Making minimal solvers inverse-free using null space computation

Hassan Bozorgmanesh Janne Heikkilä
Center for Machine Vision and Signal Analysis (CMVS),
Faculty of Information Technology and Electrical Engineering, University of Oulu, Finland
hassan.bozorgmanesh@oulu.fi janne.heikkila@oulu.fi

Abstract

In this paper, we propose a novel resultant-based method for solving polynomial systems of equations that are commonly encountered in computer vision, particularly as minimal problems. Unlike traditional algorithms that rely on matrix inversion, the primary merits of the proposed method are the numerical stability of its formulation and the lack of need to compute the inverse of a matrix by leveraging null space computations. Additionally, its formulation paves the way for more computations to be performed in the offline stage by using the sparsity of coefficient matrices, thereby reducing the computational load in the online stage. This inverse-free formulation is especially suited for sparse systems and offers improved robustness in scenarios where matrix inversion is unstable or infeasible. Experimental results demonstrate better accuracy compared to state-of-the-art methods such as SRBM and GAPS across a variety of camera geometry problems.

1. Introduction

Solving polynomial systems of equations holds a special place in computer vision, as many camera geometry problems such as camera pose estimation, triangulation, and structure-from-motion can be modeled using polynomials derived from minimal sets of data—commonly referred to as minimal problems. A minimal solver computes solutions to these minimal problems and is typically used within RANSAC (Random Sample Consensus) frameworks [15, 16].

In this application, systems with a particular structure of coefficients need to be solved repeatedly, as RANSAC randomly samples minimal subsets of data to hypothesize models. Additionally, validating models across different data subsets requires solving the minimal problem multiple times. In the context of SLAM (Simultaneous Localization and Mapping), minimal problems must be solved continuously and repeatedly [12]. Therefore, fast computation is essential to maintain high performance.

A common strategy is to symbolically manipulate a polynomial system so that, after substituting numerical values into its coefficients, the solution process is as fast as possible. The first stage, involving symbolic computations, is referred to as the offline stage, while the second stage, involving numerical calculations, is known as the online stage. It is preferable to perform as many computations as possible during the offline stage, as this can significantly accelerate the online phase.

There are two main classes of methods for solving polynomial systems of equations in minimal problems: Groebner basis methods and resultant-based methods. Groebner basis methods have been widely used in the context of minimal solvers [6, 27], while resultant-based methods have only recently gained attention. Groebner basis techniques are sensitive to coefficient perturbations and often require careful tuning. Resultant-based methods, on the other hand, offer a promising alternative but typically depend on identifying invertible or full-rank submatrices, which can be computationally expensive and numerically unstable [1, 13].

In this paper, we propose a method based on null space computation that does not require an invertible matrix at any stage. Moreover, since the coefficient matrices of polynomial systems are often sparse, this sparsity can be exploited to compute the null space either partially or entirely during the offline stage. We demonstrate that our method outperforms existing techniques in terms of accuracy and efficiency, particularly in scenarios where traditional methods struggle due to ill-conditioned matrices or sparsity. Overall, we can state the following merits of the proposed method:

- Offline stage:
  - Possibility to compute the null space entirely during the offline stage, or use a hybrid offline-online approach
  - No need to trim or manipulate the coefficient matrix
  - No need to find an invertible submatrix within the coefficient matrix
  - More suitable for sparse coefficient matrices
- Online stage:
  - Improved accuracy
  - Reduced sensitivity to numerical instability

In the next section, we briefly review the Groebner-based and resultant-based methods to solve a polynomial system of equations. In Section 3, we will describe the proposed method and cover how the null space computation can be done partially or completely offline. In Section 4, the proposed method is compared with a resultant-based and a Groebner-based method and finally Section 5 covers the conclusion.

2. Polynomial systems of equations

The goal is to solve the following system

$$f_1(x_1, x_2, ..., x_n) = ... = f_m(x_1, x_2, ..., x_n) = 0 \quad (1)$$

where n is the number of variables, m is the number of equations and each f_i is a polynomial in n variables x_1,...,x_n. Every polynomial system of equations can be written as

$$M\mathbf{x} = 0 \quad (2)$$

where M is the coefficient $m \times n$ matrix and $\mathbf{x}$ is a vector containing all monomials of the system.

It is usual to extend the polynomial system by creating new equations by multiplying exisiting ones by some monomials. This action, while preserving the solutions, enables elimination techniques such as Gaussian elimination to be performed by simplifying the forms. It also can make a coefficient matrix sparser, this can be useful for methods especially designed for sparse matrices. For resultant-based methods, this is a crucial step to make the coefficient matrix square. It is also required for the resulting coefficient matrix to be full-rank.

2.1. Groebner-based methods

Groebner-based methods can be used to convert a system of polynomial equations into an easier-to-solve equivalent system—for example, one in which an equation involves only a single variable and can therefore be solved using methods for single-variable polynomials.

One of the drawbacks of Groebner basis methods is their sensitivity to fluctuations in coefficients [18]. To use them in practical applications, one approach is to extend their definitions to enhance stability.

One of the drawbacks of Groebner basis computations in numerical settings is their sensitivity to floating-point perturbations in the coefficients [18]. Although Groebner bases are algebraically stable under generic specializations (see [7, Section 4.7, Theorem 2]), their numerical computation can be unstable, motivating extensions and modified formulations that improve robustness in practical applications.

An important subgroup of the Groebner-based methods for minimal solvers is action matrix-based methods [22, 25]. After constructing a quotient ring from the polynomial system and choosing a basis for it, a linear map is defined that connects multiplication by a polynomial to a matrix representation—this matrix is known as the action matrix. It encodes the structure of the polynomial system and allows us to solve it by computing eigenvalues and eigenvectors. Effectively, the action matrix is obtained by using Gaussian elimination on the matrix that encodes the polynomial system (called elimination template). The set of monomials is divided into three parts and based on this the elimination template is reduced. For our comparsions, we use a variant of action matrix-based methods called GAPS [22, 25].

2.2. Resultant-based methods

Simply put, the resultant of a system of polynomial equations is a polynomial in the coefficients of the system such that, for a given set of coefficients, the system has a common solution if and only if the resultant is equal to zero [3, 13].

One issue with resultant-based methods is the emergence of extra solutions. This occurs because the list of monomials is inherently connected, and linearizing the system fails to account for these connections, leading to extraneous solutions. A common goal in these methods is to eliminate or detect such solutions [3].

2.2.1 Hiding a variable

When using resultants, it is necessary for the number of equations to exceed the number of variables by one. However, systems typically have an equal number of equations and variables. To address this, one variable is often treated as a scalar [2]. In other words, to apply the properties of resultants to a square system, we treat one of the variables as a coefficient—effectively hiding it. This transforms the resultant into a polynomial in the hidden variable. Solving this polynomial then reveals the values of the hidden variable that corresponds to the solutions of the original system.

2.2.2 Adding an equation

Another approach to leveraging resultants involves adding an extra equation to the system, enabling the computation of a resultant polynomial from which the solutions can be extracted.

One major class of methods in this group is the u-resultant [17]. In this method, we add an equation of the form:

$$f_{n+1} = u_0 + u_1 x_1 + \ + u_n x_n \quad (3)$$

and then assign random values to all but one u_i ($i \neq 0$) and then hide the remaining one. Considering the linearity of the extra constraint, one of the key advantages of the u-resultant method is its ability to more effectively track the hidden variable within the coefficient matrix. This property

can be particularly useful when designing methods based on linear algebra.

A resultant-based model suggested by Bhayani et. al. [1] called Sparse Resultant-Based Minimal Solvers (SRBM) uses Schur complement of Macaulay matrix in order to convert the extended system of polynomial equations into an eigenvalue problem.

First, they add an extra equation such as $x_i - \lambda$ to the system (where λ is a new variable that will be hidden subsequently). Then, they divide the multiplication matrix for the extended system as follows:

$$M\mathbf{x} = \left(\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} + \lambda \begin{bmatrix} 0 & 0 \\ -I & 0 \end{bmatrix} \right) \begin{bmatrix} \mathbf{x_1} \\ \mathbf{x_2} \end{bmatrix} \quad (4)$$

Using the Schur complement, they have the following generialized eigenvalue problem:

$$(A_3 - A_4 A_2^{-1} A_1)\mathbf{x_1} = \lambda \mathbf{x_1}. \quad (5)$$

We use this method later for our comparisons with the proposed method. It has been proven [3] that SRBM and action-based methods can give the equivalent solver under some conditions and it therefore proves the close connection of resultant-based and Groebner-based methods.

The main drawback of this method is the need to have an invertible A_2. In practice, A_2 can be a badly scaled matrix that can greatly influence the accuracy of the method. Additionally, in the offline part, it can be difficult or even impossible to find a non-singular A_2. In the paper [1], the authors use a trimming strategy to remove extra rows and columns in order to find and locate a non-singular A_2. However, it is not guaranteed that such a matrix can be found, and the process can also be computationally expensive. In some examples, the method may get stuck during the offline stage of trimming the Macaulay matrix. Therefore, in practice, for a sparse Macaulay matrix, it would be usually impossible to find an invertible matrix without trimming. Thus, trimming creates a submatrix which is non-singular.

Also, calculating the inverse of a matrix during the online stage each time can be time-consuming, and for certain sets of given coefficients, we might obtain a matrix with a very large condition number.

Another issue that can arise, especially in problems with more variables, is that the monomials included in $\mathbf{x_1}$ do not necessarily contain information about all variables. As a result, it may not be possible to extract certain variable values from the eigenvectors simply by dividing some of their elements.

3. Proposed Method

As previously mentioned, methods such as SRBM require identifying a non-singular submatrix within the Macaulay matrix. This process can lead to numerical instability and, in some cases, it may not be feasible to find such a submatrix. It can be said that SRBM performs best with dense matrices. As we will see, the proposed method can actually be more effective for sparse matrices, while still remaining applicable to dense coefficient matrices.

In our approach, we follow a similar strategy to SRBM by adding a linear equation that incorporates the difference of a predetermined variable. The new variable—essentially equivalent to one of the original ones—is then hidden in the method.

The key innovation of the proposed method lies in using the null space instead of matrix inversion to find solutions to the polynomial system, which can be derived from an eigenvalue problem. Utilizing the null space offers greater numerical stability and allows for partial or full computation during the offline phase, thereby accelerating the online phase. It also eliminates the need to use or identify a non-singular matrix.

1) (Offline) First, choose a variable such as x_i and add the equation $x_i - \lambda$ to the system, where λ is the extra variable.
2) (Offline) Hide the extra variable λ.
3) (Offline) Extend the given polynomial system so that we have $\bar{M}\mathbf{x} = 0$ with a square $\bar{M}$ and $\mathbf{x}$ is the extended vector of monomials.
4) (Offline) Divide the rows of extended multiplication matrix such that

$$\bar{M} = \begin{bmatrix} A \\ B(\lambda) \end{bmatrix} = \begin{bmatrix} A \\ B_1 \end{bmatrix} - \lambda \begin{bmatrix} 0 \\ B_2 \end{bmatrix} \quad (6)$$

   where A contains the coefficients of equations that are multiplicands of the original equations, and $B(\lambda)$ contains coefficients of equations that are multiplicand of the extra equation. Considering the linearity of the extra equation, we can write $B(\lambda) = B_1 - \lambda B_2$. Note that elements of both B_1 and B_2 are mostly zeros with some ones.
5) (Offline or hybrid offline-online or online) Calculate the null space of A as Z (Section 3.2).
6) (Offline or online) Considering that $A\mathbf{x} = 0$, we have $\mathbf{x} = Z\mathbf{y}$ and using the lower part of Equation (6), we can write

$$B_1 Z\mathbf{y} = \lambda B_2 Z\mathbf{y} \quad (7)$$

   which is a generalized eigenvalue problem.
7) (Online) The eigenvalues of this problem are the x_i components of the solutions. The eigenvectors need to be multiplied by Z in order to give the final monomial vectors, which contain the solutions of the problem.

Note that in this algorithm, aside from Step 7—which is computed numerically during the online stage—the remaining steps can be performed during the offline stage. However, it is also possible to perform the computation of null

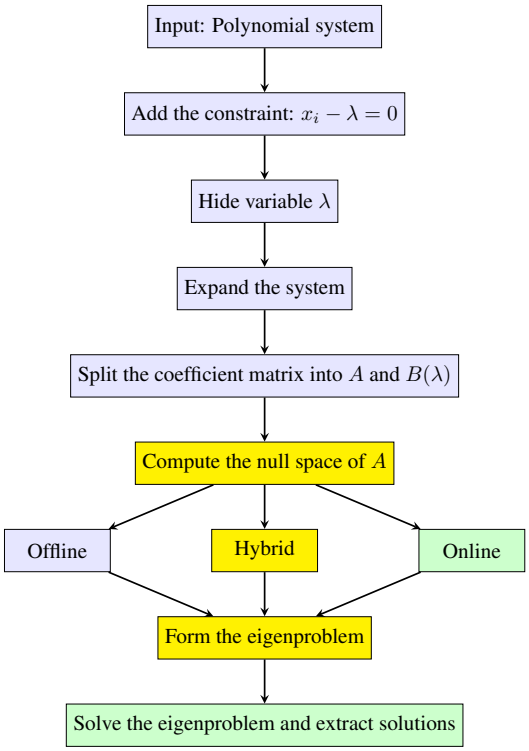


Figure 1. A diagram showcasing the flow of the proposed algorithm. Blue and green nodes represent offline and online stages, respectively. Yellow nodes represent steps that can be done in a hybrid manner.

space in a hybrid offline-online manner or entirely online. If Step 5 is performed hybrid offline-online or completely online, then subequently Step 6 is also required to be performed online. Further explanation about different stages of the algorithm will be provided in the next section.

3.1. Step 1-4: Preparing the system

At the beginning, we define a new variable, λ, and set it equal to an existing variable, such as x_i. This is done by adding a new constraint, $x_i - \lambda = 0$, to the system. Since this constraint is linear in the new variable, its simple structure can be advantageous for formulating an eigenvalue problem to subsequently solve the system.

Then, the system is expanded by multiplying its constraints by monomials. In this paper, we use the extended elimination templates of SRBM. After doing so, the resulting square coefficient system consists of two parts: the multiplicands of the original equations and the new equation. Therefore, it can be rewritten as Equation 6.

For the problems that we applied our proposed method to (see Section 4), between 20% and 80% of the rows in the coefficient matrix are multiplicands of the new equation. This percentage can be one of the important indicators when deciding whether to compute the null space during the offline stage, in a hybrid manner, or entirely online, as coefficient matrices with a higher percentage tend to be sparser.

Based on Equation 6, we can clearly see that for the extended set of monomials $\mathbf{x}$, we have $A\mathbf{x} = 0$ and $B_1\mathbf{x} = \lambda B_2\mathbf{x}$. This means, $\mathbf{x}$ is in the null space of the long matrix A. If $Z := null(A)$, then there exists a vector $\mathbf{y}$ such that $\mathbf{x} = Z\mathbf{y}$, therefore based on the lower part of Equation 6, we can write $B_1 Z\mathbf{y} = \lambda B_2 Z\mathbf{y}$. Based on the way we expand the system, we know that A is full-rank, therefore, both $B_1 Z$ and $B_2 Z\mathbf{y}$ are square matrices and System 7 is an eigenproblem.

3.2. Step 5: Computing the null space

The main computational cost of the proposed method lies in calculating the null space. If this step can be performed during the offline stage, it significantly enhances the method's efficiency. Unlike the inverse of a sparse matrix, sparsity can be kept for the the null space computations if we do not consider orthonormality condition. This fact is beneficial for symbolic computations. Moreover, as we will demonstrate, the null space can be computed using a hybrid offline-online approach.

Null space calculation has been used in the past to solve polynomial systems of equations in some applications [9, 10, 17], but it is not typically considered within the context of symbolic computation and minimal problems. One way to calculate the null space of A with size $k \times p$ is to compute the QR decomposition of A^t and then consider the last $p - k$ columns of Q [17]. However, performing a full QR decomposition in the offline stage can be costly. Essentially, we do not need the basis for the null space to be normalized and orthogonal. Also, doing so we are going to lose sparsity of the matrix and the resulting null matrix is going to be dense.

One of the key aspects of the extended multiplication matrix is that usually even if the coefficients of the polynomial system are dense, the matrix still has a considerable number of zeros and many rows with the same structure. Therefore, it would be more efficient to just use a form of Gaussian elimination in order to find the null space fully or partially in the offline stage. It would also be beneficial to sort the rows of the matrix and divide it into dense and sparse parts to better utilize a hybrid offline-online approach. A more detailed investigation is presented in the following subsection.

3.2.1 Partial calculation of null space

For calculating the null space of a matrix such as A, consider that we divide it into two parts, that is

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}, \tag{8}$$

then if we have $N_1 = null(A_1)$ and $N_2 = null(A_2)$, for calculating the null space of A, $x \in null(A)$ if and only $x \in null(A_1) \cap null(A_2)$, this means there exists a y such that $x = N_1 y$ and $A_2 x = A_2 N_1 y = 0$. Therefore, we need to find the null space of $A_2 N_1$ and calculate $x = N_1 y$ to get the members of the null space of A.

Considering that the main computational cost of symbolic null space calculations arises from the dense part of the matrix, this approach enables the development of a hybrid numerical-symbolic method for computing the null space. The Macaulay matrix can typically be divided into a large sparse part and a small dense part by sorting the rows based on the number of non-zero elements. Therefore, it is possible to compute the null space of the sparse part symbolically, and then incorporate the dense part and compute the final null space numerically as outlined in Algorithm 1.

Algorithm 1: Hybrid offline-online calculation of null space

Input: Symbolic matrix $A \in \mathbb{R}^{m \times n}$, size of rows needed for offline calculation $k < m$
Output: Matrix N, the null space of A

1 Sort A based on the number of non-zeros in descending order;
2 Initialize $S \leftarrow A(1:k,:)$;
3 Calculate the null space of symbolic matrix S using Algorithm 2 as N_S;
4 Substitute numericals inside A and N_S, denoted by A_{num} and $N_{S_{num}}$;
5 Let $N_2 \leftarrow null(A_{num}(k+1:end,:) * N_{S_{num}})$;
6 Calculate $N \leftarrow N_{S_{num}} * N_2$;
7 **return** N

Note that in the previous algorithm, when selecting a suitable value for k, we can begin with a single row and compute its null space. Additional rows can then be added iteratively, and symbolic calculations can be halted whenever the number of operations or computation time exceeds the allocated budget.

Another hybrid approach involves preconditioning the symbolic matrix to accelerate computations during the online stage. In [11], the authors suggested that by applying certain permutation matrices (found by performing randomized evaluation) to the symbolic matrix, the problem can be structured in a way that leads to faster computation. Their work focuses on applications in multibody mechanical systems, where the matrices involved are typically large and dense, making it impractical to compute them entirely during the offline stage.

3.2.2 Fraction-free Gaussian elimination

Gaussian elimination is a common method for transforming a matrix into row-echelon form. As we will see, It also plays an important role in computing the null space. A modified version of Gaussian elimination, known as the Bareiss algorithm, avoids divisions by multiplying the row to be eliminated by the pivot [24, 26]. Then, instead of dividing, we simply add the negative of the pivot row to the target row.

However, fraction-free Gaussian elimination is not well-suited for very large or dense matrices, as multiplying rows by the pivot can lead to numerical instability due to the growth of very large or very small elements. That said, in computer vision applications—where the systems are typically sparse and relatively small—fraction-free Gaussian elimination is a suitable method for computing the null space.

In the context of symbolic matrices, a dense matrix can result in extremely long symbolic expressions, making it difficult or even infeasible to compute the symbolic null space efficiently. Before applying fraction-free Gaussian elimination, it is advisable to sort the rows of the matrix based on the number of non-zero elements. This can help identify shorter pivot elements and, as previously mentioned, is essential for the hybrid symbolic-numerical approach to null space computation.

Using fraction-free Gaussian elimination, we can adopt two different strategies to compute the null space. These approaches will be briefly discussed in the following two subsections.

3.2.3 Back-substitution from reduced matrix

After transforming the matrix into row-echelon form R, the goal is to solve the system $R\mathbf{x} = 0$ to find the null space. Assuming that R is a tall, full-rank matrix of size $n \times m$, it will have $n - m$ free variables. We assign zero values to all but one of the free variables. The remaining free variable is set to one. Then, we solve the equation $R\mathbf{x} = 0$ to obtain the values of the bound variables (Step 22-29 of Algorithm 2). This process yields a basis vector for the null space of the matrix. The drawback of this approach is the need to calculate solutions of a linear system.

3.2.4 Facilitating null space calculation

One of the main challenges in symbolic null space computation is the generation of excessively long expressions, which can make symbolic calculations difficult or even infeasible to make.

One way to mitigate the impact of long expressions in symbolic null space computation is to introduce auxiliary symbols to represent them. This can be done after constructing the extended Macaulay matrix for a polynomial

Algorithm 2: Null Space Computation with Fraction-Free Elimination and Back-substitution

Input: Symbolic matrix $A \in \mathbb{R}^{m \times n}$
Output: Matrix N, the null space of A

Initialize $R \leftarrow A$, $r \leftarrow 1$, $pivots \leftarrow [\,]$;
for $c \leftarrow 1$ **to** n **do**
  Find the first nonzero entry in column c from row r to m;
  if *pivot found at* $pivot\_row$ **then**
    if $pivot\_row \neq r$ **then**
      Swap rows r and $pivot\_row$ in R;
    end
    Include c in $pivots$;
    for $i \leftarrow 1$ **to** m **do**
      if $i \neq r$ *and* $R[i, c] \neq 0$ **then**
        $R[i, :] \leftarrow R[i, :] \cdot R[r, c] - R[i, c] \cdot R[r, :]$;
      end
    end
    $r \leftarrow r + 1$;
  end
  if $r > m$ **then**
    break;
  end
end
Define free variables as $free\_vars \leftarrow \{1, \ldots, n\} \setminus pivots$;
Initialize N as a zero matrix of size $n \times size(free\_vars)$;
foreach $fv \in free\_vars$ **do**
  Set $x \leftarrow 0$, $x[fv] \leftarrow 1$;
  for $j \leftarrow |pivots|$ **to** 1 **do**
    $r \leftarrow j$, $bound \leftarrow pivots[j]$;
    $x[bound] \leftarrow -\frac{R[r, free\_vars] \cdot x[free\_vars]}{R[r, bound]}$;
  end
  Let x as the fv-th column of N;
end
return N

system by substituting each lengthy expression with a new variable. The symbolic null space is then computed using these simplified expressions. During the online stage, the auxiliary variables are replaced with their corresponding actual values, restoring the full solution.

The resulting expression for the null space typically requires simplification, which can be performed during its computation. However, by postponing this step until after the null space has been calculated, the simplification can be efficiently executed using parallelization techniques.

An unlikely ally in easing symbolic computation can, in fact, be a larger coefficient matrix. While it is common in the literature to minimize the size of the expanded coefficient matrix or elimination template, there is no clear evidence linking the size of the coefficient matrix to the accuracy of a method. For null space computation, having a larger coefficient matrix can be beneficial, as it typically results in a sparser matrix. This sparsity reduces the pressure on pivot elements and can lead to smaller algebraic expressions in the computed null space.

3.3. Step 6-7: Eigenvalue problem and extracting solutions

Depending on whether the null space was calculated offline or online, the eigenvalue problem in Equation 7 can be formulated either during the offline or online phase. After solving this system, it is common for spurious solutions to correspond to infinite eigenvalues, which are removed at this stage.

If the hidden variable λ is set to be equivalent to the variable x_i, then the eigenvalues represent the components x_i of the solution. The remaining components of the solution can be extracted from the eigenvectors. To do this, we identify monomials such that dividing one by another yields a specific component of the solution. This is equivalent to dividing the corresponding elements of the eigenvectors.

4. Camera geometry problems

In this section, we compare our proposed method with other state-of-art methods for solving camera-related problems in computer vision.

In the offline stage, the main computational cost of the SRBM method [1] lies in identifying an invertible submatrix within the coefficient matrix of the extended system. Additionally, matrix trimming may be required. Based on our experience, there is a significant risk that the method may fail during the offline stage, particularly when it becomes stuck trying to locate an invertible submatrix. In such cases, it is often necessary to attempt solving the problem using multiple parameter configurations, though in some instances, especially for overtly sparse coefficient matrices, it may not be possible to find feasible options, and the method may become stuck while attempting to remove rows. There is a small chance that, when using eigenvectors represented as monomial vectors, there may not be enough information to extract a particular component. This is due to the method selecting a smaller set of monomials for its sparse basis. In the proposed method, since we multiply the null matrix by the eigenvectors, we obtain a complete monomial vector.

In the proposed method, the primary computational cost lies in calculating the null space of a matrix, which can be efficiently performed partially or completely during the offline stage, especially when the matrix is sparse. If the null space computation is done online, both the null space computation in the proposed method and the matrix inversion

in SRBM have a theoretical complexity of $O(n^3)$. However, for sparse matrices, null space computation tends to be more numerically stable and better suited for symbolic manipulation. Another aspect that can influence the computation time during the online stage of the proposed problem is the process of substituting numerical values into the symbolic null space matrix. To accelerate this step, it would be beneficial to perform the substitutions in parallel.

For the better understanding of merits of our problem, we comapred our method with two other methods, the resultant-based SRBM and Groebner-based GAPS [22, 25] on twelve well-known camera geometry problems. The elimination matrix that we use for our proposed method is based on SRBM.

As can be seen from Table 1, the proposed method is consistently more accurate than SRBM, with the difference becoming more pronounced as the coefficient matrix becomes larger or sparser (for example, problem p9, unsynch and three-view geometry). Error measures are calculated based on the mean and median of Log10 of normalized equation residuals. Each problem is calculated for 1000 runs with random data points. In comparison to GAPS, the proposed method have the same level of accuracy in seven problems, is clearly better in three problem and performs worse in two problems.

Figure 2 demonstrates that the proposed method yields more stable results across different problems compared to SRBM. The histogram of the proposed method exhibits a more uniform structure, whereas the other methods show more sporadic behavior.

4.1. Time Analysis of SRBM vs. Proposed Method

The main advantage of the proposed method compared to SRBM lies in the offline stage. This is because SRBM needs to find a non-singular submatrix within the coefficient matrix, which it attempts to do by trimming the matrix in many different ways. However, in some cases, identifying such a submatrix is difficult or even impossible, causing the method to stall. Since many problems yield a sparse coefficient matrix, SRBM also requires the extended coefficient matrix to be made as small as possible (or denser) to increase the likelihood that trimming will produce a non-singular submatrix. This offline process is time-consuming, and in some cases, SRBM fails to find a suitable coefficient matrix altogether, preventing it from solving the system.

In contrast, our proposed method does not require finding an invertible matrix, trimming the coefficient matrix, or increasing its density, which makes it significantly faster during the offline stage. Based on our experience, during the online stage—when SRBM does manage to find a suitable coefficient matrix—the computation time of the proposed method and SRBM is quite similar.

5. Conclusion

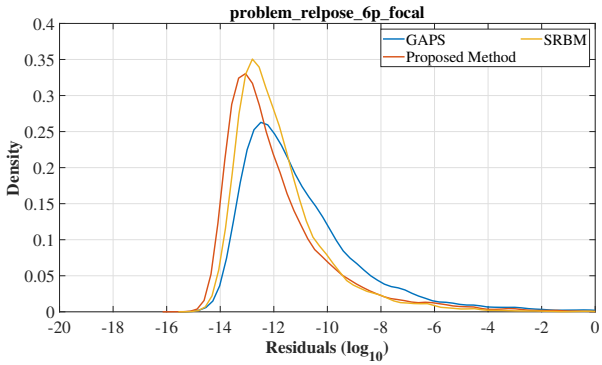
We provided a new way for calculating solutions of minimal problems using null space calculations. It was shown that a large part of computations can be done offline and it can prove a better or on-par accuracy in comparison to other state-of-art methods.

We observed that using the null space instead of matrix inverses yielded a method with superior accuracy in a majority of the problems. Therefore the proposed method can be considered an improved and more accurate version SRBM. In future work, it would be interesting to investigate how the Macaulay matrix can be constructed to improve null space computations. One approach is to use the coefficient matrix from methods such as GAPS or other methods with non-square Macaulay matrix, and then complete the remaining needed rows of the matrix by adding extra constraints related to the hidden variable. Our preliminary results indicate that this approach can yield solutions to the system. However, determining the selection criteria for achieving the best accuracy remains a challenge that needs to be addressed.

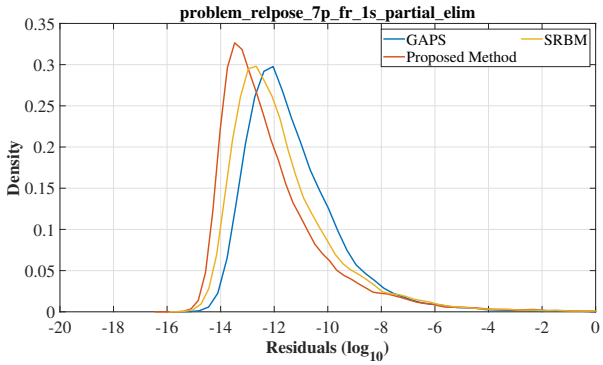
Acknowledgements. This work was supported by Academy of Finland (grant no. 355970).

Problem	E_P		E_{SRBM}		E_{GAPS}	
	mean	med	mean	med	mean	med
problem_9pt2radial [20]	**-8.7668**	**-9.5406**	-7.3922	-8.2950	-8.6463	-9.4926
problem_relpose_6p_focal [20]	**-11.8939**	**-12.4575**	-11.8400	-12.2319	-10.9547	-11.7503
problem_relpose_7p_fr_1s_partial_elim [23]	**-12.0681**	**-12.6467**	-11.6930	-12.3030	-11.1879	-11.7780
problem_opt_pnp_hesch [14]	-10.6501	-11.1942	-10.3908	-10.8760	**-11.3546**	**-11.7743**
problem_three-view_ geometry_rad._dist. [4]	**-8.2435**	**-10.3122**	-5.8022	-7.4497	-6.4892	-7.5263
problem_rel_pose_E+f_6pt [5]	-13.2676	-13.2294	-12.9122	-13.2018	**-13.6130**	**-13.8800**
problem_p4p_optimal_abs_pos [28]	-8.7149	-8.7965	-8.4963	-8.5769	**-10.7443**	**-11.1389**
problem_pc_relpose_5p_nulle_ne__simple [25]	-12.6551	**-12.9403**	**-12.6693**	-12.8629	-12.6217	-12.8542
problem_triangulation_satellite [29]	**–11.2089**	**–11.4307**	-10.7408	-10.9819	-11.1484	-11.3274
problem_unsync_relpose_diff_focal [8]	**-9.4846**	-10.6105	-7.6550	-8.7970	-9.4265	**-10.8491**
problem_8ptF_radial_1s [19]	**-12.5427**	**-13.2104**	-12.2373	-13.1171	-12.4209	-13.1464
problem_relpose_6p_focal_elim [21]	-10.9472	**-11.9029**	-9.8970	-10.6260	**-11.0329**	-11.7200

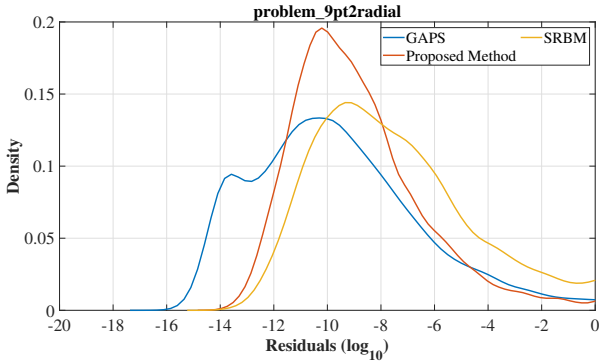
Table 1. Error comparison of the proposed method (P), SRBM and GAPS. Mean and median are computed from Log10 of normalized equation residuals.



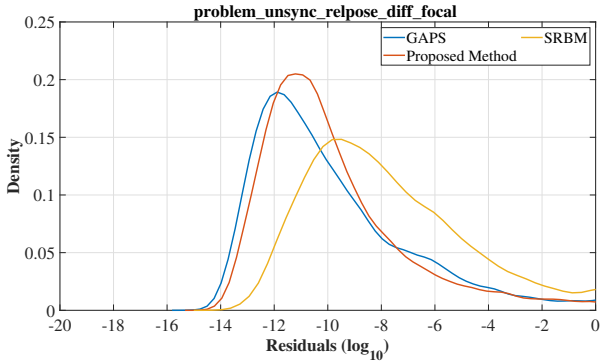
(a) Histogram of the problem Rel. pose 6p + f



(b) Histogram of the problem Rel. pose E+fλ 7pt.



(c) Histogram of the problem Rel. pose λ_1+F+λ_2 9pt



(d) Histogram of the problem Unsynch. Rel. pose

Figure 2. Histograms of Log_{10} residual errors of four problems. Errors are calculated for 1000 runs of each problem.

References

[1] Snehal Bhayani, Zuzana Kukelova, and Janne Heikkila. A sparse resultant based method for efficient minimal solvers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1770–1779, 2020. 1, 3, 6

[2] Snehal Bhayani, Zuzana Kukelova, and Janne Heikkilä. Computing stable resultant-based minimal solvers by hiding a variable. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6104–6111. IEEE, 2021. 2

[3] Snehal Bhayani, Janne Heikkilä, and Zuzana Kukelova. Sparse resultant-based minimal solvers in computer vision and their connection with the action matrix. *Journal of Mathematical Imaging and Vision*, 66(3):335–360, 2024. 2, 3

[4] José Henrique Brito, Roland Angst, Kevin Köser, Christopher Zach, Pedro Branco, Manuel Joao Ferreira, and Marc Pollefeys. Unknown radial distortion centers in multiple view geometry problems. In *Asian Conference on Computer Vision*, pages 136–149. Springer, 2012. 8

[5] Martin Bujnak, Zuzana Kukelova, and Tomas Pajdla. 3d reconstruction from image collections with a single known focal length. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1803–1810. IEEE, 2009. 8

[6] Martin Byröd, Klas Josephson, and Kalle Åström. Fast and stable polynomial equation solving and its application to computer vision. *International Journal of Computer Vision*, 84(3):237–256, 2009. 1

[7] David Cox, John Little, Donal O'shea, and Moss Sweedler. *Ideals, varieties, and algorithms*. Springer, 1997. 2

[8] Yaqing Ding, Jian Yang, Jean Ponce, and Hui Kong. Minimal solutions to relative pose estimation from two views sharing a common direction with unknown focal length. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7045–7053, 2020. 8

[9] Philippe Dreesen and Bart De Moor. Polynomial optimization problems are eigenvalue problems. In *Model-Based Control: Bridging Rigorous Theory and Advanced Technology*, pages 49–68, 2009. 4

[10] Philippe Dreesen, Kim Batselier, and Bart De Moor. Back to the roots: Polynomial system solving, linear algebra, systems theory. *IFAC Proceedings Volumes*, 45(16):1203–1208, 2012. 4

[11] Mark Giesbrecht and Nam Pham. A symbolic approach to compute a null-space basis in the projection method. In *Computer Mathematics: 9th Asian Symposium (ASCM2009), Fukuoka, December 2009, 10th Asian Symposium (ASCM2012), Beijing, October 2012, Contributed Papers and Invited Talks*, pages 243–259. Springer, 2014. 5

[12] Banglei Guan, Ji Zhao, Daniel Barath, and Friedrich Fraundorfer. Minimal solvers for relative pose estimation of multi-camera systems using affine correspondences. *International Journal of Computer Vision*, 131(1):324–345, 2023. 1

[13] Janne Heikkila. Using sparse elimination for solving minimal problems in computer vision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 76–84, 2017. 1, 2

[14] Joel A Hesch and Stergios I Roumeliotis. A direct least-squares (dls) method for pnp. In *2011 International Conference on Computer Vision*, pages 383–390. IEEE, 2011. 8

[15] Petr Hruby, Timothy Duff, Anton Leykin, and Tomas Pajdla. Learning to solve hard minimal problems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5532–5542, 2022. 1

[16] Petr Hruby, Shaohui Liu, Rémi Pautrat, Marc Pollefeys, and Daniel Barath. Handbook on leveraging lines for two-view relative pose estimation. In *2024 International Conference on 3D Vision (3DV)*, pages 376–386. IEEE, 2024. 1

[17] Gudbjörn Jónsson and Stephen Vavasis. Accurate solution of polynomial equations using macaulay resultant matrices. *Mathematics of computation*, 74(249):221–262, 2005. 2, 4

[18] Aleksey Kondratyev, Hans J Stetter, and Franz Winkler. Numerical computation of gröbner bases. *Proceedings of CASC2004 (Computer Algebra in Scientific Computing)*, pages 295–306, 2004. 2

[19] Yubin Kuang, Jan E Solem, Fredrik Kahl, and Kalle Astrom. Minimal solvers for relative pose with a single unknown radial distortion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 33–40, 2014. 8

[20] Zuzana Kukelova, Martin Bujnak, and Tomas Pajdla. Automatic generator of minimal problem solvers. In *European Conference on Computer Vision*, pages 302–315. Springer, 2008. 8

[21] Zuzana Kukelova, Martin Bujnak, and Tomas Pajdla. Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems. In *BMVC*, page 2008, 2008. 8

[22] Viktor Larsson, Kalle Astrom, and Magnus Oskarsson. Efficient solvers for minimal problems by syzygy-based reduction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 820–829, 2017. 2, 7

[23] Viktor Larsson, Magnus Oskarsson, Kalle Astrom, Alge Wallis, Zuzana Kukelova, and Tomas Pajdla. Beyond grobner bases: Basis selection for minimal solvers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3945–3954, 2018. 8

[24] Hong R Lee and B David Saunders. Fraction free gaussian elimination for sparse matrices. *Journal of symbolic computation*, 19(5):393–402, 1995. 5

[25] Bo Li and Viktor Larsson. Gaps: Generator for automatic polynomial solvers. *arXiv preprint arXiv:2004.11765*, 2020. 2, 7, 8

[26] George C Nakos, Peter R Turner, and Robert M Williams. Fraction-free algorithms for linear and polynomial equations. *ACM SIGSAM Bulletin*, 31(3):11–19, 1997. 5

[27] Henrik Stewénius. *Gröbner basis methods for minimal problems in computer vision*. PhD thesis, Citeseer, 2005. 1

[28] Linus Svärm, Olof Enqvist, Fredrik Kahl, and Magnus Oskarsson. City-scale localization for cameras with known vertical direction. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1455–1461, 2016. 8

[29] Enliang Zheng, Ke Wang, Enrique Dunn, and Jan-Michael Frahm. Minimal solvers for 3d geometry from satellite imagery. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 738–746, 2015. 8