

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Minimal solvers with truncated normal forms

Oskar Krejčí

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.
May 2025

I. Personal and study details

Student's name: **Krejčí Oskar** Personal ID number: **483579**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Science**
Study program: **Open Informatics**
Specialisation: **Cyber Security**

II. Master's thesis details

Master's thesis title in English:

Minimal Solvers with Truncated Normal Forms

Master's thesis title in Czech:

Minimální solvery se zkrácenými normálními formami

Name and workplace of master's thesis supervisor:

doc. Ing. Tomáš Pajdla, Ph.D. Applied Algebra and Geometry, CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **15.02.2024**

Deadline for master's thesis submission: **23.05.2025**

Assignment valid until: **21.09.2025**

Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Vice-dean's signature on behalf of the Dean

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work.
The student must produce his thesis without the assistance of others, with the exception of provided consultations.
Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

I. Personal and study details

Student's name: **Krejčí Oskar** Personal ID number: **483579**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Computer Science**
Study program: **Open Informatics**
Specialisation: **Cyber Security**

II. Master's thesis details

Master's thesis title in English:

Minimal Solvers with Truncated Normal Forms

Master's thesis title in Czech:

Minimální solvery se zkrácenými normálními formami

Guidelines:

- 1) Review basics of algebraic geometry and methods for solving polynomial systems [3] including truncated normal forms [1]
- 2) Select examples of minimal problems from [4,5] and develop formulations for the method presented in [2].
- 3) Carry out experiments comparing the solving technique [2] with techniques [4,5] regarding the speed and numerical stability of the solvers.

Bibliography / sources:

- [1] S Telen. Solving Systems of Polynomial Equations. PhD Thesis. KU Leuven. 2020.
[2] M R Bender, S Telen. Yet another eigenvalue algorithm for solving polynomial systems. arXiv:2105.08472v2. 2022.
[3] D A Cox , J Little , D O'Shea. Ideals, Varieties, and Algorithms. Springer 2015.
[4] E Martyushev, J Vrablikova, T Pajdla. Optimizing Elimination Templates by Greedy Parameter Search. arXiv:2203.14901v1. 2022.
[5] E Martyushev, S Bhayani, T Pajdla. Automatic Solver Generator for Systems of Laurent Polynomial Equations. arXiv:2307.00320v1. 2023.

Acknowledgements

Hereby I would like to thank my supervisor doc. Ing. Tomáš Pajdla, Ph.D. and my family. Mr. doc. Ing. Tomáš Pajdla, Ph.D. has provided helpful advice and observations during consultations of the thesis. My family has been patient and supportive during the time.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses. In Prague, 23. May 2025

Abstract

This work is concerned with the theoretical background, implementation and testing of three minimal solvers that provide the algebraic backbone for fast, robust computation of solutions to systems of polynomial equations. First, the reader is familiarized with terms of the underlying theory of solving some systems and some illustrative examples. Next, I provide rundown of general approaches also with examples, following with some specific implementations provided for the purpose of this thesis. Finally, I test all three minimal solvers with regards to speed and numerical stability.

Keywords: Minimal solvers, Multivariable Polynomial algebra, Ideals and Varieties, Eigenvalues and Eigenvectors, (Truncated) Normal forms, Gröbner basis, Resultants, Numerical algorithms

Supervisor: doc. Ing. Tomáš Pajdla, Ph.D.

Abstrakt

Tato práce se zabývá teoretickým základem, implementací a testováním tří minimálních solverů, které implementují algebraický základ pro rychlý a robustní výpočet řešení soustav polynomiálních rovnic. Nejprve se čtenář seznámí se základními pojmy teorie řešení některých soustav a s některými ilustrativními příklady. Dále uvádím shrnutí obecných přístupů s příklady a následně některé konkrétní implementace použité pro účely této práce. Nakonec testuji všechny tři minimální řešení z hlediska rychlosti a numerické stability.

Klíčová slova: Minimální solvery, Polynomiální algebra s více proměnnými, Ideály a variety, Vlastní čísla a vektory, (Zkrácené) Normální formy, Gröbnerova báze, Resultanty, numerické algoritmy

Příklad názvu: Minimální solvery se zkrácenými normálními formami

Contents

1	Introduction	1
2	Related work	3
2.1	Stitching	4
2.2	Absolute pose	4
2.3	Rolling shutter pose	4
2.4	Absolute pose quivers	5
2.5	Unsynchronized relative pose	5
2.6	Optimal PnP (Hesch)	5
2.7	Optimal PnP (Cayley)	5
2.8	Optimal pose 2pt v2	6
3	Background	9
3.1	Algebraic geometry	9
3.1.1	Varieties	10
3.1.2	Ideals	13
3.2	Methods of solving systems of polynomial equations	15
3.2.1	Gröbner basis	15
3.2.2	Truncated normal forms	19
3.2.3	Macaulay's Resultants	21
3.3	Minimal solvers	23
4	Implementation	27
4.1	ZeroDimSolve (Maple)	27
4.2	EigenvalueSolver (Julia)	28
4.3	Elimination templates (Matlab)	30
4.4	Tests	31
5	Experiments	33
5.1	Numerical stability	33
5.2	Speed	34
6	Conclusion	37
	Bibliography	39

Figures

3.1 A nephroid curve shown in red, touching a blue circle, with radius 2, on the x-axis.	11
3.2 Three varieties in different colors. In red is the yz-plane, i.e. all points from the set $f(0; y; z) y, z \in \mathbb{R}$. Similarly, xy-plane is in blue and xz-plane in green.	12
3.3 The intersection (blue) and union (magenta) of varieties $V_1; V_2$ and V_3	13

Tables

2.1 Table with name of considered problems along with strings under which they can be found in Martyushev's GitHub repository, number of solutions and reference to further details.	7
5.1 Categorization of tasks into classes from [4] and size of the largest candidate set.	33
5.2 Total relative error of all solutions provided by comparison of two different solvers.	34
5.3 This table contains information about how long it took each solver to produce the solution, with the unit being milliseconds (ms).	35



Chapter 1

Introduction

This work is concerned with the solving of systems of polynomial equations. Such systems represent problems in kinematics, robotics, acoustics, geometry, chemistry, mathematical biology, and other areas. We only consider systems from geometric computer vision tasks that are described by polynomials containing terms only with nonnegative integer exponents and nonzero coefficients, so more general polynomials (Laurent polynomials) are not handled.

There are plenty of methods to solve such systems, with the most important being eigenvalue algorithms (also known as algebraic algorithms) and homotopy continuation (which is outside of the scope of this thesis). The eigenvalue algorithms consist of two main steps. The first is reduction to univariate polynomial using linear algebra operations and the second is then finding the numerical solutions of it.

I have been provided with 3 eigenvalue algorithms (implemented in Matlab, Maple, and Julia apiece) that handle the said systems differently. A common name for all of them is the so-called minimal solvers. Minimal solvers have the property that they require minimum input to yield a solution. A classical example is the five-point algorithm (P5P) in epipolar geometry that uses five point correspondences between two views to compute the essential matrix, which encodes relative rotation and translation. However, more geometric computer vision problems, such as estimating transformations (rotation matrix and translation) or different parameters in 3D reconstruction and motion estimation, are studied in greater detail.

This thesis is divided as follows. In Background, I give the basic mathematical terminology necessary for the reader to understand how minimal problems are solved. Then insights into the solvers are presented. In chapter Implementation, I explain how the formulations of the problems are transformed for the method in [4]. In Experiments I compare all the solvers with respect to speed and numerical stability.

Chapter 2

Related work

The mathematics behind minimal solvers with truncated normal forms (TNF) is covered in undergraduate text in mathematics on Ideals, Varieties, and Algorithms by David Cox, et al. and Mr. Simon Telen's dissertation, ([8] and [30] respectively). Truncated normal forms are originally defined in [31], definition 2.2. and further expanded on in [30].

To my knowledge, the first automatic generator (AG) was developed in [16]. The minimal solver implemented in Matlab by Evgeniy Martyushev, et al. is available at <https://github.com/martyushev/EliminationTemplates>, see [20] or [21]. Simon Telen with Matías R. Bender implemented the other minimal solver in the Julia programming language as part of [4].

TNF-derived minimal solvers have been successfully employed for geometric estimation problems in vision, such as absolute pose (P3P) and relative pose (five-point essential matrix) where the underlying polynomial systems are zero-dimensional [31]. By integrating TNF with sparse resultant constructions, one can derive compact minimal solvers that rival or outperform classic action-matrix methods in size and numerical stability [5]. In structure-from-motion pipelines, TNF solvers enable fast hypothesis generation within RANSAC frameworks, directly leveraging their small eigenproblems to produce all candidate poses from minimal point correspondences [22].

The TNF methodology extends beyond generic systems: when the ideal I has fewer solutions than its Bézout bound or special structure, TNF algorithms can adapt via modified resultant maps and refined projection subspaces to still recover all isolated roots. Recent work [22] presents algorithms for non-generic zero-dimensional ideals that reduce TNF computation complexity by exploiting system sparsity and using blockstructured Macaulay matrices. These advances broaden the applicability of TNF solvers to systems arising in kinematic loop closures, multiview triangulation, and polynomial eigenvalue problems in engineering.

Automated toolchains including implementations in SiGNAL, Macaulay2, and custom MATLAB toolboxes now incorporate TNF generation routines alongside Gröbnerbasis and resultant-based solvers, giving practitioners flexible choices based on problem scale and conditioning [7]. Ongoing research focuses on hybridization with homotopy continuation and sparse resultant

selection schemes to further shrink solver footprints and improve runtime, targeting real-time applications in AR/VR and autonomous navigation [5]. As symbolic-numeric methods advance, TNF-based minimal solvers stand poised to become a standard component in next-generation geometric estimation and algebraic-geometry software.

In [21], many of the computer vision problems are compactly presented in Table 1. I have arbitrarily chosen 8 of them to test all the minimal solvers with regards to speed and numerical stability, namely Stitching, Absolute pose (P4P + fr), Rolling shutter pose, Absolute pose quivers, Unsynchronized relative pose, Optimal PnP (Hesch), Optimal PnP (Cayley) and Optimal pose 2pt v2. We will further investigate each of the tasks briefly in order to provide some insight to the reader into the problem and the equations that describe them. At the end one finds the table with columns containing the name of each problem, a string that is used in the GitHub repository mentioned above, number of solutions to the problem and reference to material where it comes from.

2.1 Stitching

Image stitching task is one where we assume multiple (possibly overlapping) pictures taken from one camera, differently rotated for each one (in [7] also with unknown focal length and radial distortion). Furthermore, 3 point references are sufficient to find the camera position. A typical example is creating a panoramic picture with your phone camera, but taking multiple photos and then using them as inputs into this problem instead of using the feature in your phone.

2.2 Absolute pose

The problem of estimating the absolute pose (position and rotation) of a camera in the world coordinate system is one of if not the oldest and most important in computer vision [6]. It further divides into more scenarios, such as planar or non-planar scenes or (un-)calibrated camera. It is generally referred to as PnP problem (Perspective-n-Point), where n corresponds to how many 2D-3D correspondences are needed to provide a solution for a specific instance of this problem (P3P, P4P, P5P, P7P [13], ...).

2.3 Rolling shutter pose

A roller shutter camera is one which does not record a whole picture of a scene at one single instant at a time, but rather in scan lines, either vertically or horizontally. The pattern is the same as the one on the roll shutters on windows, hence the name.

In the rolling shutter pose task, researchers estimate the absolute pose of a shutter camera from a picture it took while moving. Movement, in addition

to distance of objects from the camera, are variables that determine how great the distortion is. However, the speed, with which a shutter camera records, is so fast that the assumption of constant speed and linear movement proved to be sufficient [25].

2.4 Absolute pose quivers

The problem of absolute pose quivers is very similar to PnP problem. However, here we employ lines or directions as well, so instead of having only n point-to-point correspondences we also admit line-to-line correspondences. Their combination is called a quiver, so an m -quiver would be a point-to-point correspondence together with line-to-line correspondences while these lines pass through that point. This is explained nicely in [15].

2.5 Unsynchronized relative pose

The unsynchronized relative pose problem turns around two cameras, two-view geometry and tracking a dynamic scene. There are two main reasons for them not to be synchronized - time shift and frame rate. It is not easy and practical put multiple cameras in sync and, moreover, one has much more freedom if it is not necessary. For the scope of this thesis, I assume two cameras with different (but constant relative) pose, tracking a dynamic scene and also allowing a time shift between them. The exact understanding is explained in [2].

2.6 Optimal PnP (Hesch)

Joel A. Hesch looked at the problem of estimating absolute camera pose in a different way. First, he stated it as an optimization task, specifically as constrained non-linear least-squares minimization task. Then, he set the criterion function to the least squares of measurement errors and proposed a method (called DLS - direct least squares) that solves it [11].

There are two key advantages over other methods. Firstly, it provides all solutions as minima of the cost function, so it works in the general case (i.e. when $n = 3$). Note that some methods ([14, 19, 26]) work when a unique solution is admissible (i.e. when $n = 6$) or need a workaround in edge case setups. Secondly, it requires no initialization as opposed to e.g. standard Gauss-Jordan iterative method.

2.7 Optimal PnP (Cayley)

The goal of absolute pose estimation is understood as obtaining the translation vector (t) in world coordinate system and the rotation matrix (R). The

rotation matrix comes from special orthogonal group in 3 dimensions, which means, mathematically speaking:

$$R \in SO(3)$$

$$SO(3) = \{R \in \mathbb{R}^{3 \times 3}; R^T R = I; \det(R) = 1\}$$

Generally, one can represent the rotation matrix by a quaternion $q = [a; b; c; d]^T$. Then, a generic rotation matrix (using Rodrigues' rotation formula [32] around general axis $(u_x; u_y; u_z)$)

$$R(\mathbf{u}) = \begin{bmatrix} \cos \theta + u_x^2(1 - \cos \theta) & u_x u_y(1 - \cos \theta) & u_z \sin \theta & u_x u_z(1 - \cos \theta) + u_y \sin \theta \\ u_y u_x(1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2(1 - \cos \theta) & u_y u_z(1 - \cos \theta) & u_x \sin \theta \\ u_z u_x(1 - \cos \theta) - u_y \sin \theta & u_z u_y(1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2(1 - \cos \theta) & u_z \sin \theta \\ -u_x \sin \theta & u_y \sin \theta & -u_z \sin \theta & \cos \theta + u_x^2(1 - \cos \theta) \end{bmatrix} \quad (2.1)$$

is transformed into

$$R(a; b; c; d) = \frac{1}{4} \begin{bmatrix} a^2 + b^2 - c^2 - d^2 & 2(bc - ad) & 2(bd + ac) & 2(cd - ab) \\ 2(bc + ad) & a^2 - b^2 + c^2 - d^2 & 2(cd - ab) & 2(bd - ac) \\ 2(bd - ac) & 2(cd + ab) & a^2 - b^2 - c^2 + d^2 & 2(bc - ad) \\ 2(cd - ab) & 2(bd + ac) & 2(bd - ac) & a^2 + b^2 + c^2 - d^2 \end{bmatrix} \quad (2.2)$$

Cayley parametrization uses a specific quaternion $q = [1; b; c; d]^T$ to represent the rotation matrix, hence (2.2) becomes:

$$R(1; b; c; d) = \frac{1}{4} \begin{bmatrix} 1 - (b^2 + c^2 + d^2) & 2(bc - ad) & 2(bd + ac) & 2(cd - ab) \\ 2(bc + ad) & 1 - (b^2 + d^2) & 2(cd - ab) & 2(bd - ac) \\ 2(bd - ac) & 2(cd + ab) & 1 - (b^2 - c^2) & 2(bc - ad) \\ 2(cd - ab) & 2(bd + ac) & 2(bd - ac) & 1 - (b^2 + c^2) \end{bmatrix} \quad (2.3)$$

This parametrization is useful for DLS, as demonstrated in [23]. For details with regards to how transformation between (2.1) and (2.2) is done, I refer the reader to [9].

2.8 Optimal pose 2pt v2

This again a task to find the absolute pose of a camera (R, t) from a scene. However, it is set up in a city where there are millions of 3D points in unifying the result. Many of them are so-called outliers. Which points count as outliers is usually determined according to what the reprojection error of that point is. There are many outliers in this scenario, so more effort is put into fast detection (rejection) of them rather than accuracy of the estimate. Specifically, this is called localization. One of algorithms to tackle this task is shown in [29], where the authors claim it to work reliably even in cases with more than 99% outliers.

Problem name	Problem string	Number of solutions	Details
Stitching	stitching	18	[7]
Absolute pose	p4p_fr	16	[6]
Rolling shutter pose	rollingshutter	8	[25]
Absolute pose quivers	pose_quiver	20	[15]
Unsynchronised relative pose	unsynch_relpose	16	[2]
Optimal PnP (Hesch)	opt_pnp_hesch	27	[11]
Optimal PnP (Cayley)	opt_pnp_nakanoC	40	[23]
Optimal pose 2pt v2	optpose2pt_v2	24	[29]

Table 2.1: Table with name of considered problems along with strings under which they can be found in Martyushev's GitHub repository, number of solutions and reference to further details.

Chapter 3

Background

In this chapter, I provide pieces of the underlying theory of solving systems of polynomial equations. First, I present some basic definitions and terms from the mathematics supporting the algebraic geometry, the building blocks of which are the ideals and varieties. Next, I present more or less detailed insight into 3 methods that not only are in the scope of this thesis, but also approach the solution of specified systems differently. Finally, few words are given to the concept of minimal solvers in general.

3.1 Algebraic geometry

The essence of polynomial systems is a polynomial, so let us define precisely what it is. A polynomial consists of monomials:

Definition 1. A monomial is the product

$$x = \prod_{i=1}^n x_i^{i_i} \tag{3.1}$$

where we have generally n variables x_i to some non-negative integer powers i_i . x is then the relaxed notation as x substitutes the n -tuple (i_1, \dots, i_n) that "identifies" the particular monomial.

Definition 2. A polynomial p is a finite linear sum with coefficients in some field F .

$$p = \sum_j c_j x^j; \quad c_j \in F \tag{3.2}$$

1. c_j is called the coefficient of the monomial x^j . If $c_j \neq 0$, then $c_j x^j$ is called a term of p .
2. The total degree of a nonzero polynomial is the maximum $|j|$ of its terms and the zero polynomial has the total degree undefined.

where x^j is a monomial as defined above, (3.1). I denote the set of all polynomials in variables x_1, \dots, x_n with coefficients from the field K with the standard notation $K[x_1, \dots, x_n]$ (from now on I will use the abbreviation K or explicitly write out the variables). I kindly refer the reader, unfamiliar with mathematical fields, to [1].

■ 3.1.1 Varieties

Varieties are geometric objects that depict the solution to a polynomial system, hence mathematically:

Definition 3 (Variety).

$$V(p_1; \dots; p_s) = \{x_i \in K \mid p_j(x_1, \dots, x_n) = 0 \text{ } \forall j\} \quad (3.3)$$

where $V(p_1; \dots; p_s)$ is the variety defined by s polynomials $p_1; \dots; p_s$. Varieties follow 2 important rules, namely that the intersection and the union of two varieties is again a variety:

Lemma 1. Given two varieties $A(p_1; \dots; p_s)$ and $B(q_1; \dots; q_k)$, their intersection and union are also varieties:

$$A \cap B = V(p_1; \dots; p_s; q_1; \dots; q_k); \quad (3.4)$$

$$A \cup B = V(p_i; q_j) \quad (3.5)$$

Next, I give two examples to demonstrate varieties. The first one is a curve in \mathbb{R}^2 called the nephroid ([3]), which is a specific type of more general epicycloid¹ with two cusps and you can find it in a daily situation - a nephroid appears at the bottom of a coffee mug when sun rays enter it and reflect on the inner side of the cylinder-like shape.

The nephroid is defined by the following implicit equation:

$$(x^2 + y^2 - 4a^2)^3 = 108a^4y^2 \quad (3.6)$$

or equivalently (parameterically):

$$\begin{aligned} x(\theta) &= a(3\cos(\theta) - \cos(3\theta)) \\ y(\theta) &= a(3\sin(\theta) + \sin(3\theta)) \end{aligned} \quad (3.7)$$

$0 \leq \theta < 2\pi$

with 'a' being a parameter that relates to the scale of the curve. If you set $a = 1$, then you obtain exactly the curve in picture 3.1. In terms of what has been set up above, we have:

- our field K set to \mathbb{R} , as the coefficients in (3.6) are 1; 1 and $4a$ on the left hand side and $108a^4$ on the right hand side.
- two variables - x and y .
- a variety defined by one polynomial - $V((x^2 + y^2 - 4a^2)^3 - 108a^4y^2)$. Note that we have to move terms to one side since one must have 0 on the other, see the definition 3. This, however, is a trivial transformation and always possible.

To illustrate Lemma 1, assume we have $K = \mathbb{R}$, polynomials from $\mathbb{R}[x; y; z]$ and three varieties $V_1 = V(x)$; $V_2 = V(y)$ and $V_3 = V(z)$, which represent yz -plane, xz -plane and xy -plane in 3D, respectively. They are in Figure 3.2.

Figure 3.1: A nephroid curve shown in red, touching a blue circle, with radius 2, on the x-axis.

In the next picture (Figure 3.3), you can see their union and intersection in magenta and blue color.

We obtain their intersection as follows:

$$\begin{aligned}
 V_1(x) \setminus V_2(y) \setminus V_3(z) &= \\
 (V_1(x) \setminus V_2(y)) \setminus V_3(z) &= \\
 V(x; y) \setminus V_3(z) &= \\
 V(x; y; z) &
 \end{aligned}
 \tag{3.8}$$

and the union:

$$\begin{aligned}
 V_1(x) [V_2(y) [V_3(z) &= \\
 (V_1(x) [V_2(y)) [V_3(z) &= \\
 V(xy) [V_3(z) &= \\
 V(xyz) &
 \end{aligned}
 \tag{3.9}$$

The intersection is understood as all points where all of the varieties vanish at the same time, so we group together all the polynomials and consider it new variety. In 3.8, we had 3 varieties, each in just one variable. First, we grouped V_1 and V_2 to get $V(x; y)$, which is the z-axis, and then we intersected this z-axis with the xy-plane. The resulting variety is $V(x; y; z)$ that vanish

¹<https://en.wikipedia.org/wiki/Epicycloid>

Figure 3.2: Three varieties in different colors. In red is the yz -plane, i.e. all points from the set $\{(0; y; z) \mid y, z \in \mathbb{R}\}$. Similarly, xy -plane is in blue and xz -plane in green.

only at the origin, as we have 3 easy equations $x = 0; y = 0$ and $z = 0$ - that have no other possible solution.

For the union, however, it suffices that at least one of them vanish, so we pair each polynomial from one variety with each one from the other to ensure this. In 3.9, we first unionize V_1 and V_2 , getting $V(xy)$. This variety vanishes everywhere, where $x = 0$ or $y = 0$, hence having yz -plane and xz -plane as its solution. Note that, at this point, the variable z can admit any value as it does not influence the solution to the equation $xy = 0$. The last step is to connect $V(xy)$ and V_3 . This is easy straightforward as each one contains only one polynomial, so we multiply xy with z to obtain $V(xyz)$. Equivalently, this is solution to the system of polynomial equations with only one equation - $xyz = 0$. The possibilities are that one, two or all three of the variables are 0. If only one variable is equal to 0, then we get the corresponding canonical plane. In the case when two are 0, we get the corresponding axis (e.g. $x = 0$ and $z = 0$ means we find ourselves at y -axis). Lastly, when all of the variables are equal to 0, we obtain the origin, common to all three of the original varieties.

Figure 3.3: The intersection (blue) and union (magenta) of varieties V_1, V_2 and V_3 .

3.1.2 Ideals

Ideals are algebraic objects connected to varieties. Generally, an ideal has to satisfy three conditions:

Definition 4 (Ideal). An ideal is a subset $I \subseteq K[x_1, \dots, x_n]$ such that:

1. $0 \in I$,
2. $p, r \in I \implies p + r \in I$,
3. $p \in I \wedge r \in K[x_1, \dots, x_n] \implies rp \in I$.

Next, five useful categories of ideals are used in the context of polynomials:

Definition 5. There are five important kinds of ideals in the context of polynomials:

1. Principal ideal $I_{\text{Principal}}$ is any one that is generated by only one element:

$$I_{\text{Principal}} = \langle p \rangle; \quad p \in K \quad (3.10)$$

2. Prime - A prime ideal I_{Prime} is one that suits:

$$(I_{\text{Prime}} \cap K \neq I_{\text{Prime}}) \implies (p \in I_{\text{Prime}} \implies r \in I_{\text{Prime}}): \quad (3.11)$$

3. Background

3. Primary - A primary ideal $I_{\text{Primary}} \subseteq K$ satisfies:

$$\{p, r\} \subseteq K \implies \{p, r\} \subseteq I_{\text{Primary}} \text{ or } \{p, r\} \subseteq I_{\text{Primary}}^m \quad (3.12)$$

4. Radical - A radical ideal I_{Radical} (denoted with the square root symbol \sqrt{I}) is the set

$$\sqrt{I} = \{p \in K \mid \exists n \in \mathbb{N} : p^n \in I\} \quad (3.13)$$

5. Maximal - An ideal I is called maximal if $I \subsetneq K$ and for another ideal $J \subseteq K$ we have

$$I \subsetneq J \implies J = K \quad (3.14)$$

Having a system

$$\begin{aligned} p_1 &= 0; \\ p_2 &= 0; \\ &\vdots \\ p_s &= 0; \end{aligned} \quad (3.15)$$

we have two ways of obtaining an ideal. The first is directly putting the left side of the system as a basis:

$$\langle p_1, \dots, p_s \rangle = \left\langle \sum_{i=1}^s r_i p_i \mid r_i \in K[x_1, \dots, x_n] \right\rangle \quad (3.16)$$

and the other is to take the variety first, then creating ideal out of it, defined as:

Definition 6 (Ideal of a variety) Let's have a variety $V \subseteq K$, then the set

$$I(V) = \{p \in K \mid p(x_1, \dots, x_n) = 0 \forall (x_1, \dots, x_n) \in V\} \quad (3.17)$$

is called the ideal of V .

Ideals 3.16 and 3.17 are related as

$$\langle p_1, \dots, p_s \rangle \subseteq I(V(p_1, \dots, p_s)); \quad (3.18)$$

meaning that $I(V(p_1, \dots, p_s))$ can contain polynomials which we are not able to write as element of the set from 3.16.

The system 3.15 is also called a basis of the ideal Equation 3.16 or generators of the ideal. There may be multiple basis to an ideal, but changing the basis does not change its variety:

$$\langle p_1, \dots, p_s \rangle = \langle r_1, \dots, r_t \rangle \implies V(p_1, \dots, p_s) = V(r_1, \dots, r_t) \quad (3.19)$$

One especially useful, called the Gröbner basis, is described below in 3.2.1. This idea may be familiar with vector span in linear algebra with the difference being the multiplication by polynomials instead of scalars.

3.2 Methods of solving systems of polynomial equations

In this section I provide a rundown of some methods that somehow provide solutions to polynomial systems described earlier. First, I familiarize the reader with Gröbner basis, as I am most familiar with it. Next, another approach called Truncated Normal Forms is discussed. That is the most recent one. Finally, the oldest one is described. The resultant method, in the modern sense, dates back to 18th century, to the work of Etienne Bézout. After reading this chapter, the reader should be able to spot what method is used if given an algorithm.

3.2.1 Gröbner basis

A Gröbner basis is a particular kind of generating set for an ideal in a polynomial ring over a field, providing a powerful algebraic tool for solving systems of polynomial equations. Given an ideal I in a polynomial ring $K[x_1, \dots, x_n]$, a Gröbner basis G for I is a set of polynomials with the same ideal as its span, but with special properties that make it particularly useful for polynomial division and solving algebraic systems.

The Buchberger's algorithm is the one that computes this basis and comes from the PhD thesis of Bruno Buchberger in 1965. The Gröbner basis itself is named after his adviser Wolfgang Gröbner. Sometimes, one may find the term standard basis, because the Japanese mathematician Heisuke Hironaka developed an analogy in 1964 ([2]). The algorithm is motivated by two problems that arise in case of dividing a polynomial by more than one divisors in more than one variable. Formally, we want to express given polynomial p as

$$p = q_1 p_1 + \dots + q_j p_j + r; \quad (3.20)$$

where $p_1, \dots, p_j; r \in K[x_1, \dots, x_n]$. The r is commonly called the remainder and q_1, \dots, q_j are referred to as the quotients. One can find the complete algorithm that achieves this in [2] in chapter 2, but the two problems that are encountered is that both the quotients as well as the remainder might change depending on how we order p_1, \dots, p_j . Before we introduce the Buchberger's algorithm, it makes sense to define or explain key concepts in advance as they facilitate the understanding.

Monomial ordering

The reader should be familiar with the polynomial long division algorithm, where both the dividend and the divisor are first reordered in decreasing power of x . This is easy as there is only one variable, so it suffices to compare two integers. Monomial orderings are necessary to clarify the relation between x^α and x^β , which reduces to comparing $\alpha \succeq \beta$ (see 3.1). Besides being able to order monomials, a monomial ordering is required to respect operations on polynomials that we use - sums and products. However, multiplication

3. Background

distributes over addition, so in this case, we only need to consider what happens in multiplication of a polynomial by a monomial. All of these properties are considered in the definition 7.

Definition 7 (Monomial ordering) The relation $>$ on the set of monomials $x \in \mathbb{Z}^n_0$ is a monomial ordering if:

1. $(x > x') \wedge (x' > x'') \implies x > x''$ (transitivity)
2. $(> \wedge \mathbb{Z}^n_0) \implies + > +$
3. Exactly one of $x > x'$, $x = x'$ and $x < x'$ is true.
4. $\forall x \in \mathbb{Z}^n_0, \exists x' \in \mathbb{Z}^n_0 : x' > x$.

The last point in the definition is referred to as a well-ordering. The second point encodes the multiplication of polynomial by a monomial. If it was not in the definition, ordering of terms could change after multiplication. In other words, one wants all the original exponent vectors to be shifted equally after the multiplication. Next, it is noteworthy to define 3 commonly used monomial orderings that suit the definition 7:

1. **lexicographic ordering ($>_{\text{lex}}$):** Assume that $x \in \mathbb{Z}^n_0$. Then $x >_{\text{lex}} x'$ if the leftmost nonzero entry of $x - x'$ is greater than 0. Quick example:

$$\begin{aligned} x^{(2;9;7;0)} &<_{\text{lex}} x^{(4;6;0;1)}, \quad (4;6;0;1) - (2;9;7;0) = \\ &= (2; -3; -7; 1), \quad w^4x^6z >_{\text{lex}} w^2x^9y^7 \end{aligned} \quad (3.21)$$

2. **graded lexicographic ordering ($>_{\text{grlex}}$):** Assume that $x \in \mathbb{Z}^n_0$. Then $x >_{\text{grlex}} x'$ if

$$\sum_{i=1}^n x_i = \sum_{i=1}^n x'_i \implies \sum_{i=1}^n (x_i - x'_i) >_{\text{lex}} 0 \quad (3.22)$$

Graded lexicographic order prioritizes the total degree of monomials (i.e. the sum of the exponents) and decides lexicographically in case the total degrees equal.

3. **graded reverse lexicographic ordering ($>_{\text{grevlex}}$):** Assume that $x \in \mathbb{Z}^n_0$. Then $x >_{\text{grevlex}} x'$ if

$$\sum_{i=1}^n x_i = \sum_{i=1}^n x'_i \implies \sum_{i=1}^n (x_i - x'_i) <_{\text{lex}} 0 \quad (3.23)$$

nonzero entry of $x - x'$ is negative

Again, here the total degree is considered first, and if two monomials have it equal, then we go from the least important variable to the most

important one, meaning

$$x_1 >_{\text{grevlex}} x_2 >_{\text{grevlex}} \dots >_{\text{grevlex}} x_n \quad (3.24)$$

$$(1; 0; \dots; 0) >_{\text{grevlex}} (0; 1; \dots; 0) >_{\text{grevlex}} \dots >_{\text{grevlex}} (0; \dots; 0; 1);$$

and where the first variable with different power in both monomials is encountered, we declare the one with smaller power as greater than the other.

For each of the monomial orderings, there are $n!$ ways of ordering the variables, so people usually consider $x > y > z$, but $y > z > x$ is equally valid. After we fix the monomial ordering, we can talk about more necessary terms:

Definition 8. Let $p = \sum c_i x^i \in \mathbb{C}[x]$ with monomial ordering $>$

- Multidegree of p :

$$\text{mdeg}(p) = \max_j \{j \text{ terms of } p\} \quad (3.25)$$

- Leading coefficient of p : $\text{LC}(p) = c_{\text{mdeg}(p)}$
- Leading monomial of p : $\text{LM}(p) = x^{\text{mdeg}(p)}$
- Leading term of p : $\text{LT}(p) = \text{LC}(p) \text{LM}(p)$

To summarize this section, let us identify the terms in two different monomial orderings on one polynomial $p = 3wxy^3z - 7x^2y^4z + 2w^2y^4 - x^3y^2z^2 \in \mathbb{C}[w; x; y; z]$:

- First considering the graded lexicographic ordering $>_{\text{grlex}}$ ($w > x > y > z$), the terms of p would be ordered as

$$p = -x^3y^2z^2 - 7x^2y^4z + 2w^2y^4 + 3wxy^3z$$

with

- $\text{mdeg}(p) = (0; 3; 2; 2)$
- $\text{LT}(p) = -x^3y^2z^2 \in \mathbb{C}[w; x; y; z]$
- $\text{LC}(p) = -1 \in \mathbb{C}$
- $\text{LM}(p) = x^3y^2z^2 \in \mathbb{C}[w; x; y; z]$
- The total degree is $\max(0; 3; 2; 2) = 3 = \sum (0; 3; 2; 2)_i$
- On the other hand, if we consider the graded reverse lexicographic ordering $>_{\text{grevlex}}$ ($w > x > y > z$) now, things change a bit:

$$p = 7x^2y^4z - x^3y^2z^2 + 2w^2y^4 + 3wxy^3z$$

where

- $\text{mdeg}(p) = (0; 2; 4; 1)$
- $\text{LT}(p) = 7x^2y^4z \in \mathbb{C}[w; x; y; z]$
- $\text{LC}(p) = 7 \in \mathbb{C}$
- $\text{LM}(p) = x^2y^4z \in \mathbb{C}[w; x; y; z]$
- The total degree stays the same: $\max(0; 2; 4; 1) = 4 = \sum (0; 2; 4; 1)_i$

■ S-polynomial

The S-polynomial is a special kind of polynomial designed to cancel the leading terms of the two input polynomials and it is used in the Buchberger's algorithm to compute a Gröbner basis. We first need to define the least common multiple of monomials.

Definition 9 (Least common multiple) Let $x_1; \dots; x_n \in K$ be monomials. Then the least common multiple of $x_1; \dots; x_n$ is denoted and defined as

$$\text{LCM}(x_1; \dots; x_n) = x_1^{\alpha_1} \dots x_n^{\alpha_n} \quad \alpha_i = \max(\beta_i; \gamma_i) \quad (3.26)$$

Definition 10 (S-polynomial) Let $p; q \in K$. Their S-polynomial is defined as:

$$S(p; q) = \frac{\text{LCM}(\text{LM}(p); \text{LM}(q))}{\text{LT}(p)} p - \frac{\text{LCM}(\text{LM}(p); \text{LM}(q))}{\text{LT}(q)} q \quad (3.27)$$

To illustrate how one can create the S-polynomial, assume two polynomials $p; q \in R[x; y; z]$ and the standard lexicographic ordering $>_{\text{lex}}$ ($x > y > z$):

$$\begin{aligned} p &= 6x^2y^2z - 2xy^2z^3 \\ q &= 3x^3yz^4 + 5y^3z \end{aligned} \quad (3.28)$$

Then,

$$\begin{aligned} S(p; q) &= \frac{\text{LCM}(x^2y^2z; x^3yz^4)}{6x^2y^2z} p - \frac{\text{LCM}(x^2y^2z; x^3yz^4)}{3x^3yz^4} q = \\ &= \frac{x^3y^2z^4}{6x^2y^2z} p - \frac{x^3y^2z^4}{3x^3yz^4} q = \\ &= \frac{1}{6}(xz^3)(6x^2y^2z - 2xy^2z^3) - \frac{1}{3}(y)(3x^3yz^4 + 5y^3z) = \quad (3.29) \\ &= x^3y^2z^4 - \frac{1}{3}x^2y^2z^6 - x^3y^2z^4 - \frac{5}{3}y^4z = \\ &= -\frac{1}{3}x^2y^2z^6 - \frac{5}{3}y^4z \end{aligned}$$

■ Buchberger's algorithm

This algorithm takes the set of s polynomials $\{p_1; \dots; p_s\}$ from K and possibly the monomial ordering as input, although in most literature the monomial ordering is already assumed to be fixed beforehand, and outputs the Gröbner basis of the ideal $\langle p_1; \dots; p_s \rangle$. Assuming the reader is familiar with what has been declared above, the Buchberger's algorithm is simple and short, so let us present it straight away:

```

1 Input: An ideal  $F = \langle f_1, f_2, \dots, f_s \rangle \subseteq k[x_1, \dots, x_n]$ ,
2       A monomial order  $>$ 
3 Output: A Groebner basis  $G$  for the ideal  $F$ 
4
5 function Buchberger( $F, >$ ):
6      $G := F$ 
7     DO
8          $G' := G$ 
9         FOR each pair  $(p, q), p \neq q$  in  $G'$  DO
10             $r := \overline{S(p; q)}^G$ 
11            if  $r \neq 0$  then
12                 $G := G \cup \{r\}$ 
13        while  $G \neq G'$ 
14        return  $G$ 
15 end function

```

Listing 3.1: Buchberger's Algorithm for Computing a Gröbner Basis

Finally, $\overline{S(p; q)}^G$ denotes division of $S(p; q)$ by elements of G . Generally, the Gröbner basis is not unique, however there exists the reduced Gröbner basis, which is unique to an ideal, where the unnecessary generators are omitted. Again, this is similar to finding a base to linearly dependent set of vectors. The Gröbner basis G has the property that dividing $p \in K$ by G , we receive the remainder r , which is unique. Moreover, if $V(I)$ is finite the Buchberger's algorithm outputs G such that it contains a univariate polynomial and often even more

$$\begin{aligned}
 &g_n(x_n) \\
 &g_{n-1}(x_n; x_{n-1}) \\
 &g_{n-2}(x_n; x_{n-1}; x_{n-2}) \\
 &\vdots \\
 &g_1(x_n; x_{n-1}; x_{n-2}; \dots; x_1);
 \end{aligned} \tag{3.30}$$

so it is then simple to retrieve all the solutions.

3.2.2 Truncated normal forms

We start off with couple of definitions. Throughout this section, assume that $K = \mathbb{C}[x_1, \dots, x_n]$ and I is such that $V(I)$ is finite (also zero-dimensional). Before we define (truncated) normal forms, it is necessary to first introduce an auxiliary term, an exact sequence:

Definition 11 (Exact sequence) In group theory, the sequence groups and group homomorphisms

$$G_0 \xrightarrow{f_1} G_1 \xrightarrow{f_2} G_2 \xrightarrow{f_3} \dots \xrightarrow{f_n} G_n$$

is called exact at G_i if $\text{im}(f_i) = \ker(f_{i+1})$ and exact if it holds for all i .

3. Background

A fundamental algebraic fact is that $K=I$ is a finite-dimensional C -vector space if and only if I is zero-dimensional (i.e. I defines finitely many points counting multiplicity). Let $\dim_C(K=I) = n$. Then for any $p \in K$, multiplication by p induces a C -linear map

$$M_p : K=I \rightarrow K=I; \quad (g + I) \mapsto pg + I;$$

which in a fixed basis of $K=I$ becomes a $n \times n$ matrix. A classical result is that the eigenvalues of the multiplication matrices M_{x_i} recover the coordinates of the solutions in $V(I)$. In practice, one first fixes a basis of the quotient (or a normal form for $K=I$) and then builds the action matrices M_{x_i} . The common eigenpairs of these commuting matrices give all solutions. Now we can define normal form and its restricted version, TNF.

Definition 12 (Normal form). A normal form on $K=I$ is a linear map $N : R=I \rightarrow B=K$ with B being a vector subspace of dimension n over C such that $N|_B = \text{id}_B$ and the sequence

$$0 \rightarrow I \rightarrow R=I \xrightarrow{N} B \rightarrow 0$$

is exact.

Definition 13 (Truncated normal form) Let $B \subseteq V \subseteq K$, where B and V are finite dimensional vector subspaces, $x_i \in B \subseteq V$; $1 \leq i \leq n$ and $\dim_C(B) = \dim_C(K=I) = n$. A truncated normal form on $V=I$ is a linear map $N : V \rightarrow B$ such that the sequence

$$0 \rightarrow I \setminus V \rightarrow V \xrightarrow{N} B \rightarrow 0$$

is exact and $N|_B = \text{id}_B$. That is, N is the restriction to V of a normal form w.r.t. I .

Thus N projects V onto B along the $I \setminus V$. If in addition $x_i \in B \subseteq V$ for all i , then $M_{x_i} : B \rightarrow B$ given by $b \mapsto N(x_i b)$ is well-defined and coincides with multiplication by x_i in $K=I$. In this way a TNF encodes the quotient structure $K=I$ via linear algebra on V . In fact, one can show that if a linear map $N : V \rightarrow C$ satisfies conditions

- $\ker N = I \setminus V$,
- $\exists v \in V$ such that $v + I$ is a unit in $K=I$,
- $N|_W : W \rightarrow C$ is surjective,

then one can restrict it to a subspace $B \subseteq W$ of dimension n to obtain a TNF. In practice, W is chosen as the maximal subspace of V closed under multiplication by the x_i :

$$W = \{ p \in V : x_i \in V; 1 \leq i \leq n \} \tag{3.31}$$

Geometrically, $R=I$ is the coordinate ring of the variety $V(I)$, and the basis B of $R=I$ corresponds to a choice of free monomials or functions

distinguishing the points. In classical Gröbner basis methods one chooses B to be the set of standard monomials for some term order. In border basis approaches one chooses B to be an order ideal of monomials closed under division, without specifying an overall order. Border bases are more flexible than Gröbner bases (they are not tied to a single term order) and are known to offer improved numerical stability. Truncated normal forms further generalize this by allowing B and V to consist of non-monomial basis functions. For example, if many of the roots lie in a real interval or box, one may choose B of Chebyshev or orthogonal polynomial basis functions to improve conditioning. In effect, a TNF is just a projector onto a complement of $I \setminus V$, and any basis of that complement (monomial or not) can be used.

■ Example

Consider the system in $K = \mathbb{C}[x; y]$

$$I = \langle x^2 + y^2 - 1; x - y \rangle \tag{3.32}$$

This consists of a unit circle and the line $x = y$. The Bézout bound is 2, and indeed there are two solutions: $(x_1; y_1) = (1/\sqrt{2}; 1/\sqrt{2})$ and $(x; y) = (-1/\sqrt{2}; -1/\sqrt{2})$. A TNF approach proceeds as follows. Choose $V = \text{span}\{1; x; y; x^2; xy; y^2\}$ (all monomials up to degree 2) so that $I \setminus V$ has codimension $= 2$. One finds by linear algebra a map $N : V \rightarrow \mathbb{C}^2$ whose kernel contains $I \setminus V$. A suitable basis of the quotient is $B = \{1; x\}$, because in $K=I$ we have $x = y$ and $x^2 = \frac{1}{2}$ (from $x^2 + y^2 = 1$), so $\{1; x\}$ spans $K=I$. The TNF projection gives the action of x :

$$x \cdot 1 = x; \quad x \cdot x = x^2 = \frac{1}{2} \pmod{I} \tag{3.33}$$

In the basis $\{1; x\}$, the multiplication matrix is

$$M_x = \begin{pmatrix} 0 & \frac{1}{2} \\ 1 & 0 \end{pmatrix}; \tag{3.34}$$

whose eigenvalues are $\pm 1/\sqrt{2}$. These indeed are the x -coordinates of the two solutions (and $y = x$). From the eigenvector one recovers each full solution. This simple example illustrates how a TNF (even one using just monomial basis here) yields the companion matrix whose eigenvalues gives the answers.

■ 3.2.3 Macaulay's Resultants

A resultant is a scalar invariant that encodes whether two (or more) polynomials share a common root in a given variable and, by extension, it provides a mechanism for eliminating that variable from a system. Historically, the idea originates in the 19th century work of Sylvester and Bézout. Given two univariate polynomials

$$p(x) = a_m x^m + \dots + a_0; \quad q(x) = b_n x^n + \dots + b_0 \tag{3.35}$$

Example

Given

$$\begin{aligned} p(x; y) &= x^2 + yx + 1 = 0; \\ q(x; y) &= x + y^2 - 3 = 0; \end{aligned} \tag{3.41}$$

view both equations as polynomials in x with coefficients in $C[y]$ (i.e. $p, q \in C[y][x]$). Here, the $\deg_x p = 2$ and $\deg_x q = 1$, so the Sylvester matrix is $(2 + 1) \times (2 + 1)$ matrix whose first block of rows (only 1 row) are the coefficients of p and the next two rows belong to q :

$$\text{Syl}_{p,q} = \begin{pmatrix} 1 & y & 1 & 0 & 0 \\ 0 & 1 & y^2 & 3 & 0 \\ 0 & 0 & 1 & y^2 & 3 \end{pmatrix} \tag{3.42}$$

Then by construction, we have

$$\text{Res}_x(p; q) = \det(\text{Syl}_{p,q}) = 0 \iff x : f(x; y) = g(x; y) = 0 : \tag{3.43}$$

Next, let us compute the resultant:

$$\begin{aligned} \text{Res}_x(p; q) &= \begin{vmatrix} 1 & y & 1 & 0 & 0 \\ 0 & 1 & y^2 & 3 & 0 \\ 0 & 0 & 1 & y^2 & 3 \end{vmatrix} = \\ &= y^2 \cdot 3^2 + 1 \cdot y \cdot y^2 \cdot 3 = \\ &= y^4 - y^3 - 6y^2 + 3y + 10 \end{aligned} \tag{3.44}$$

We obtain univariate polynomial in y of degree 4. The polynomial was solved numerically and has two real $y_{1,2}$ and two complex solutions $y_{3,4}$:

$$\begin{aligned} y^4 - y^3 - 6y^2 + 3y + 10 = 0 \implies & y_1 = 2 \\ & y_2 = 2.0796 \\ & y_{3,4} = 1.5398 \pm 0.1826i \end{aligned} \tag{3.45}$$

Finally, we take each y_i , substitute it back to the original system 3.41 and solve two equations in x to retrieve the corresponding values of x

$$\begin{aligned} x_1 &= -1 \\ x_2 &= 1.3247 \\ x_{3,4} &= 0.6624 \pm 0.5623i \end{aligned} \tag{3.46}$$

Macaulay's resultants generalize this process onto systems of equations in n variables.

3.3 Minimal solvers

Minimal solvers are specialized algorithms that solve systems of polynomial equations arising from geometric constraints, using only the absolute minimum

- **Integration with RANSAC** : Minimal solvers are central to robust estimation, and hybrid schemes combine minimal sampling with non-minimal refinement [28] (LO-RANSAC) to balance speed and accuracy.
- **Event-Camera Motion (Five-Point Eventail Solver)**: A recent minimal solver for event-camera egomotion [10] jointly estimates line parameters and velocity, improving robustness in high-speed scenarios.
- **Resultant-Based Methods** : Emerging approaches use sparse resultants and Newton polytopes to derive highly efficient solvers, sometimes outperforming Gröbner-basis based ones in size and numerical stability [5].
- **Automated Solver Generators** : Libraries like PoseLib [18] provide tool-chains for generating, optimizing, and benchmarking minimal solvers across dozens of geometric problems.

Chapter 4

Implementation

In this chapter I briefly describe the implementation of the three solvers, that were provided to me, together with my implementation of tests.

4.1 ZeroDimSolve (Maple)

This one is implemented in Maple by my supervisor, doc. Ing. Tomáš Pajdla, Ph.D. and looks like the following:

```
1 fB2Coffs := proc(f,B)
2 local m, t;
3 t := table(zip((a,b)->b=a,[coefs(f,B,'m')],[m]));
4 map(b->`if`(assigned(t[b]),t[b],0),B)
5 end proc;
6
7 ZeroDimSolve := proc(F)
8 local J, o, B, f, fBmJ, Mt, V, N, C, S;
9 J := PolynomialIdeals[Radical](PolynomialIdeals[
10 PolynomialIdeal](F));
11 o := tdeg(op(indets(F)));
12 B := Groebner[NormalSet](J, o)[1];
13 f := add(zip((x,y)->y*x, convert(RandomVector(nops(o)),
14 list), [op(o)]));
15 fBmJ := Groebner[NormalForm](map(b->f*b, B), J, o);
16 Mt := Matrix(map(f->fB2Coffs(f, B), fBmJ));
17 V := LinearAlgebra[Eigenvectors](evalf(Mt))[2];
18 V := V . (LinearAlgebra[MatrixInverse](LinearAlgebra[
19 DiagonalMatrix](V[1])));
20 N := Groebner[NormalForm]([op(o)], J, o);
21 C := map(n->fB2Coffs(n, B), N);
22 S := [ListTools[Transpose](map(c->convert((Matrix(c)) .
23 V, list), C)), op(o)];
24 end proc;
```

`fB2Coffs` is a helper method that returns the coefficient vector of f in basis B . The `ZeroDimSolve` procedure expects system of polynomial equations \mathbb{F} , returning solutions in S . This solver works properly under two assumptions. First, $V(F)$ has to be zero-dimensional, i.e. only finitely many points lie on the variety and, secondly, there are no repeated solutions.

4. Implementation

On the first line, J is the radical ideal of $\mathbb{H}[F]$. Next, σ is set to the $>$ grevlex monomial order and B is the monomial base of $K[x_1; \dots; x_n] = J$. On the next few lines, the multiplication matrix M_t is assigned. Then the eigenvalues and eigenvectors of M_t are numerically evaluated. On the line 17, the variables are reduced into the normal basis, producing coefficient vector C . Finally, the solutions are then obtained from the normalized matrix V and C .

4.2 EigenvalueSolver (Julia)

Algorithm 2 in Bender & Telen (2022) [4] is an eigenvalue-based solver that takes an admissible tuple $(F; A_0; (E_0; \dots; E_s); D)$ (explained in [4], table 1) and returns a candidate set solutions of the polynomial system $F(x) = 0$ in the algebraic torus $(\mathbb{C}^*)^n$ (meaning the set contains all the true solutions, but there might be some fake ones). The algorithm works by building a Macaulay matrix $M(F; E; D)$ for the system and performing linear algebra to extract the multiplication operators and their eigenstructure. I describe its steps of Algorithm 2 according to my understanding:

- Macaulay matrix and cokernel (steps 1-3): First, the algorithm generates a random mixing matrix O of size $(\sum_{j \in E} |E_j|) \times |D|$ and multiplies it with the Macaulay matrix $M(F; E; D)$ to form a compressed matrix $M_O = M(F; E; D) \cdot O$. Intuitively, this projects the large Macaulay matrix into a smaller space while preserving its kernel. Next, it computes the (right) cokernel of this compressed matrix via a singular value decomposition (SVD). The cokernel represents linear relations among the rows of $M(F; E; D)$ and will serve as a basis for the quotient algebra of the system.
- Choosing f_0 and computing N_{f_0} (steps 4-8): The algorithm then picks a generic (random) element f_0 in the polynomial space R^{A_0} . This f_0 plays the role of a generic linear form used to define a basis. It computes the matrix $N_{f_0} = \text{Coker}(F; E; D) \cdot M(f_0; E_0; D)$. In essence, N_{f_0} applies the multiplication-by- f_0 operator (restricted to supports E_0 and D) to the cokernel basis. A QR-decomposition with column pivoting is performed on N_{f_0} . This yields an upper-triangular factor R_0 of full rank (of size \dots , where \dots is the number of solutions) and a column pivot permutation p . The pivot columns $p_1; \dots; p_{\dots}$ identify a subset $B \subseteq E_0$ of exponent vectors. These exponents form a basis of the quotient space (one for each solution).
- Building multiplication matrices (steps 9-12): With the basis B in hand, the algorithm loops over each exponent $j \in A_0$ (where $A_0 = 0; \dots; k$ is the chosen support containing the origin). For each j , it forms the matrix

$$N_{x^{a_j}; B} = \text{Coker}(F; E; D) \cdot M(x^{a_j}; B; D); \quad (4.1)$$

representing multiplication by the monomial x^i on the subspace spanned by B . It then solves the linear system $R_0^T X = N_{x^i;B}^i Q_0$ by back-substitution (where Q_0 is the unitary factor from QR). The solution X defines the compressed multiplication matrix M_{x^i} in the R_0 -basis. In this way, the algorithm obtains all multiplication-by- x^i operators on the quotient algebra (for $i \in A_0$).

- Forming random eigenvalue test matrices (steps 12-15): Next, the algorithm takes random linear combinations of the computed multiplication matrices. It sets

$$M_g = \sum_{i \in A_0} c_i M_{x^i} \quad (4.2)$$

and similarly M_h with a different random combination. These matrices M_g and M_h commute with each other (under the admissibility conditions) and thus share eigenvectors. The algorithm computes the distinct eigenvalues $\lambda_1, \dots, \lambda_k$ of M_g and the corresponding left-eigenspaces V_i (one for each eigenvalue).

- Extracting solutions (steps 16-24): For each eigenvalue λ_i , the algorithm finds the intersection of the corresponding left-eigenspace V_i of M_g with that of M_h via the subroutine `GetEigenspace()`. If the resulting space V is non-zero, then V is (generically) a one-dimensional eigenspace corresponding to a single solution. Let $m = \dim(V)$. The algorithm chooses a random test matrix T of size $m \times m$ and computes, for each $j = 1, \dots, k$, the matrix

$$\hat{M}_j = V M_{x^j} T (V T)^{-1} \quad (4.3)$$

The average trace $\lambda_{ij} = \text{trace}(\hat{M}_j) = m$ then gives the value of x^j at the solution, because it equals the eigenvalue of multiplication by x^j on that eigenspace. Finally, knowing all λ_{ij} for the generators in A_0 , the algorithm recovers the actual solution coordinates $x \in (C \setminus \{0\})^n$ by solving the relations 2.3, 2.4 of the paper. This yields one candidate solution per eigenvalue. All such x are collected into the output set Z . By Theorem 2.2 of the paper, this set Z is guaranteed to contain every true solution of F in $(C \setminus \{0\})^n$.

The table 1 in [4] defines 5 classes of polynomial systems in which I categorized the systems representing selected tasks. The EigenvalueSolver package implements solving functions for each category. The names are of the following form - `solve_XY_*****`, where $XY \in \{CI; OD\}$ and $***** \in \{dense; mixed; multi_dense; multi_unmixed; unmixed\}$. Be aware that not all combinations are possible. CI stands for complete intersection, which is the same as saying that the given system has equally many equations as variables. OD stands for overdetermined. However all of the selected problems fall into the dense category.

4.3 Elimination templates (Matlab)

An elimination template is essentially a coefficient matrix (often a Macaulay matrix) that transforms a system of polynomial equations into a form suitable for constructing an action matrix. The action matrix's eigenvectors correspond to the solutions of the original system. The key advantage of elimination templates is that they can be precomputed and applied to any instance of a problem with the same monomial structure but varying coefficients, which proved to be very fast, see [5,21]. Below is displayed one from Matlab (for complete implementation see the GitHub repository¹ or [21]).

```

1  function [C,U,dU] = coefs_p4p_fr(data)
2
3  M1 = data{1};
4  M2 = data{2};
5  M3 = data{3};
6
7  C = zeros(4,20);
8  C(25) = M1(1)*M2(1)+M1(2)*M2(2)+M1(3)*M2(3)+M1(4)*M2
9  (4);
10 C(26) = M1(1)*M3(1)+M1(2)*M3(2)+M1(3)*M3(3)+M1(4)*M3
11 (4);
12 .
13 .
14 C(39) = M2(9)*M3(25)+M2(13)*M3(21)+M2(10)*M3(26)+M2
15 (14)*M3(22)+M2(11)*M3(27)+M2(15)*M3(23)+M2(12)*M3(28)+
16 M2(16)*M3(24);
17 C(71) = M2(9)*M3(29)+M2(13)*M3(9)+M2(10)*M3(30)+M2
18 (14)*M3(10)+M2(11)*M3(31)+M2(15)*M3(11)+M2(12)*M3(32)+
19 M2(16)*M3(12);
20 C = C./repmat(sqrt(sum(C.*conj(C),2)),1,size(C,2));
21 .
22 .
23 .
24 end

```

Listing 4.1: A snippet of elimination template creation

data encodes the specific task parameters. From the row 7 on, the first part of elimination template is computed. Outside it is passed to another function that creates an action matrix, eigenvalues of which determine the tasks solutions. The whole process takes only few lines of code in Matlab, so it is easily understood.

¹<https://github.com/martyushev/EliminationTemplates>

4.4 Tests

All of the tests were done in the same language as their solver implementation. To test the numerical stability, I have extracted the solutions into .csv files and then handled them separately in Python with the NumPy library. The python file `Solution_comparator.py`, attached to the thesis, implements couple of functions that handle .csv files. Be aware, that it may not be in working state, as the file was continuously changing. It should only serve as insight into the matter. For testing of the speed, I used time measuring libraries in the programming languages of the solvers

- Matlab - `timeit()` function
- Maple - `Usage[RealTime]()` from the package `CodeTools`
- Julia - `@time` and `@btime` from package `BenchmarkTools`

The `EigenvalueSolver` outputs a set of candidate solutions, so it was necessary to find the true solutions somehow in order to compare them with solutions from the other solvers. This was implemented in the function `match_rows_by_distance()`, that takes two numpy arrays of complex numbers and matches the solutions one with another (as the name suggests, one row represents one solution, as it was ensured in the tests that they were saved in the correct format). Other than that, it contains functions implemented much earlier, which do not solve this, and some helper function to trim or sanitize the .csv files, so NumPy can read them.

Chapter 5

Experiments

In this chapter I present the results that I was able to achieve in the available time span. Note, that with two (Optimal pose 2pt v2 and Unsynchronized relative pose) tasks out of the 8 selected there occurred errors in the Julia package that I was not able to resolve myself (such as running out of memory when calling the solver or referencing invalid indices and couple others...) as I am not experienced enough in the Julia programming language to modify the solver implemented by someone else. Therefore, I decided to discard them from the experiments, leaving Optimal PnP (Hesch) , Optimal PnP (Cayley) , Absolute pose , Absolute pose quivers , Rolling shutter pose and Stitching in the pool of tasks. Everything in this chapter was tested on a machine with 16 GB of RAM and AMD Ryzen 5 5600 6-Core Processor running at 3.50 GHz. First, in the table 5.1 is category of each of the six tasks as defined in [4] and described above in 4.

Notice, that it is the most restricted class that the systems fit into. That does not mean they can not fit into more general class and indeed it happened when testing, for example, the absolute pose problem. Both the functions `solve_CI_dense` and `solve_CI_mixed` returned different sets of candidate. Interestingly, the one from `solve_CI_mixed` contained exactly 16 elements (so, it could not be smaller, see 2.1) as opposed to the other with 36.

Absolute pose	CI - dense	36
Absolute pose quivers	CI - dense	57
Optimal PnP (Cayley)	OD - dense	40
Optimal PnP (Hesch)	CI - dense	27
Rolling shutter	OD - dense	8
Stitching	CI - dense	81

Table 5.1: Categorization of tasks into classes from [4] and size of the largest candidate set.

5.1 Numerical stability

To test the solvers against each other, I have picked one random dataset, distributed it to all the solvers and examined the solutions afterwards. This

was done for each of the tasks. However, in order not to introduce dozens of seemingly random tables, let the following one summarize the numerical stability. In this part of testing, I have generated a random dataset in Matlab and converted it into formats suitable for the tests in the other programming languages. Then I run the tests and saved the returned solution into .csv files and matched the solutions as mentioned above. Note, that this was done for all pairs of solution sets, because they differ in the order of solutions, at the very least.

The table 5.2 presents values representing the total relative error, computed across all matched solutions and all variables. As shown in [1], the elimination templates from Matlab are numerically accurate. Notice that the EigenvalueSolver is also very accurate, few orders more than the ZeroDimSolve. However, Maple does not target numerical accuracy.

	Julia \$ Matlab	Matlab \$ Maple
Absolute pose	1.221751e-10	2.041053e-10
Absolute pose quivers	3.423370e-10	4.835119e-06
Optimal PnP (Cayley)	2.760135e-08	3.872734e-05
Optimal PnP (Hesch)	1.128568e-11	8.818869e-07
Rolling shutter	1.910939e-11	2.526061e-12
Stitching	1.104621e-12	2.625615e-12

Table 5.2: Total relative error of all solutions provided by comparison of two different solvers.

■ 5.2 Speed

To test the speed of the solvers, I used already implemented time measuring functions (look at 4.4 to see which ones). I have run the tests multiple (>10) times and averaged the time over the runs. Results of this phase are in table 5.3.

It is also possible to call the solving function for more generic systems which also return correct candidates, but takes much longer time to finish. For example, I have tested this on the absolute pose problem, which fits into CI-dense category. Measuring the time spent in the corresponding function yielded 71,497 ms. However, one can also obtain a feasible candidate set by calling solve_CI_mixed, which is not as restrictive, but takes around half a second to finish. That is around 7 times slower.

The testing of the EigenvalueSolver was split into two parts. The first values consist of time that the solver spent creating the admissible tuple with finding the solution itself and the second only shows how much time was spent solving the system. Observe, that when testing Optimal PnP (Cayley), it took more time to only solve the system than with the creation of admissible tuple. I can thereby confirm the behaviour, explained in [4], section 4.2.1., which may happen for overdetermined systems. So it is no mistake.

The EigenvalueSolver was much slower in comparison with the elimination templates and in scope of our systems, because the number of solutions, that the systems we work with have, are much lower than where the solver shines, which is around couple of hundreds. This is to be compared with table 4 in [4], for example. The ZeroDimSolve is the slowest as the computation of Gröbner basis is generally very slow.

All of the solvers proved to be numerically stable, but there are clear differences with regards to speed and time.

Task / Solver	EigenvalueSolver - solve... & Algorithm 2 only	ZeroDimSolve	Matlab
Absolute pose	71,497 & 7,884458	136,120000	0,292825
Absolute pose quivers	146,463645 & 14,272391	104,001	0,341182
Optimal PnP (Cayley)	31,211236 & 74,025717	0,518000	1,130988
Optimal PnP (Hesch)	19,071680 & 4,591700	172,880000	0,442684
Rolling shutter	7,134899 & 4,883463	22,000000	0,183761
Stitching	100,306998 & 15,305004	58,002050	0,192021

Table 5.3: This table contains information about how long it took each solver to produce the solution, with the unit being milliseconds (ms).



Chapter 6

Conclusion

At the beginning, I have familiarized the reader with a cluster of tasks from computer vision. Next, I provided insights into the mathematical background that covers the implementation of minimal solvers in practice, with some specific ones. In 4, I explained the specific implementations and my way of testing of the techniques. Finally, I demonstrated the behaviour, speed and numerical stability in particular, of the minimal solvers in experiments (5).

The EigenvalueSolver takes the second place with regard to speed as the polynomial systems we are working with in this thesis having very small amount of solutions and the construction of admissible tuples takes up remarkable time. The ZeroDimSolve, on the other hand, is slow due to the computation of Gröbner basis. All of the minimal solvers are numerically stable, but the EigenvalueSolver and Matlab elimination templates are few orders of magnitude more precise than ZeroDimSolve.



Bibliography

- [1] math.arizona.edu. <https://math.arizona.edu/~cais/223Page/hout/236w06fields.pdf> . [Accessed 03-05-2025].
- [2] Cenek Albl, Zuzana Kukelova, Andrew Fitzgibbon, Jan Heller, Matej Smid, and Tomas Pajdla. On the two-view geometry of unsynchronized cameras, 2017. URL:<https://arxiv.org/abs/1704.06843> , arXiv: 1704.06843.
- [3] D. Arganbright. Practical Handbook of Spreadsheet Curves and Geometric Constructions. Taylor & Francis, 1993. URL: <https://books.google.com/books?id=Wt4a96UJI78C>
- [4] Matías R. Bender and Simon Telen. Yet another eigenvalue algorithm for solving polynomial systems, 2022.arXiv:2105.08472 .
- [5] Snehal Bhayani, Janne Heikkilä, and Zuzana Kukelova. Sparse resultant-based minimal solvers in computer vision and their connection with the action matrix. *J. Math. Imaging Vis.* , 66(3):335–360, June 2024.
- [6] Martin Bujnak, Zuzana Kukelova, and Tomas Pajdla. New efficient solution to the absolute pose problem for camera with unknown focal length and radial distortion. In Ron Kimmel, Reinhard Klette, and Akihiro Sugimoto, editors, *Computer Vision – ACCV 2010* , pages 11–24, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [7] Martin Byröd, Matthew Brown, and K.J. Åström. Minimal solutions for panoramic stitching with radial distortion. 01 2009. doi:10.5244/C.23.41 .
- [8] D.A. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer International Publishing, 2015. URL: <https://books.google.cz/books?id=yL7yCAAQBAJ>
- [9] Jian Dai. Euler Rodrigues formula variations, quaternion conjugation and intrinsic connections. *Mechanism and Machine Theory* 92, 10 2015. doi:10.1016/j.mechmachtheory.2015.03.004 .

- [10] Ling Gao, Hang Su, Daniel Gehrig, Marco Cannici, Davide Scaramuzza, and Laurent Kneip. A 5-point minimal solver for event camera relative motion estimation. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, page 8015–8025. IEEE, October 2023. URL: <http://dx.doi.org/10.1109/ICCV51070.2023.00739>, doi : 10.1109/iccv51070.2023.00739.
- [11] 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, 2011. doi : 10.1109/ICCV.2011.6126266.
- [12] Heisuke Hironaka. Resolution of singularities of an algebraic variety over a field of characteristic zero: I. *Annals of Mathematics*, 79(1):109–203, 1964. URL: <http://www.jstor.org/stable/1970486>.
- [13] Michael Hofmann and Darius M. Gavrilă. Multi-view 3d human pose estimation combining single-frame recovery, temporal integration and model adaptation. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2214–2221, 2009. URL: <https://api.semanticscholar.org/CorpusID:2485854>.
- [14] Fredrik Kahl and Didier Henrion. Globally optimal estimates for geometric reconstruction problems. *Int. J. Comput. Vis.*, 74(1):3–15, August 2007.
- [15] Yubin Kuang and Karl Åström. Pose estimation with unknown focal length using points, directions and lines. In *[Host publication title missing]*, pages 529–536. Computer Vision Foundation, 2013. The authoritative version of this paper will be available in IEEE Xplore.; IEEE International Conference on Computer Vision (ICCV), 2013 ; Conference date: 01-12-2013 Through 08-12-2013.
- [16] Zuzana Kukelova, Martin Bujnak, and Tomas Pajdla. Automatic generator of minimal problem solvers. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 302–315, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [17] Viktor Larsson. *Computational Methods for Computer Vision: Minimal Solvers and Convex Relaxations*. Doctoral thesis (monograph), Lund University, 2018. Defence details Date: 2018-06-01 Time: 13:15 Place: lecture hall MH:H, Centre for Mathematical Sciences, Sölvegatan 18, Lund University, Faculty of Engineering LTH, Lund External reviewer Name: Li, Hongdong Title: Doctor A liation: Australian National University —.
- [18] Viktor Larsson and contributors. PoseLib - Minimal Solvers for Camera Pose Estimation, 2020. URL: <https://github.com/viktorlarsson/PoseLib>.

- [19] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate $o(n)$ solution to the pnp problem. *International Journal of Computer Vision*, 81, 02 2009. doi : 10. 1007/s11263-008-0152-6.
- [20] Evgeniy Martynushev, Snehal Bhayani, and Tomas Pajdla. Automatic solver generator for systems of laurent polynomial equations, 2023. arXiv: 2307. 00320.
- [21] Evgeniy Martynushev, Jana Vrablokova, and Tomas Pajdla. Optimizing elimination templates by greedy parameter search, 2022. arXiv: 2203. 14901.
- [22] Bernard Mourrain, Simon Telen, and Marc Van Barel. Truncated normal forms for solving polynomial systems: Generalized and efficient algorithms, 2018. URL: <https://arxiv.org/abs/1803.07974>, arXiv: 1803. 07974.
- [23] Gaku Nakano. Globally optimal dls method for pnp problem with cayley parameterization. pages 78.1–78.11, 01 2015. doi : 10. 5244/C. 29. 78.
- [24] David Nistér. An efficient solution to the five-point relative pose problem. <https://www-users.cse.umn.edu/~hspark/CSci5980/nister.pdf>, 2004. [Accessed 19-05-2025].
- [25] Olivier Saurer, Marc Pollefeys, and Gim Lee. A minimal solution to the rolling shutter pose estimation problem. 09 2015. doi : 10. 1109/IR0S. 2015. 7353540.
- [26] Gerald Schweighofer and Axel Pinz. Globally optimal $o(n)$ solution to the pnp problem for general camera models. 01 2008. doi : 10. 5244/C. 22. 55.
- [27] Jingnan Shi. Perspective-n-Point: P3P — jingnanshi.com. https://jingnanshi.com/blog/pnp_minimal.html, 2021. [Accessed 19-05-2025].
- [28] John Smith. RANSAC: should we always sample with the minimal number of data points? <https://stats.stackexchange.com/questions/322812/ransac-should-we-always-sample-with-the-minimal-number-of-data-points>, 2018. [Accessed 19-05-2025].
- [29] 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, 2017. doi : 10. 1109/TPAMI. 2016. 2598331.
- [30] Simon Telen. *Solving Systems of Polynomial Equations*. PhD thesis, Arenberg Doctoral School, 2020-09-21.

- [31] Simon Telen, Bernard Mourrain, and Marc Van Barel. Solving polynomial systems via truncated normal forms. *SIAM J. Matrix Anal. Appl.*, 39:1421–1447, 2018. URL: <https://api.semanticscholar.org/CorpusID:119286611>.
- [32] Wikipedia contributors. Rodrigues' rotation formula. https://en.wikipedia.org/wiki/Rodrigues'_rotation_formula, 2025. [Online; accessed: 2025-4-27]. URL: https://en.wikipedia.org/wiki/Rodrigues'_rotation_formula.