Efficient Solvers for Minimal Problems by Syzygy-based Reduction

Viktor Larsson  Kalle Åström  Magnus Oskarsson
Centre for Mathematical Sciences
Lund University
{viktorl,kalle,magnuso}@maths.lth.se

Abstract

In this paper we study the problem of automatically generating polynomial solvers for minimal problems. The main contribution is a new method for finding small elimination templates by making use of the syzygies (i.e. the polynomial relations) that exist between the original equations. Using these syzygies we can essentially parameterize the set of possible elimination templates.

We evaluate our method on a wide variety of problems from geometric computer vision and show improvement compared to both handcrafted and automatically generated solvers. Furthermore we apply our method on two previously unsolved relative orientation problems.

1. Introduction

One of the success stories of computer vision is using robust estimation schemes such as RANSAC [13] in multiple view geometry estimation. With a hypothesis and test framework, one can efficiently handle large amounts of outliers in the measured data. A key element in such a framework is the ability to model the problem using a small or minimal subset of data points – a so-called minimal problem. A classic example is the 5-point algorithm for estimating the relative pose between two cameras, given only image point measurements, [25, 12, 41, 47]. The underlying problems in multiple view geometry naturally lead to systems of polynomial equations in several variables. In order to devise tractable algorithms, robust and fast solvers of polynomial equations are needed. The predominant way to solve minimal problems in computer vision is using methods based on Gröbner bases. The reason is that this often leads to fast and numerically stable algorithms. These methods were popularized in computer vision by Stewénius [46]. The earliest examples were methods that were very much handcrafted [24, 48], but since then much effort has been put into making the process of constructing the solvers more automatic. One of the main challenges is ways of automatically constructing the so-called elimination template, which is the main focus of this work.

Our contributions in this paper are:
(i) A non-iterative method for automatically finding the monomial expansion of the initial system of equations.
(ii) A non-iterative reduction step, that often gives a much more compact representation of the expanding set.
(iii) An efficient implementation of these ideas which produces stand-alone code for solving arbitrary instances of the problem.
(iv) Solvers for two previously unsolved minimal cases, based on our developed system.

1.1. Related work

The most closely related work is by Kukelova et al. [33], where the authors presented an automatic method for generating polynomial solvers. The automatic generator allows the user to specify a set of polynomial equations and then automatically generates stand-alone code for solving arbitrary problem instances. This automatic generator has been widely adopted in the computer vision community and it has been used to solve several problems in geometric computer vision (see e.g. Table 1 and the references therein.)

The solvers generated using [33] are built on the action matrix method that reduces the polynomial equation system to an eigenvalue problem. Their automatic generator works by first computing a Gröbner basis in Macaulay2 [15] for a random problem instance. The Gröbner basis computations and the Gaussian eliminations are performed in some prime field \( \mathbb{Z}_p \) to avoid numerical problems. The main drawback of the approach in [33] is that as the number of variables and equations grows the iterative search process that alternates between expanding the equation system and performing Gaussian elimination. Once sufficiently many equations have been generated a pruning step is used to remove any unnecessary equations. The Gröbner basis computations and the Gaussian eliminations are performed in some prime field \( \mathbb{Z}_p \) to avoid numerical problems. The main drawback of the approach in [33] is that as the number of variables and equations grows the iterative search and pruning step can quickly become intractable. In this work we propose a new method for finding the elimination template in place of the iterative search used...
in [33]. We show on a large number of examples that our approach almost always produces smaller elimination templates and faster polynomial solvers.

While our focus has been on constructing smaller templates there has been a number of works that have addressed the problem of making solvers more numerically stable [9, 26] and faster [6, 32]. It is possible that these methods could be applied in conjunction with our method, but this is left as further work.

2. Background

In this section we remind the reader of some of the basic facts and definitions from algebraic geometry that we will use throughout the paper. For a more thorough introduction see [10].

Let \( X = (x_1, \ldots, x_n) \) be a number of variables, and \( \mathbb{K} \) be some field. Then the set of all multivariate polynomials (with \( n \) variables) over \( \mathbb{K} \) is denoted \( \mathbb{K}[X] \), and this set with their natural operations forms a ring. In this paper we will only consider the case when \( \mathbb{K} = \mathbb{C} \) or \( \mathbb{K} = \mathbb{Z}_p \) for some prime number \( p \). For a set of polynomials \( F = \{ f_i \}_{i=0}^m \) the set of shared zeros, i.e.

\[
\mathcal{V}(F) = \{ x \in \mathbb{K}^n \mid f_i(x) = 0, \ i = 1, 2, \ldots, m \}
\]  

(1)

is called an affine variety and the set of all polynomial combinations of the elements in \( F \), i.e.

\[
I(F) = \{ p \in \mathbb{K}[X] \mid p = \sum h_i f_i, \ h_i \in \mathbb{K}[X] \},
\]  

(2)

forms an ideal in the polynomial ring \( \mathbb{K}[X] \). When it is clear from the context which polynomials are meant we will omit \( F \) and simply write \( \mathcal{V} \) and \( I \).

Similar to the univariate case there exists a division algorithm for multivariate polynomials. Unfortunately the remainder depends on the order in which the polynomials are listed. Fortunately for any ideal \( I \) there exist special sets of generators \( G = \{ g_i \}_{i=0}^l \) called Gröbner bases, such that the remainder after division with \( G \) is uniquely defined, regardless of how the individual \( g_i \) are listed. This allows us to define the normal form of a polynomial \( p \in \mathbb{K}[X] \) with respect to \( G \) as the unique remainder after division with \( G \). This is denoted \( p^G \). Note that \( p \in I \) if and only if \( p^G = 0 \).\footnote{For simplicity we take \( \alpha \) as a monomial here but the theory holds for a general polynomial as well.}

For a Gröbner basis \( G \) the normal set is the set of all monomials not divisible by any element in \( G \). It is easy to see that the normal form for any polynomial lies in the linear span of the normal set.

Another object of interest is the quotient space \( \mathbb{K}[X]/I \), which is the set of equivalence classes over \( I \) (i.e. two elements are equivalent if their difference lies in \( I \)). If an affine variety \( \mathcal{V} \) is zero dimensional (i.e. there are finitely many solutions) then the corresponding quotient space \( \mathbb{K}[X]/I \) will be a finite dimensional vector space. For any Gröbner basis \( G \) of \( I \) we have that the normal set forms a vector space basis of the quotient space \( \mathbb{K}[X]/I \).

2.1. The action matrix method

Next we give a brief overview of the action matrix method for solving polynomial systems. The main idea is to reduce the problem to an eigenvalue problem for which there exist good numerical methods. For a more thorough review of the action matrix method and how it has been applied in computer vision we recommend [37], [33] and [9].

Consider the operator \( T_\alpha : \mathbb{K}[X]/I \to \mathbb{K}[X]/I \) which multiplies a polynomial with the fix monomial\footnote{For simplicity we take \( \alpha \) as a monomial here but the theory holds for a general polynomial as well.} \( \alpha \in \mathbb{K}[X] \), i.e.

\[
T_\alpha [p(x)] = [\alpha(x)p(x)], \quad p \in \mathbb{K}[X].
\]  

(3)

The operator \( T_\alpha \) is a linear map and thus if we choose a (linear) basis \( b \) for the quotient space we can express the operator with a matrix \( M \), i.e.

\[
[\alpha b] = \left[ \sum_j m_{ij} b_j \right] \iff [\alpha b] = [Mb].
\]  

(4)

For each \( x \in \mathcal{V} \) we must have \( Mb(x) = \alpha(x)b(x) \). Thus if we evaluate \( \alpha \) and \( b \) at the solutions we get eigenvalues and eigenvectors for the matrix \( M \). So if we can find the action matrix \( M \) we can recover the solutions by solving an eigenvalue problem, and hence we have reduced the solving of the system of polynomial equations to a linear algebra problem.

The monomials \( r_i = \alpha b_i \) are called the reducible monomials. If a Gröbner basis \( G \) is known and \( b \) is chosen as the normal set we can recover the action matrix by reducing the reducible monomials with the Gröbner basis, i.e. \( \tau^G_i = \sum_j m_{ij} b_j \). Due to roundoff error it is usually not possible to directly compute a Gröbner basis for a polynomial system corresponding to a real problem instance. Instead an alternative approach is taken to recover the action matrix. The idea is based on the observation that each

\[
r_i - \sum_j m_{ij} b_j \in I
\]  

(5)

and thus there exist some polynomials \( h_{ij} \in \mathbb{K}[X] \) such that

\[
r_i - \sum_j m_{ij} b_j = \sum_j h_{ij} f_j.
\]  

(6)

To find the action matrix \( M \) the original set of equations \( \{ f_j(x) = 0 \}_{j=0}^m \) is expanded by adding new equations formed by multiplying each \( f_j \) by some monomials. If we have multiplied by sufficiently many monomials (i.e. all monomials in the unknown \( h_{ij} \)) we can express each polynomial \( f_j \) linearly in the expanded set of equations.

To do this in practice (see e.g. [26] for details) we write the expanded set of equations as \( CX = 0 \), where the matrix \( C \) is called the elimination template and \( X \) is a vector of all the monomials occurring in the equations. By reordering
the monomials we can rewrite this as
\[ CX = \begin{bmatrix} C_E & C_R & C_B \end{bmatrix} \begin{bmatrix} x_E \\ x_R \\ x_B \end{bmatrix} = 0, \] (7)
where we have grouped the monomials into excessive monomials \( x_E \), reducible monomials \( x_R \) and basis monomials \( x_B \). The excessive monomials are simply the monomials which are neither reducible nor basis monomials. Now since we know \( x_R = M x_B \) we simply perform Gaussian elimination on (7) and get the following form on the lower part of (7)
\[ \begin{bmatrix} 0 & I & -M \end{bmatrix} \begin{bmatrix} x_E \\ x_R \\ x_B \end{bmatrix} = 0 \] (8)
from which we can extract the action matrix \( M \).2

3. Finding elimination templates in \( \mathbb{Z}_p[X] \)

Now we will present our proposed approach for finding elimination templates for a given problem. Similarly to [33] we start by generating instances of the problem where the equations have coefficients in some prime field \( \mathbb{Z}_p \). Due to the exact arithmetic available in these fields we can easily compute a Gröbner basis \( G = \{ g_k \}_{k=0}^n \) for the ideal. From the Gröbner basis we find a linear basis \( b \) for the quotient space by forming the normal set. As described in Section 2.1 we can then find the action matrix by reducing each of the reducible monomials,
\[ r^G_i = \sum_j m_{ij} b_j. \] (9)
Note that this only gives the action matrix for this particular integer instance, which we are not particularly interested in.

However, by keeping track of how the Gröbner basis elements are formed we can express each \( g_k \) in the generators \( f_j \), i.e.3
\[ g_k = \sum_k c_{kj} f_j \] (10)
where the coefficients \( c_{kj} \in \mathbb{Z}_p[X] \) are polynomials. Since each \( r_i = \sum m_{ij} b_j \in I \) we can then find polynomials \( h_{ij} \in \mathbb{Z}_p[X] \) such that
\[ r_i - \sum_j m_{ij} b_j = \sum_k a_{ik} g_k = \sum_k a_{ik} (\sum_j c_{kj} f_j) = \sum_j h_{ij} f_j \] (11)
by dividing by \( \{ g_k \} \) and substituting each \( g_k \) with (10). While the polynomials \( h_{ij} \) are specific to this instance, they will typically have the same structure and only the coefficients (i.e. numbers) will be different for different problem instances. Thus to form our elimination template we expand our equation set by multiplying each equation \( f_j(x) = 0 \) by the monomials in \( h_{ij} \) for each \( i = 1, 2, \ldots, n_r \).

Typically the elimination templates found using this method will be quite large. In the next section we will show how we can find simpler polynomials \( h_{ij} \) that give more compact elimination templates.

3.1. Reducing the expansion

In the previous section we showed how to obtain polynomials \( h_{ij} = (h_{i1}, \ldots, h_{in}) \in \mathbb{Z}_p[X]^n \) such that we could represent the polynomials needed for forming the action matrix, i.e.
\[ p_i = r_i - \sum_j m_{ij} b_j = \sum_j h_{ij} f_j. \] (12)
These representations of \( p_i \) in \( \{ f_j \} \) are however not unique, since for any \( s = (s_1, \ldots, s_n) \in \mathbb{Z}_p[X]^n \) which satisfies
\[ \sum_j s_j f_j = 0, \] (13)
we also have
\[ p_i = \sum_j (h_{ij} + s_j) f_j. \] (14)
Consider the set of all \( s \in \mathbb{Z}_p[X]^n \) which satisfy this, i.e.
\[ \mathcal{M} = \{ s \in \mathbb{Z}_p[X]^n \mid \sum_j s_j f_j = 0 \}. \] (15)
This set forms a sub-module in \( \mathbb{Z}_p[X]^n \) and is called the first syzygy module of \( (f_1, \ldots, f_n) \) [11]. It captures all polynomial relations between the original equations \( \{ f_j(x) = 0 \}_{i=0}^m \) and it is clear that any representation of \( p_i \) in \( \{ f_j \} \) can be written as \( h_i + s \) for some element \( s \in \mathcal{M} \).

Finding the \( s \in \mathcal{M} \) which yields that smallest template or the best numerics is a difficult problem. Instead we now present a simple heuristic that usually works well in practice. We start by computing a Gröbner basis \( G_{\mathcal{M}} \) for the module \( \mathcal{M} \). This is done on the prime field problem instance using Macaulay2. The Gröbner basis depends on the monomial order chosen for \( \mathcal{M} \) and in this work we have used the Term-Over-Position-GRevLex which is the default order for modules in Macaulay2. Next to find a simpler representation of each \( p_i \) we compute the normal form w.r.t. \( G_{\mathcal{M}} \) for each \( h_i \), i.e.
\[ \overline{h}_i = \overline{h}_i^{G_{\mathcal{M}}}. \] (16)
This can be thought of as removing as much as possible of \( \mathcal{M} \) from the representation. The following proposition shows that the new representations \( \overline{h}_i \) are minimal in the sense that the maximum degree of the monomials is minimized. Note that there can be multiple representations with minimal degree and this approach only finds one of them.

Proposition 1. If \( \mathcal{M} \) is defined as above, \( p = \sum_j h_{ij} f_j \) and \( G_{\mathcal{M}} \) is a Gröbner basis for \( \mathcal{M} \) with respect to TOP-GRevLex (or any other degree first monomial order), then

---

2Note that in practice some reducible monomials might be available from the basis monomials.
3In Macaulay2 this can be accomplished using ChangeMatrix.
Assume that there exist some $s \in \mathcal{M}$ such that

$$\max_i \deg \tilde{h}_i + s_i < \max_i \deg \bar{h}_i,$$

(18)

Then since the monomial order is degree first we must have $LM(\tilde{h} + s) < LM(\bar{h})$ which implies $LM(\tilde{h}) = LM(s)$. But then $\tilde{h}$ is divisible by an element in $\mathcal{M}$ which is a contradiction.

3.2. Implementation details

We have written an automatic generator in MATLAB, which uses the technique described above to find and reduce the elimination templates. The generator is similar to that of Kukelova et al. [33] in that it only requires the user to specify the problem equations and then generates stand-alone MATLAB code that can be used to solve arbitrary problem instances. The elimination template generation and reduction are performed in just a few lines of Macaulay2 [15]. The automatic generator allows the user to easily experiment with different parameters such as which action monomial to use and the monomial ordering for the ideal, which can greatly affect the size of the polynomial solver. The generator can also automatically identify and exploit any variable aligned symmetries as described in [35]. In the implementation we do not perform any refinement on the elimination templates (except for the reduction step described in Section 3.1). It is possible that by using template optimization techniques such as those described in [26, 39] the results could be further improved. We have made the code for generating solvers publicly available. For most of the problems that we have tried, the solver generation time is quite small. The median running time for all the problems described in Table 1 of our automatic generator (executed on a standard desktop computer) is 5.7s.

4. Evaluation of the reduction step

In this section we evaluate the reduction step proposed in Section 3.1. Note that while the reduction does not guarantee that the template will be smaller we will show that this is often the case in practice.

To perform the evaluation we applied the automatic generator to a wide variety of minimal problems from the computer vision literature. Table 1 shows both the original template sizes as reported by the authors and the resulting templates from our proposed automatic generator. We have marked the templates with the smallest number of elements in bold. It can be seen that in general the reduction step produces a smaller template. More interestingly is perhaps that the reduced template is often smaller than the template in the original paper. Note that many of the papers used the automatic generator from Kukelova et al. [33] (indicated with (*) in the table). For many of the tested problems we get a significant decrease in template size, and for some a very large decrease. For instance for the problem of estimating relative pose with a known rotation direction we go from a template of size 411 × 489, [45] to a template of size 40 × 57. If we assume that the time complexity of the solver is quadratic in the number of rows and linear in the number of columns this corresponds to a speed-up factor of 900. The problem contains a symmetry that was not used by the original authors and we have used the method in [35] to remove it. (Here the smaller template doesn’t contain all the variables in the basis for the quotient space, but the remaining variables can be extracted linearly from the initial equations.)
<table>
<thead>
<tr>
<th>Problem</th>
<th>Author</th>
<th>Original template size</th>
<th>Proposed generator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rel. pose 5pt</td>
<td>Siewenius et al. [47]</td>
<td>10 x 20</td>
<td>10 x 20 x 20</td>
</tr>
<tr>
<td>Rel. pose 6pt one-sided rad. dist.</td>
<td>Kuang et al. [30]</td>
<td>12 x 24</td>
<td>11 x 20 x 20</td>
</tr>
<tr>
<td>TDOA offset rank 2, 7,4 pts</td>
<td>Kuang et al. [28]</td>
<td>20 x 15</td>
<td>20 x 15 x 20</td>
</tr>
<tr>
<td>Rel. pose + one focal 6pt</td>
<td>Bujnak et al. [4] (*)</td>
<td>21 x 30</td>
<td>21 x 30 x 20</td>
</tr>
<tr>
<td>P3.5P + focal</td>
<td>Wu [52]</td>
<td>20 x 43</td>
<td>24 x 45 x 20</td>
</tr>
<tr>
<td>Rel. pose + const. focal 6pt</td>
<td>Kukelova et al. [33] (*)</td>
<td>31 x 46</td>
<td>31 x 50 x 20</td>
</tr>
<tr>
<td>Rel. pose + rad. dist. 8pt</td>
<td>Kuukela et al. [33] (*)</td>
<td>32 x 48</td>
<td>31 x 49 x 32</td>
</tr>
<tr>
<td>Rel. pose 6pt ones-sided rad. dist.</td>
<td>Kuang et al. [30]</td>
<td>48 x 70</td>
<td>34 x 60 x 34</td>
</tr>
<tr>
<td>TDOA offset rank 2, 5,6 pts</td>
<td>Kuang et al. [28]</td>
<td>105 x 83</td>
<td>105 x 83 x 40</td>
</tr>
<tr>
<td>Rolling shutter pose</td>
<td>Saurer et al. [44] (*)</td>
<td>48 x 56</td>
<td>48 x 55 x 47</td>
</tr>
<tr>
<td>Generalized P4P + scale</td>
<td>Ventura et al. [51] (*)</td>
<td>48 x 56</td>
<td>50 x 55 x 47</td>
</tr>
<tr>
<td>Stitching + const. focal + rad. dist. 3pt</td>
<td>Naroditsky et al. [39]</td>
<td>54 x 77</td>
<td>96 x 108 x 48</td>
</tr>
<tr>
<td>TDOA offset rank 3, 9,5 pts</td>
<td>Kuang et al. [28]</td>
<td>70 x 31</td>
<td>70 x 31 x 70</td>
</tr>
<tr>
<td>TDOA offset rank 3, 7,6 pts</td>
<td>Kuang et al. [28]</td>
<td>255 x 157</td>
<td>255 x 157 x 75</td>
</tr>
<tr>
<td>Generalized rel. pose 6pt</td>
<td>Stewenius et al. [48]</td>
<td>60 x 120†</td>
<td>135 x 164 x 99</td>
</tr>
<tr>
<td>Optimal PnP</td>
<td>Hesc et al. [21]</td>
<td>120 x 120</td>
<td>93 x 116 x 88</td>
</tr>
<tr>
<td>Triangulation from satellite im.</td>
<td>Zheng et al. [53] (*)</td>
<td>93 x 120</td>
<td>93 x 116 x 88</td>
</tr>
<tr>
<td>Optimal PnP (Casley)</td>
<td>Nakano [38] (*)</td>
<td>124 x 164</td>
<td>186 x 161 x 118</td>
</tr>
<tr>
<td>P4P + focal + rad. dist.</td>
<td>Bujnak et al. [5] (*)</td>
<td>136 x 152</td>
<td>140 x 144 x 140</td>
</tr>
<tr>
<td>Rel. pose + rad. dist. 6pt</td>
<td>Kukelova et al. [33] (*)</td>
<td>238 x 290</td>
<td>223 x 290 x 154</td>
</tr>
<tr>
<td>Rel. pose + 2 rad. dist. 9pt</td>
<td>Kukelova et al. [33] (*)</td>
<td>179 x 203</td>
<td>355 x 298 x 165</td>
</tr>
<tr>
<td>Rel. pose 7pt one-sided focal + rad. dist.</td>
<td>Kuang et al. [30]</td>
<td>200 x 231</td>
<td>249 x 214 x 185</td>
</tr>
<tr>
<td>Weak PnP</td>
<td>Larsson et al. [35]</td>
<td>234 x 276</td>
<td>568 x 498 x 189</td>
</tr>
<tr>
<td>Weak PnP (2x2 sym)</td>
<td>Larsson et al. [35]</td>
<td>104 x 90</td>
<td>83 x 90 x 49</td>
</tr>
<tr>
<td>Rolling shutter R6P</td>
<td>Albl et al. [2] (*)</td>
<td>196 x 216</td>
<td>222 x 230 x 204</td>
</tr>
<tr>
<td>Optimal pose w dir 4pt</td>
<td>Svarm et al. [49]</td>
<td>280 x 252</td>
<td>371 x 351 x 203</td>
</tr>
<tr>
<td>Rel. pose w dir. 3pt</td>
<td>Saurer et al. [45] (*)</td>
<td>411 x 489</td>
<td>287 x 324 x 210</td>
</tr>
<tr>
<td>Rel. pose w dir. 3pt (using sym.)</td>
<td>-</td>
<td>-</td>
<td>94 x 111 x 40</td>
</tr>
<tr>
<td>Abs. pose quivers</td>
<td>Kuang et al. [27]</td>
<td>372 x 386</td>
<td>420 x 406 x 217</td>
</tr>
<tr>
<td>L₂ 3 view triangulation (Relaxed)</td>
<td>Kukelova et al. [34] (*)</td>
<td>274 x 305</td>
<td>399 x 384 x 239</td>
</tr>
<tr>
<td>Rel. pose w angle 4pt</td>
<td>Li et al. [36] (*)</td>
<td>270 x 290</td>
<td>280 x 304 x 266</td>
</tr>
<tr>
<td>Refractive P5P</td>
<td>Haner et al. [16]</td>
<td>280 x 399</td>
<td>410 x 480 x 240</td>
</tr>
<tr>
<td>TDOA offset rank 3, 6,8 pts</td>
<td>Kuang et al. [28]</td>
<td>1359 x 754</td>
<td>1359 x 754 x 356</td>
</tr>
<tr>
<td>Optimal PnP</td>
<td>Zheng et al. [54] (*)</td>
<td>575 x 656</td>
<td>812 x 704 x 521</td>
</tr>
<tr>
<td>Optimal PnP (using sym.)</td>
<td>Zheng et al. [54] (*)</td>
<td>348 x 376</td>
<td>484 x 408 x 302</td>
</tr>
<tr>
<td>Optimal pose w dir 3pt</td>
<td>Svarm et al. [49]</td>
<td>1,260 x 1,278</td>
<td>918 x 726 x 544</td>
</tr>
<tr>
<td>Optimal PnP (quaternion)</td>
<td>Nakano [38] (*)</td>
<td>630 x 710</td>
<td>958 x 693 x 604</td>
</tr>
<tr>
<td>Refractive P6P + focal</td>
<td>Haner et al. [16]</td>
<td>648 x 917</td>
<td>2,196 x 1,913 x 636</td>
</tr>
<tr>
<td>Rel. pose + const. focal + rad. dist. 7pt</td>
<td>Jiang et al. [23]</td>
<td>886 x 1,011</td>
<td>1,393 x 1,237 x 581</td>
</tr>
<tr>
<td>Dual-Receiver TDOA 5pt</td>
<td>Burgess et al. [7]</td>
<td>2,625 x 2,352</td>
<td>850 x 1,167 x 455</td>
</tr>
<tr>
<td>Optimal PnP (using sym.)</td>
<td>Nakano [38] (*)</td>
<td>1,936 x 1,976</td>
<td>1,698 x 1,153 x 1,102</td>
</tr>
<tr>
<td>L₂ 3 view triangulation</td>
<td>Kukelova et al. [34] (*)</td>
<td>1,866 x 1,975</td>
<td>2,647 x 2,584 x 1,759</td>
</tr>
</tbody>
</table>

(*) Original template constructed using [33]. If several elimination templates are used, the largest of these templates is reported. †: The problem contains variable-aligned symmetries [3, 31, 35] that was automatically found and removed by our generator. ‡: The original template doesn’t generate the full Gröbner basis, and some additional operations on the template are performed.

Table 1. Comparison of elimination template sizes for some common minimal problems in computer vision. The template with the fewest elements, for each problem, is shown in bold.
or equivalently by eliminating λ

\[ \text{det}\left( \begin{bmatrix} \ell_{1k} \\ R_2^T \ell_{2k} \\ R_3^T \ell_{3k} \end{bmatrix} \right) = 0, \quad k = 1, 2 \]  

which gives us two quadratic equations in the elements of \( R_2 \) and \( R_3 \). Since there are only two parameters in the rotations we can use these two equations to solve for the rotations independently from the rest of the variables.

Next we note that the constraint in (23) is invariant to the scale of the rotation matrices. We exploit this by parameterizing the rotations by non-unit quaternions (i.e. the Cayley transform), \( q_2 = (1, \ s_2v^T)^T \), \( q_3 = (1, \ s_3v^T)^T \). This parameterization gives us two quartic equations in the unknowns \( s_2 \) and \( s_3 \). Fixing the first element of the quaternion introduces a degeneracy for any 180-degree rotation.

Using the proposed automatic generator we construct a solver for this system. The resulting template size is 12 \( \times \) 20 and the system has 8 solutions. The solutions include two false solutions \( s_2 = s_3 = \pm 1 \) that were introduced by the non-unit quaternion parameterization. These can easily be discarded, and from the true solutions we can recover the correct rotations by rescaling each quaternion to unit length.

When the rotations are known we can use them to recover the translations \( t_2 \) and \( t_3 \). Using the point correspondence we can parameterize the two translations using the depths as

\[ t_k = \lambda_k x_k - R_k X, \quad k = 2, 3. \]  

Since \( P_1 = [I \ 0] \) we can select the scale such that \( X = x_1 \). Inserting (24) into the line constraints gives linear equations in the unknowns \( \lambda_2 \) and \( \lambda_3 \), which allows us to solve for the translations. We evaluated the performance of the two-step solver on 10,000 random synthetic instances. Figure 2 shows the residuals for the rotation estimation and the distance from the recovered pose to the ground truth. In average solving for the rotations and finding all translations took less than one millisecond per instance. Figure 3 shows an example where we have used the minimal solver in a RANSAC framework to estimate the relative pose of three

6. Three views with known rotation axis

In order to test our framework on a novel case, we turn our attention to the problem of pose estimation in three views when the rotation axis is known. Known rotation axis can occur in practice when there is additional information from sensors such as accelerometers which are common in modern cell phones. The problem is minimal when we observe one point and two lines. The problem has previously been studied for two views, where the minimal case is three points [14, 40, 50], and for two generalized cameras, where the minimal case is four points [20, 50]. Let the cameras be chosen as

\[ P_1 = [I \ 0], \ P_2 = [R(\theta_2, v) \ t_2], \ P_3 = [R(\theta_3, v) \ t_3] \]  

where \( R(\theta, v) \) denotes rotation of \( \theta \) radians around the rotation axis \( v \). Since the rotation axis is assumed to be known we can w.l.o.g. assume that \( v = (0, 1, 0)^T \) by rotating the image coordinate systems. The two line constraints can be formulated as

\[ \text{rank}\left( \begin{bmatrix} P_1^T \ell_{k1} \\ P_2^T \ell_{k2} \\ P_3^T \ell_{k3} \end{bmatrix} \right) = 2, \quad k = 1, 2 \]  

where \( \ell_{ij} \) denotes line \( i \) observed in image \( j \). The rank constraint can be encoded as a polynomial constraint by requiring each of the 3 \( \times \) 3 submatrices to have zero determinant. Finally the single point correspondence gives us three sets of equations

\[ \lambda_k x_k = P_k X, \quad k = 1, 2, 3 \]  

or equivalently by eliminating \( \lambda_k \)

\[ x_k \times P_k X = 0, \quad k = 1, 2, 3 \]  

where \( x_k \) denotes the image point in image \( k \).

6.1. Building an efficient two-step solver

To build an efficient solver for this problem we start by noting that the top 3 \( \times \) 3 submatrix in (20) only contains the rotation matrices. The determinant constraints are

Figure 1. Kernel smoothed histograms of residual errors for 5,000 runs – of from left to right – image stitching with unknown focal length and radial distortion [8, 39], the optimal PnP-method of Hesch et al. [21] and the optimal PnP-method of Zheng et al. [54].
which also shows the template sizes and average runtimes for the solvers. Note that there is about a three orders of magnitude difference in runtime between the fastest and slowest solvers, which shows the need of finding a good parameterization.

6.2. Importance of good parameterization

In the previous section we developed an efficient solver by choosing a clever parameterization of the problem. Finding a good parameterization is often one of the main difficulties when building polynomial solvers for minimal problems. Knowing which parameterization will yield good solvers is non-trivial and usually a trial and error approach is taken. Automatic tools such as the generator proposed in this paper greatly speeds up this process and allows for much faster prototyping.

To illustrate the importance of good parameterization we show three alternative parameterizations for this problem.

1. First we fix the scale by setting $X = x_1$ and directly parameterize the two translations $t_2 = (t_{21}, t_{22}, t_{23})^T$ and $t_3 = (t_{31}, t_{32}, t_{33})^T$. The rotations are represented using unit quaternions, $q_2 = (s_{21}, s_{22}v^T)$ and $q_3 = (s_{31}, s_{32}v^T)^T$. This system has 24 solutions, however there exist two independent two-fold symmetries and by removing these we get the desired 6 solutions.

2. Next we remove the translations by parameterizing the depths $\lambda_k$ in the 2nd and 3rd image, i.e. $t_k = \lambda_k x_k - R_k x_1$. This reduces the number of unknowns to 6.

3. Finally we tried using unit-quaternions when solving for the rotations using only the line constraints. The parameterization avoids the two false solutions introduced by the non-unit quaternion parameterization. However there still remains a two-fold symmetry.

The different parameterizations are summarized in Table 2 which also shows the template sizes and average runtimes in this paper greatly speeds up this process and allows for much faster prototyping.

6.2. Importance of good parameterization

In the previous section we developed an efficient solver by choosing a clever parameterization of the problem. Finding a good parameterization is often one of the main difficulties when building polynomial solvers for minimal problems. Knowing which parameterization will yield good solvers is non-trivial and usually a trial and error approach is taken. Automatic tools such as the generator proposed in this paper greatly speeds up this process and allows for much faster prototyping.

To illustrate the importance of good parameterization we show three alternative parameterizations for this problem.

1. First we fix the scale by setting $X = x_1$ and directly parameterize the two translations $t_2 = (t_{21}, t_{22}, t_{23})^T$ and $t_3 = (t_{31}, t_{32}, t_{33})^T$. The rotations are represented using unit quaternions, $q_2 = (s_{21}, s_{22}v^T)$ and $q_3 = (s_{31}, s_{32}v^T)^T$. This system has 24 solutions, however there exist two independent two-fold symmetries and by removing these we get the desired 6 solutions.

2. Next we remove the translations by parameterizing the depths $\lambda_k$ in the 2nd and 3rd image, i.e. $t_k = \lambda_k x_k - R_k x_1$. This reduces the number of unknowns to 6.

3. Finally we tried using unit-quaternions when solving for the rotations using only the line constraints. The parameterization avoids the two false solutions introduced by the non-unit quaternion parameterization. However there still remains a two-fold symmetry.

The different parameterizations are summarized in Table 2 which also shows the template sizes and average runtimes for the solvers. Note that there is about a three orders of magnitude difference in runtime between the fastest and slowest solvers, which shows the need of finding a good parameterization.

7. Nine lines in three views

In order to test the boundaries of our method, we have looked at a challenging unsolved minimal case. The classical problem of minimally estimating the geometry of three projective views of lines is an inherently difficult problem, [19, p. 413] and probably also not well numerically conditioned. Whereas algorithms for projective reconstruction from points in three views have existed for a long time, [22, 43], there exists no practical method for the corresponding minimal problem using lines. The problem is minimal with either six points or nine lines. There are also algorithms for different minimal combinations of lines and points, cf. [42]. Linear algorithms have been developed for over-constrained solutions of at least 13 lines [17], and for combinations of lines and points [18], and there exist non-linear methods for the over-constrained cases of 10-12 lines [29]. We assume that we have three unknown uncalibrated cameras, viewing nine unknown lines in space. Each camera has eleven parameters, and each line has four degrees of freedom. In addition to this, we can only determine a solution up to a global projective coordinate system, with 15 parameters. Each viewed line in each camera gives two constraints on our parameters. Since $3 \cdot 11 + 4 \cdot 9 - 15 = 2 \cdot 3 \cdot 9$, this gives a minimal system. A major reason that the line case is much more difficult than the point case, is that the projective coordinate system can be efficiently parameterized with five points. With lines we use fewer lines, and we need to be very careful in order to avoid specialized situations, e.g. lines intersecting. We have experimented with a large number of parameterizations of our problem, and in the end we found that the following gave the most tractable solution. First of all we make coordinate changes in the images so that the first two lines in each image are represented by $\ell_1 = (1, 0, 0)^T$ and $\ell_2 = (0, 1, 0)^T$. This corresponds to the lines $x = 0$ and $y = 0$ respectively. We then make a projective coordinate change so that the first camera is given by $P_1 = [I|0]$. This fixes 11 of the 15 degrees of freedom, and also gives that the first 3D-line must lie on the plane $\Pi_1 = P_1^T\ell_1 = (1, 0, 0, 0)^T$ and the second line on the plane $\Pi_2 = P_1^T\ell_2 = (0, 1, 0, 0)^T$. This in turn fixes the first two lines up to two parameters each. We can now fix the final four degrees of freedom of our coordinate system by specifying two additional planes ($\Pi'_1$ and $\Pi'_2$) that the first two lines should lie on. (Assuming that a 3D-line is represented by two points $X$ and $X'$ this places two linear constraints on the coordinate change homography $H$, $(HX)' = \Pi'_1$ and $(HX') = \Pi'_2$ for each line). Choosing $\Pi'_1 = (0, 1, 0, -1)^T$ and $\Pi'_2 = (0, 0, 1, -1)^T$ gives...
that the two additional cameras can be written

$$ P_3 = \begin{bmatrix} x_1 & 1 & 0 & -1 \\ 0 & x_2 & x_3 & -x_3 \\ x_4 & x_5 & x_6 & x_7 \end{bmatrix}, \quad P_3 = \begin{bmatrix} x_8 & x_9 & 0 & -x_9 \\ 0 & x_{10} & 1 & -1 \\ x_{11} & x_{12} & x_{13} & x_{14} \end{bmatrix}. $$

(25)

where \((x_1, \ldots, x_{14})\) are unknown parameters. We can now for each image-line triplet, \((\ell_i, \ell_j, \ell_k), i = 3, \ldots, 9,\) construct the matrix \(M_i = [P_i^T \ell_i, P_i^T \ell_j, P_i^T \ell_k].\) If the three image lines are views of the same line, the corresponding planes should intersect, and hence rank \(M_i = 2,\) and all \(3 \times 3\) sub-determinants of \(M_i\) should vanish. This gives three linearly independent second-degree polynomial constraints on \((x_1, \ldots, x_{14})\) for each \(i\) and in total 21 equations in the 14 parameters.

### 7.1. Building a solver

We have used our automatic generator on the formulation in (25). This leads to 36 solution\(^4\), using nine lines, resulting in a template with size 20,273 x 14,281 without the reduction step. This problem has a large number of variables, and the resulting equations contain a large syzygy-set, and we have not been able to calculate this in Macaulay2 entirely. We have run a partial reduction step, based on taking all possible combinations of four equations, and this leads to a template of size 16,278 x 13,735. To evaluate our generated solver we ran the following test. We generated a large number of ground truth synthetic problems. We then ran our solver on the corresponding data, and compared the closest of our 36 solutions to the ground truth solution. The resulting histogram is shown in Figure 4. It also shows the difference to the ground truth solution after non-linear refinement of our solution. Our final elimination template is large, but very sparse. The complete solver has an average runtime of 17.8 s in Matlab on a standard desktop computer (Intel I7-3930K 64 GB ram).

### 8. Conclusions

In this paper we have presented a new method for finding elimination templates, using syzygy modules. The module encapsulates the ambiguity in representing the polynomials that are needed for constructing the action matrix. We have achieved state-of-the-art results by finding the normal form w.r.t. the module, but it is possible that a more advanced search over the syzygies would yield even better results, and this is an interesting venue for further work.

---

\(^4\)In [1] the ideal of the trifocal tensor was investigated. Here they show that the corresponding variety has degree 297. However it turns out that most of these solutions do not correspond to a valid three view camera geometry, and these degeneracies are not present in our parameterization.
References


