbner basis solvers. Our new method can be fully automated and incorporated into existing tools for automatic generation of efficient polynomial solvers and as such it represents a competitive alternative to popular Gröbner basis methods for minimal problems in computer vision.

1. Introduction

Computing camera geometry is one of the most important tasks in computer vision [17] with many applications e.g. in structure from motion [39], visual navigation [38], large scale 3D reconstruction [19] and image localization [37].

The robust estimation of camera geometry is usually based on solving so-called minimal problems [35, 24, 23], i.e., problems that are solved from minimal samples of input data, inside a RANSAC framework [14, 9, 36]. Since the camera geometry estimation has to be performed many times in RANSAC [14], fast solvers to minimal problems are of high importance. Minimal problems often result in complex systems of polynomial equations in several variables. A popular approach for solving minimal problems is to design procedures that can efficiently solve only a special class of systems of equations, e.g., systems resulting from the 5-pt relative pose problem [35], and move as much computation as possible from the “online” stage of solving equations to an earlier pre-processing “offline” stage.

Most of the state-of-the-art specific minimal solvers are based on Gröbner bases and the action-matrix method [10]. The Gröbner basis method was popularized in computer vision by Stewenius [40]. The first efficient Gröbner basis solvers were mostly handcrafted [41, 42] and sometimes very unstable [43]. However, in the last 15 years much effort has been put into making the process of constructing the solvers more automatic [24, 29, 30] and the solvers stable [6, 7] and more efficient [29, 30, 28, 5, 32]. There are now powerful tools available for the automatic generation of efficient Gröbner basis solvers [24, 29].

While the Gröbner basis method for generating efficient minimal solvers was thoroughly studied in computer vision and all recently generated Gröbner basis 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-based method. The resultant-based method was manually applied to several computer vision problems [25, 16, 16, 20, 23, 25]. However in contrast to the Gröbner basis method, there is no general method for automatically generating efficient resultant-based minimal solvers. The most promising results in this direction were proposed by Emiris [12] and Heikkilä [18], where methods based on sparse resultants were proposed and applied to camera geometry problems. While these methods can be extended for general minimal problems that appear in computer vision and can be automated, they usually lead (due to linearizations) to larger and less efficient solvers than Gröbner basis solvers.

In this paper, we propose a novel approach to generating minimal solvers using sparse resultants, which is based on adding an extra equation of a special form to the in-
put system. Our algorithm is inspired by the ideas explored in [18, 12], but thanks to the special form of added equation and by solving the resultant as a small eigenvalue problem, in contrast to a polynomial eigenvalue problem in [18], the new approach achieves significant improvements over [18, 12] in terms of efficiency of the generated solvers. Specifically our contributions include,

- A novel sparse resultant-based approach to generating polynomial solvers based on adding an extra equation of a special form and transforming the resultant matrix constraint to a regular eigenvalue problem.
- Two procedures to reduce the size of resultant matrix that lead to faster solvers than the best available state-of-the-art solvers for some minimal problems.
- A general method for automatic generation of efficient resultant-based polynomial solvers for many important minimal problems that achieves competitive performance in terms of speed and stability with respect to the best available state-of-the-art solvers generated by highly optimized Gröbner basis techniques [29, 32]. The automatic generator of resultant-based solvers will be made publicly available at [45].

2. Theoretical background and related work

In this paper we use notation and basic concepts from the book by Cox et al. [10]. Our objective is to solve $m$ polynomial equations,

$$
\{f_1(x_1, \ldots, x_\nu) = 0, \ldots, f_m(x_1, \ldots, x_\nu) = 0\} \quad (1)
$$

in $\nu$ unknowns, $X = \{x_1, \ldots, x_\nu\}$. Let $\mathbb{C}[X]$ denote the set of all polynomials in unknowns $X$ with coefficients in $\mathbb{C}$. The ideal $I = (f_1, \ldots, f_m) \subset \mathbb{C}[X]$ is the set of all polynomial combinations of our 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. Using the ideal $I$ we can define the quotient ring $A = \mathbb{C}[X]/I$ which is the set of equivalence classes over $\mathbb{C}[X]$ defined by the relation $a \sim b \iff (a - b) \in I$. If $I$ has a zero-dimensional variety then the quotient ring $A = \mathbb{C}[X]/I$ is a finite-dimensional vector space over $\mathbb{C}$. 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 $A = \mathbb{C}[X]/I$.

2.1. Gröbner Basis method

Gröbner bases can be used to solve our system of polynomial equations (1). One of the popular approaches for solving systems of equations using Gröbner bases is the multiplication matrix method, known also as the action matrix method [10, 44]. This method was recently used to efficiently solve many of the minimal problems in computer vision [23, 24, 29, 32]. The goal of this method is to transform the problem of finding the solutions to (1) to a problem of eigendecomposition of a special multiplication matrix [11]. Let us consider the mapping $T_f : A \rightarrow A$ of the multiplication by a polynomial $f \in \mathbb{C}[X]$. $T_f$ is a linear mapping for which $T_f = T_g$ iff $f - g \in I$. In our case $A$ is a finite-dimensional vector space over $\mathbb{C}$ and therefore we can represent $T_f$ by its matrix with respect to some linear basis $B$ of $A$. For a basis $B = ([b_1], \ldots, [b_k])$ consisting of $k$ monomials, $T_f$ can be represented by $k \times k$ multiplication (action) matrix $M_f := (m_{ij})$ such that $T_f([b_j]) = [fb_j] = \sum_{i=1}^k m_{ij}[b_i]$. It can be shown [11] that $\lambda \in \mathbb{C}$ is an eigenvalue of the matrix $M_f$ iff $\lambda$ is a value of the function $f$ on the variety $V$. In other words, if $f$ is e.g. $x_n$ then the eigenvalues of $M_f$ are the $x_n$-coordinates of the solutions of (1). The solutions to the remaining variables can be obtained from the eigenvectors of $M_f$. This means that after finding the multiplication matrix $M_f$, we can recover the solutions by solving the eigendecomposition of $M_f$ for which efficient algorithms exist. Moreover, if the ideal $I$ is a radical ideal, i.e. $I = \sqrt{I}$, [11], then $k$ is equal to the number of solutions to the system (1). Therefore, Gröbner basis methods usually solve an eigenvalue problem of a size that is equivalent to the number of solutions of the problem. For more details and proofs we refer the reader to [10].

The coefficients of the multiplication matrix $M_f$ are polynomial combinations of coefficients of the input polynomials (1). For computer vision problems these polynomial combinations are often found “offline” in a pre-processing step. In this step, a so-called elimination template is generated, which is actually an expanded set of equations constructed by multiplying original equations with different monomials. This template matrix is constructed such that after filling it with coefficients from the input equations and performing Gauss-Jordan(G-J) elimination of this matrix, the coefficients of the multiplication matrix $M_f$ can be obtained from this eliminated template matrix.

The first automatic approach for generating elimination templates and Gröbner basis solvers was presented in [24]. Recently an improvement to the automatic generator [24] was proposed in [29] to exploit the inherent relations between the input polynomial equations and it results in more efficient solvers than [24]. The automatic method from [29] was later extended by a method for dealing with saturated ideals [30] and a method for detecting symmetries in polynomial systems [28].

In general, the answer to the question “What is the smallest elimination template for a given problem?” is not known. In [32] the authors showed that the method [29], which is based on the grevlex ordering of monomials and the so-called standard bases of the quotient ring $A$ is not
optimal in terms of template sizes. The authors of [29] proposed two methods for generating smaller elimination templates. The first is based on enumerating and testing all Gröbner bases w.r.t. different monomial orderings, i.e., the so-called Gröbner fan. By generating solvers w.r.t. all these Gröbner bases and using standard bases of the quotient ring $A$, smaller solvers were obtained for many problems. The second method goes “beyond Gröbner bases” and it uses a manually designed heuristic sampling scheme for generating “non-standard” monomial bases $B$ of $A = \mathbb{C}[X]/I$. This heuristic leads to more efficient solvers than the Gröbner fan method in many cases. While the Gröbner fan method will provably generate at least as efficient solvers as the grevlex-based method from [29], no proof can be in general given for the “heuristic-based” method. The proposed heuristic sampling scheme uses only empirical observations on which basis monomials will likely result in small templates and it samples a fixed number (1000 in the paper) of candidate bases consisting of these monomials. Even though, e.g. the standard grevlex monomial basis will most likely be sampled during the sampling, it is in general not clear how large templates it will generate for a particular problem. The results will also depend on the number of bases tested inside the heuristic.

2.2. Sparse Resultants

An alternate approach towards solving polynomial equations is that of using resultants. Simply put, a resultant is an irreducible polynomial constraining coefficients of a set of $n + 1$ polynomials, $F = \{f_1(x_1, \ldots, x_n), \ldots, f_{n+1}(x_1, \ldots, x_n)\}$ in $n$ variables to have a non-trivial solution. One can refer to Cox et al. [10] for a more formal theory on resultants. We have $n+1$ equations in $n$ variables because resultants were initially developed to determine whether a system of polynomial equations has a common root or not. If a coefficient of monomial $x^\alpha$ in the $i^{th}$ polynomial of $F$ is denoted as $u_{i,\alpha}$ the resultant is a polynomial $\text{Res}(u_{i,\alpha})$ with $u_{i,\alpha}$ as variables.

Using this terminology, the basic idea for a resultant based method is to expand $F$ to a set of linearly independent polynomials which can be linearised as $M([u_{i,\alpha}])b$, where $b$ is a vector of monomials of form $x^\alpha$ and $M([u_{i,\alpha}])$ has to be a square matrix that has full rank for generic values of $u_{i,\alpha}$, i.e. det $M([u_{i,\alpha}]) \neq 0$. The determinant of the matrix $M([u_{i,\alpha}])$ is a non-trivial multiple of the resultant $\text{Res}(u_{i,\alpha})$ [10]. Thus det $M([u_{i,\alpha}])$ must vanish, if the resultant vanishes, i.e. $\text{Res}(u_{i,\alpha}) = 0 \implies \text{det} M([u_{i,\alpha}]) = 0$. It is known that $\text{Res}(u_{i,\alpha})$ vanishes iff the polynomial system $F$ has a solution [10]. This gives us the necessary condition for the existence of roots of $F = 0$. Hence the equation det $M([u_{i,\alpha}]) = 0$ gives us those values of $u_{i,\alpha}$ such that $F = 0$ have a common root.

Resultants can be used to solve $n$ polynomial equations in $n$ unknowns. The most common approach used for this purpose is to hide a variable by considering it as a constant. By hiding, say $x_n$, we obtain $n$ polynomials in $n-1$ variables, so we can use the concept of resultants and compute $\text{Res}([u_{i,\alpha}], x_n)$ which now becomes a function of $u_{i,\alpha}$ as well as $x_n$. Algorithms based on hiding a variable attempt to expand $F$ to a linearly independent set of polynomials that can be re-written in a matrix form as

$$M([u_{i,m}], x_n)b = 0,$$  \hspace{2cm} (2)

where $M([u_{i,m}], x_n)$ is a square matrix whose elements are polynomials in $x_n$ and coefficients $u_{i,\alpha}$ and $b$ is the vector of monomials in $x_1, \ldots, x_{n-1}$. For simplicity we will denote the matrix $M([u_{i,\alpha}], x_n)$ as $M(x_n)$ in the rest of this section. Here we actually estimate a multiple of the actual resultant via the determinant of the matrix $M(x_n)$ in (2). This resultant 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 see [10]. Such a hidden variable approach has been used in the past to solve various minimal problems [16, 20, 23, 25].

The most common way to solve the original system of polynomial equations is to transform (2) to a polynomial eigenvalue problem (PEP) [11] that transforms (2) as

$$(M_0 + M_1 x_n + \ldots + M_l x_n^l)b = 0,$$  \hspace{2cm} (3)

where $l$ is the degree of the matrix $M(x_n)$ in the hidden variable $x_n$ and matrices $M_0, \ldots, M_l$ are matrices that depend only on the coefficients $u_{i,\alpha}$ of the original system of polynomials. The PEP (3) can be easily converted to a generalized eigenvalue problem (GEP):

$$Ay = x_nBy,$$  \hspace{2cm} (4)

and solved using standard efficient eigenvalue algorithms [25]. Basically, the eigenvalues give us the solution to $x_n$ and the rest of the variables can be solved from the corresponding eigenvectors, $y$ [10]. But this transformation to a GEP relaxes the original problem of finding the solutions to our input system and computes eigenvectors that do not satisfy the monomial dependencies induced by the monomial vector $b$. And many times it also introduces extra parasitic (zero) eigenvalues leading to slower polynomial solvers.

Alternately, we can add a new polynomial

$$f_{n+1} = u_0 + u_1 x_1 + \cdots + u_n x_n$$  \hspace{2cm} (5)

to $F$ and compute a so called u-resultant [10] by hiding $u_0, \ldots, u_n$. In general random values are assigned to $u_1, \ldots, u_n$. The $u$-resultant matrix is computed from these $n+1$ polynomials in $n$ variables in a way similar to the one explored above. For more details one can refer to [10].
For sparse polynomial systems it is possible to obtain more compact resultants using specialized algorithms. Such resultants are commonly referred to as Sparse Resultants. A sparse resultant would mostly lead to a more compact matrix $M(x_n)$ and hence a smaller eigendecomposition problem. Emiris et al. [13, 8] have proposed a generalised approach for computing sparse resultants using mixed-subdivision of polytopes. Based on [13, 8] Emiris proposed a method for generating a resultant-based solver for sparse systems of polynomial equations, that was divided in “offline” and “online” computations. The resulting solvers were based either on the hidden-variable trick (2) or the $u$-resultant of the general form (5). As such the resulting solvers were usually quite large and not very efficient. More recently Heikkilä [18] have proposed an improved approach to test and extract smaller resultants. This method transforms (2) to a GEP (4) and solves for eigenvalues and eigenvectors to compute solutions to unknowns. The methods [8, 12, 13, 18] suffer from the drawback that they require the input system to have as many polynomials as unknowns to be able to compute a resultant. Additionally, the algorithm [18] suffers from other drawbacks and cannot be directly applied to most of the minimal problems. These drawbacks can be overcome, as we describe in the supplementary material. However, even with our proposed improvements the resultant-based method [18], which is based on hiding one of the input variables in the coefficient field, would result in a GEP with unwanted eigenvalues and in turn unwanted solutions to original system (1). This leads to slower solvers for most of the studied minimal problems.

Therefore, we investigate an alternate approach where instead of hiding one of the input variables [12, 18] or using $u$-resultant of a general form (5) [12], we introduce an extra variable $\lambda$ and a new polynomial of a special form, i.e., $x_i - \lambda$. The augmented polynomial system is solved by hiding $\lambda$ and reducing a constraint similar to (2) into a regular eigenvalue problem that leads to smaller solvers than [12, 18]. Next section lays the theoretical foundation of our approach and outlines the algorithm along with the steps for computing a sparse resultant matrix $M(\lambda)$.

### 3. Sparse resultants using an extra equation

We start with a set of $m$ polynomials from (1) in $n$ variables $x_1, \ldots, x_n$ to be solved. Introducing an extra variable $\lambda$ we define $x' = [x_1, x_2, \ldots, x_{n+1}, \lambda]$ and an extra polynomial $f_{m+1}(x') = x_{n+1} - \lambda$. Using this, we propose an algorithm inspired by [18] and [12] to solve the following augmented polynomial system for $x'$,

$$f_1(x') = 0, \ldots, f_{m}(x') = 0, f_{m+1}(x') = 0. \tag{6}$$

Our idea it to compute its sparse resultant matrix $M(\lambda)$ by hiding $\lambda$ in a way that allows us to solve (6) by reducing its linearization (similar to (2)) to an eigenvalue problem.

### 3.1. Sparse resultant and eigenvalue problem

Our algorithm computes the monomial multiples of the polynomials in (6) in the form of a set $T = \{T_1, \ldots, T_m, T_{m+1}\}$ where each $T_i$ denotes the set of monomials to be multiplied by $f_i(x')$. We may order monomials in each $T_i$ to obtain a vector form, $T_i = \text{vec}(T_i)$ and stack these vectors as $T = [T_1, \ldots, T_m, T_{m+1}]$. The set of all monomials present in the resulting extended set of polynomials $\{x^\alpha f_i(x'), \forall x^\alpha \in T_i, i = 1, \ldots m + 1\}$ is called the monomial basis and is denoted as $B = \{x^\alpha | \alpha \in \mathbb{Z}_{\geq 0}^n\}$. The vector form of $B$ w.r.t. some monomial ordering is denoted as $b$. Then the extended set of polynomials can be written in a matrix form,

$$M b = 0, \tag{7}$$

The coefficient matrix $M$ is a function of $\lambda$ as well as the coefficients of input polynomials (6). Let $\varepsilon = |B|$. Then by construction [18] $M$ is a tall matrix with $p \geq \varepsilon$ rows. We can remove extra rows and form an invertible square matrix which is the sparse resultant matrix mentioned in previous section. While Heikkilä [18] solve a problem similar to (7) as a GEP, we exploit the structure of newly added polynomial $f_{m+1}(x')$ and propose a block partition of $M$ to reduce the matrix equation of (7) to a regular eigenvalue problem.

**Proposition 3.1.** Let $f_{m+1}(x') = x_n - \lambda$, then there exists a block partitioning of $M$ in (7) as:

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}, \tag{8}$$

such that (7) can be converted to an eigenvalue problem of the form $Xb' = \lambda b'$.

**Proof:** In order to block partition the columns in (8) we need to partition $B$ as $B = B_\lambda \cup B_c$ where

$$B_\lambda = B \cap T_{m+1}, \quad B_c = B - B_\lambda. \tag{9}$$

Let us order the monomials in $B_c$, such that $b = \text{vec}(B) = [\text{vec}(B_\lambda) \ \text{vec}(B_c)]^T = [b_1 \ b_2]^T$. Such a partition of $b$ induces a column partition of $M$ (7). We row partition $M$ such that the lower block is row-indexed by monomial multiples of $f_{m+1}(x')$ which are linear in $\lambda$ (i.e. $x^\alpha (x_n - \lambda)$, $x^\alpha \in T_{m+1}$) while the upper block is indexed by monomial multiples of $f_{1}(x'), \ldots, f_{m}(x')$. Such a row and column partition of $M$ gives us a block partition as in (8). As $[M_{11} \ M_{12}]$ contains polynomials independent of the $\lambda$ and $[M_{21} \ M_{22}]$ contains polynomials of the form $x^\alpha (x_n - \lambda)$ we obtain

$$M_{11} = A_{11}, \quad M_{12} = A_{12}, \quad M_{21} = A_{21} + \lambda B_{21}, \quad M_{22} = A_{22} + \lambda B_{22}, \tag{10}$$

where $A_{11}, A_{12}, A_{21}$ and $A_{22}$ are matrices dependent only on the coefficients of input polynomials in (6). We assume here
that $A_{12}$ has full column rank. Substituting (10) in (8) gives

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 \\ B_{21} & B_{22} \end{bmatrix}$$

(11)

We can order monomials so that $T_{m+1} = b_1$. Now chosen partition of $M$ implies that $M_{21}$ is column indexed by $b_1$ and row indexed by $T_{m+1}$. As $[M_{21} M_{22}]$ has rows of form $x^{\alpha_j}(x_i - \lambda), x^{\alpha_j} \in T_{m+1} \implies x^{\alpha_j} \in B_\lambda$. This gives us, $B_{21} = -I$, where $I$ is an identity matrix of size $|B_\lambda|$ and $B_{22}$ is a zero matrix of size $|B_\lambda| \times |B_c|$. This also means that $A_{21}$ is a square matrix of same size as $B_{21}$. Thus we have a decomposition as

$$M = M_0 + \lambda M_1 = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 \\ -I & 0 \end{bmatrix},$$

(12)

where $M$ is a $p \times \varepsilon$ matrix. If $M$ is a tall matrix, so is $A_{12}$ from which we can eliminate extra rows to obtain a square invertible matrix $A_{12}$ while preserving the above-mentioned structure, as discussed in Section 3.3. Let $b = [b_1 \ b_2]^T$. Then from (7) and (12) we have

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 \\ -I & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = 0$$

$$\implies A_{21} b_1 + A_{22} b_2 - \lambda b_1 = 0$$

(13)

Eliminating $b_2$ from the above pair of equations we obtain

$$\begin{bmatrix} \lambda A_{21} \hat{A}_{12}^\dagger A_{11} \hat{A}_{12}^\dagger \hat{A}_{11} \end{bmatrix} b_1 = \lambda b_1,$$

(14)

which is the schur complement of $A_{12}$. If $A_{12}$ does not have full column rank, we change the partitioning of columns of $M$ by changing the partitions, $B_{\alpha} = \{x^{\alpha} \in T_{m+1} | x_i x^{\alpha} \in B\}$ and $B_\alpha = B - B_{\alpha}$ by exploiting the form of $f_{m+1}(x')$. This gives us $A_{21} = I$ and $A_{22} = 0$. It also results in a different $A_{12}$ and a different $A_{12}$ after removing extra rows. Hence from (12) we have

$$M = M_0 + \lambda M_1 = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & 0 \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 \\ B_{21} & B_{22} \end{bmatrix},$$

(15)

which is substituted in (7) to get $\hat{A}_{11} b_1 + \hat{A}_{12} b_2 = 0$ and $\lambda (B_{21} b_1 + B_{22} b_2) + b_1 = 0$. Eliminating $b_2$ from these equations we get an alternate eigenvalue formulation:

$$(B_{21} - B_{22} \hat{A}_{12}^\dagger \hat{A}_{11}) b_1 = -(1/\lambda) b_1.$$
monomial bases so computed and selects the smallest monomial basis \( B \) among them along with the corresponding set of monomial multiples \( T \) from which the coefficient matrix \( \mathbb{M} \) is constructed as described in Section 3.1.

Next, we list the prominent features of our approach and how they seek to address the shortcomings of \([12, 18]\):

1. We attempt to generate the smallest basis \( B \) by testing adding an extra polynomial (5) of a special form \( x_i - \lambda \) for each \( i \) in \( 1, \ldots, n \).
2. We explicitly test for rank of \( \mathbb{M} \) for each candidate basis \( B \) to ensure that we have a full rank solver. This addresses the issue of rank-deficient solvers in \([18]\).
3. The partition of monomial basis, \( B = B_\lambda \sqcup B_c \) (9) or the alternate partition of \( B \) as described in Proposition 3.1) highlights our approach that leads to a favourable decomposition of the coefficient matrix \( \mathbb{M} \) as in (12), for solving (7) as an eigenvalue problem. This helps us compute much smaller and more stable solvers as compared to ones generated in \([12, 13, 18]\).
4. The special form of the extra polynomial aids us to construct \( \mathbb{M} \) that is largely smaller than the one constructed by general \( \nu \)-resultant solvers in \([12]\).
5. Our method can generate solvers for \( m \geq n \) in (1).

### 3.3. Removing columns from coefficient matrix

The next step in our method is to attempt to reduce the size of the coefficient matrix \( \mathbb{M} \) computed in the previous section. For this, we select columns of \( \mathbb{M} \) one by one in a random order to test for their removal. For each such column, we select rows (say \( r_1, \ldots, r_k \)) that contain non-zero entries in the column and also consider all columns (say \( c_1, \ldots, c_l \)) that have non-zero entries in \( r_1, \ldots, r_k \). Then we can remove these \( k \) rows and \( l \) columns from \( \mathbb{M} \) only if the following conditions hold true for the resulting reduced matrix \( \mathbb{M}_{\text{red}} \).

This also means that we would be removing monomials from \( B \) that index \( c_1, \ldots, c_l \) and removing monomials from \( T \) that index \( r_1, \ldots, r_k \).

1. After eliminating the monomials from \( T \), we require that there is at least one monomial left in each \( T_j \).
2. If \( \mathbb{M} \) is of size \( p \times \varepsilon \), the reduced matrix \( \mathbb{M}_{\text{red}} \) would be of size \( (p - k) \times (\varepsilon - l) \). Then we require \( p - k \geq \varepsilon - l \) and \( \text{rank}(\mathbb{M}_{\text{red}}) = \varepsilon - l \).
3. \( \mathbb{M}_{\text{red}} \) must be block partitioned and decomposed as in Proposition 3.1.

We repeat the above process until there are no more columns that can be removed. We 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 (14) (or alternately (16)). Now, reusing the notation, let’s denote \( \mathbb{M} \) to be the reduced coefficient matrix and denote \( B \) and \( T \) to be reduced monomial basis and set of monomial multiples, respectively.

If \( \mathbb{M} \) 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. These rows are chosen in a way so that the three conditions mentioned above are still 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 \( |T_{m+1}|(= |B_\lambda|) \) as much as possible and ensure that the matrix \( A_\lambda \) and hence \( X \) (14)(or in (16)) 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 \( \{T_1, \ldots, T_m\} \). Detailed algorithms for these two steps of matrix reduction are provided in the supplementary material. But we note that at the end of these two steps, we have the sparse resultant matrix, \( \mathbb{M} \) satisfying (7) which is then reduced to the eigenvalue formulation (14) or to the alternate formulation (16).

### 4. Experiments

We evaluate the performance of our method by comparing the stabilities as well as computational complexities of the solvers generated using our method with the state-of-art Gröbner basis 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 that studied in \([32]\). Results for additional problems are provided in the supplementary material.

#### 4.1. Evaluation

The comparison of the computational complexity of minimal solvers is based on the sizes of matrix templates to be solved. E.g. a solver of size \( 11 \times 20 \) in the table means inverting a \( 11 \times 11 \) matrix and then a computation of \( 20 - 11 = 9 \) eigenvalues and eigenvectors. So in Table 1 we compare the size of templates in our resultant-based solvers with the templates used in state-of-the-art Gröbner basis solvers as well as in the original solvers proposed by the respective authors (see column 3). The Gröbner basis solvers used for comparison include the solvers generated using the approach in \([29]\), the Gröbner fan and heuristic-based approaches in \([32]\). As we can see from Table 1, our new resultant-based approach leads to the smallest templates and hence fastest solvers for most of the minimal problems while for only a few problems our generated solver is slightly larger than the state-of-the-art solver based on the Gröbner fan or the heuristic-based method \([32]\). For some solvers though we have a slightly larger eigenvalue problem, the overall template size is considerably smaller. E.g. in the problem of estimating the relative pose and radial distortion parameter from 6pt correspondences \([24]\) we have an eigenvalue problem of size \( 56 \times 56 \) and matrix inversion of size \( 39 \times 39 \) whereas the heuristic-based solver has a \( 52 \times 52 \) eigenvalue problem but inversion of a larger
with Gröbner basis solvers. As it is not feasible to generate a Gröbner fan solver in reasonable time. It is worth noting that here we do not compare our solvers’ sizes with resultant-based solvers generated by original versions of [18] and [12]. These methods can not be directly applied to most of the studied minimal problems as they can not handle more equations than unknowns. With [18] we also failed to generate full rank solvers for some problems. Even after proposing extensions to these methods [18, 12], the generated solvers were larger than ours, and GEP involved in [18] led also to many unwanted solutions. We give the sizes of these solvers in supplementary material along with a brief description of our proposed improvements to [18].

We evaluate and compare the stabilities of our solvers from Table 1 with Gröbner basis solvers. As it is not feasible to generate scene setups for all considered problems, we instead evaluate the stability of minimal solvers using 5K instances of random data points. Stability measures include mean and median of \( \log_{10} \) of normalized equation residuals for computed solutions as well as the solvers failures as a % of 5K instances for which at least one solution has a normalized residual > 10^{-3}. These measures on randomly generated inputs have been shown to be sufficiently good indicators of solver stabilities [29]. Table 2 shows stabilities of solvers for six minimal problems selected from Table 1. We note that for the “Rel.pose \( \lambda+E+\lambda \)” problem, our solver is not only faster, but also more stable than the state-of-the-art solvers whose histogram is provided in the supplementary material along with the stabilities for the remaining problems and the histograms of their residuals. In general, our new method generates solvers that are stable with only very 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 2 (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 1, column 3). 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+fλ solver on synthetic scenes:** We study the numerical stability of the new resultant-based solver for the problem of estimating the relative pose of one calibrated and one camera with unknown focal length and radial distortion from 7-point correspondences, i.e. the Rel. pose E+fλ 7pt problem from Table 1. We considered the formulation “elim. \( \lambda \)” proposed in [32] that leads to the smallest solvers. We studied the performance on noise-free data and compared it to the results of Gröbner basis solvers from Table 1.

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 \( \lambda_{gt} \) was drawn random from the interval \([-0.7, 0]\) representing distortions of cameras with a small distortion up to slightly more than GoPro-style cameras. A graph for the \( \log_{10} \) of the relative errors of the distortion parameter \( \lambda \) as well as the focal length \( f \) are provided in the supplementary material.

**P4Pfr solver on real images:** We evaluated the 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 [26] and in [31] 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 Figure 1 (top). The Reality Capture software [1] 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 x 40) in a RANSAC framework. Similar to [31], we used the camera and distortion parameters obtained from [1] as ground...
Here we propose a novel algorithm for generating efficient minimal solvers based on sparse resultants that achieves significant improvements over existing resultant-based methods in terms of efficiency of the generated solvers. Our experiments on many minimal problems on real and synthetic scenes show that the new method is a competitive alternative to the highly optimised Gröbner basis methods. The fact that new resultant-based solvers have for many problems the same size as the state-of-the-art heuristic or GFan solvers, shows that these solvers are maybe already "optimal" w.r.t. template sizes. On the other hand, there is no one general method (GFan/heuristic/resultant), which proves the highly optimised Gröbner basis methods. The fact that new resultant-based solvers have for many problems the same size as the state-of-the-art heuristic or GFan solvers, shows that these solvers are maybe already "optimal" w.r.t. template sizes. On the other hand, there is no one general method (GFan/heuristic/resultant), which prov-

5. Conclusion

Here we propose a novel algorithm for generating efficient minimal solvers based on sparse resultants that achieves significant improvements over existing resultant-based methods in terms of efficiency of the generated solvers. Our experiments on many minimal problems on real and synthetic scenes show that the new method is a competitive alternative to the highly optimised Gröbner basis methods. The fact that new resultant-based solvers have for many problems the same size as the state-of-the-art heuristic or GFan solvers, shows that these solvers are maybe already "optimal" w.r.t. template sizes. On the other hand, there is no one general method (GFan/heuristic/resultant), which prov-

6. Acknowledgement

The authors would like to thank Academy of Finland for the financial support of this research (grant no. 297732). Zuzana Kukelova was supported by OP RDE project International Mobility of Researchers MSCA-IF at CTU Reg. No. CZ.02.2.69/0.0/0.0/17_050/0008025 and by the ERC-CZ grant MSMT LL1901.
References


