3D Relative Pose Estimation from Six Distances

Nikolas Trawny, Xun S. Zhou, and Stergios I. Roumeliotis
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455
E-mail: {trawny,zhou,stergios}@cs.umn.edu

***Abstract*—In this paper, we present three fast, hybrid numeric-algebraic methods to solve polynomial systems in floating point representation, based on the eigendecomposition of a so-called multiplication matrix. In particular, these methods run using standard double precision, use only linear algebra packages, and are easy to implement. We provide the proof that these methods do indeed produce valid multiplication matrices, and show their relationship. As a specific application, we use our algorithms to compute the 3D relative translation and orientation between two robots, based on known egomotion and six robot-to-robot distance measurements. Equivalently, the same system of equations arises when solving the forward kinematics of the general Stewart-Gough mechanism. Our methods can find all 40 solutions, trading off speed (0.08s to 1.5s, depending on the choice of method) for accuracy.**

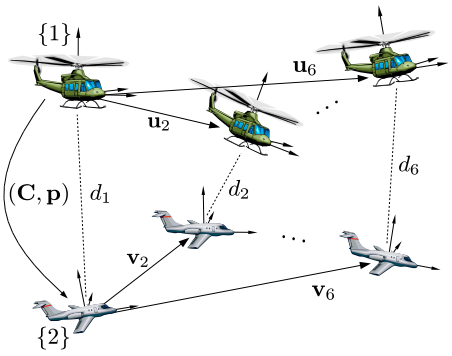


Fig. 1. 3DD Problem geometry.

I. Introduction

For the successful operation of multi-robot systems, accurate knowledge of the 3D robot-to-robot position and orientation (pose) is essential. Common methods to determine this relative transformation, such as direct manual measurements, absolute measurements with respect to a common frame (e.g., using GPS), or indirect correlation of sensor measurements (e.g., map or image matching) either provide insufficient accuracy, or can be infeasible in GPS-denied or featureless areas. In recent work [1], we have addressed this problem by using robot-to-robot distance measurements from different vantage points. Our previous method uses 10 distance measurements plus known robot egomotion to linearly compute the robot-to-robot relative pose. In order to increase robustness to outliers and noise (e.g., using RANSAC), the aim of this paper is to present a solution algorithm for the *minimal* problem that requires only 6 distance measurements to arrive at a discrete set of solutions for the six unknowns.

The forward kinematics of the general Stewart-Gough mechanism [2] represent the mechanical analogue to this minimal problem. The general Stewart-Gough mechanism consists of two platforms connected by six legs, whose lengths determine the pose of the end platform relative to the base platform. The forward-kinematics problem is then to determine the relative pose of both platforms given the leg lengths and their attachment coordinates. For conciseness, we will from now on refer to both this, as well as the 3D robot relative pose problem as the 3DD (3D-distance) problem. It has been shown that the 3DD problem requires solving a system of polynomial equations that has 40 (generally complex) solutions [3], [4], all of which can be real [5]. Unfortunately, solving this polynomial system has proven quite difficult. The currently available techniques capable of finding all 40 solutions are either slow and not suitable for real-time application (run-time can reach minutes, even hours) and/or they are highly non-trivial to implement (they require non-standard data types, specialized libraries, and a large amount of symbolic calculations).

The contribution of this paper is twofold: (i) We present the theoretical proofs for the correctness of two polynomial system solvers. In particular, these proofs cover a generalization of [6], [7], as well as the method of [8], and establish their relationship. Both approaches employ the paradigm of solving polynomial systems using the eigendecomposition of a so-called multiplication matrix [9], also referred to as action matrix. These solvers are generally applicable to a wide range of problems arising in robotics and computer vision. (ii) We apply these solvers to the 3DD problem, and provide solution algorithms that are very fast (0.08s - 1.5s), use standard double precision data types, and can be very easily implemented using only linear algebra libraries.

Following the description of related work (Section II), we present the 3DD problem in Section III, motivating the need for fast polynomial solvers. We provide a brief background on the theory of polynomial system solving in Section IV and outline the solution algorithms in Section V. We end the paper with simulation results (Section VI), and conclusions and an outlook on future work in Section VII.

II. Related Work

Although the two incarnations of the 3DD problem - determining the relative pose of two robots from distance measurements, or solving the forward kinematics of the general Stewart-Gough mechanism - are mathematically equivalent, in that they both lead to the same system of multivariate polynomials, research has mostly concentrated on the latter. Current algorithms for solving the forward kinematics problem

of the general Stewart-Gough mechanism can be broadly divided into numeric and algebraic methods.

Newton's method is a widely used numeric approach to obtain solutions to the forward kinematics of the Stewart mechanism [10] due to its high speed and ease of implementation. It works extremely well when initialized close to a true solution, as is the case when tracking a slowly-moving manipulator at high update rates. However, it may converge very slowly or even diverge when initialized poorly, and will generally not be able to find all 40 solutions without prior information about their approximate locations.

Newton's method is also a building block of polynomial continuation, that uses homotopy [11] to track the known solutions of a starting system continuously until they match those of the actual system of interest. Polynomial continuation was the earliest numerical tool used to find all 40 solutions of the forward kinematics problem [3]. Free implementations of this method, such as PHCpack [12], can successfully solve this problem, and were used as ground-truth in our simulations. Unfortunately, in our experiments, the black-box implementation of PHCpack required on average more than 120 s to solve one instance of the problem and is hence too slow for real-time application (although the forward kinematics have reportedly been solved using continuation methods in 14 s [4]). Moreover, continuation methods may sometimes miss solutions [13].

One algebraic method for solving a polynomial system is to compute its Gröbner basis [14] with respect to lexicographical monomial ordering. Such a Gröbner basis is essentially an equivalent polynomial system from which the solution can be easily obtained. In particular, for this problem it will contain one univariate polynomial of degree 40, whose roots form the solutions of the original system for that variable. The values of the remaining variables can be obtained from the other polynomials of the Gröbner basis via back substitution. Despite recent advances in algorithms to compute Gröbner bases, computing one symbolically for this problem has proven intractable, and even for specific numerical instances using rational numbers still requires several seconds up to minutes [13].

A second algebraic method is to use dialytic elimination or resultants to obtain the 40-degree univariate polynomial. This has first been achieved by Husty [15], and since then refined [16], [17]. To our knowledge, Lee and Shim's algorithm [17] is currently the fastest solution method for the forward kinematics (the authors report timings of 0.02 s). Unfortunately, all the elimination algorithms mentioned above are quite challenging to implement, they require non-standard, high-precision data-types (e.g., with 30 digits precision [17]) to cope with numerical error accumulation, and a large amount of symbolic computations.

The algorithms presented in this paper solve a system of polynomials by an eigendecomposition of an associated multiplication matrix, a general hybrid algebraic-numeric approach pioneered by Auzinger and Stetter [9] and described in detail in [18]. This concept has recently been successfully applied to solve minimal problems in computer vision [19], [20]. More precisely, our methods are inspired by the general techniques to solve polynomial systems in floating point representation presented recently in [19] and [8]. Two modifications to [19] have been introduced with the aim of improving numerical stability [6], [7], however, without proving that these methods will indeed yield a valid multiplication matrix, and without providing conditions under which they will work. In this paper, we prove that a generalized version of [6], [7], as well as the approach in [8] do indeed yield valid multiplication matrices for a specific class of problems in which certain rank conditions are fulfilled (which are tacitly implied in [6], [7]). Our methods make extensive use of linear algebra, in particular QR factorization, and require only standard double precision. Their high speed and accessibility make them therefore attractive for real-time applications in industrial settings, e.g., in flight simulator control.

III. Problem Formulation

To motivate the need for fast polynomial solvers, in this section we will outline the 3DD problem and show how it results in a system of polynomials. In the next two sections we will then discuss algorithms to solve this system.

Assume that two robots move in 3D and take six robot-to-robot distance measurements $d_i, i = 1, \ldots, 6$. We assume without loss of generality that the global frames of each robot, $\{1\}$ and $\{2\}$, are attached to the points where the first mutual measurement takes place (cf. Fig. 1). Further, we assume that each robot knows the coordinates, $\mathbf{u}_i := {}^1\mathbf{u}_i$ and $\mathbf{v}_i := {}^2\mathbf{v}_i$, of its location at the time of the remaining five measurements with respect to its own global frame of reference. The objective is to find the 6 degree-of-freedom transformation, i.e., the translation $\mathbf{p} := {}^1\mathbf{p}_2$ and rotation $\mathbf{C} := {}^1_2\mathbf{C}$ of the second frame with respect to the first.

The distance measurements can be expressed as the length of vector $\mathbf{w}_i$, $i = 1, \ldots, 6$, connecting the two robots at the time of measurement.

$$d_i = ||\mathbf{w}_i||_2 = \sqrt{\mathbf{w}_i^T \mathbf{w}_i}, \quad \mathbf{w}_i := \mathbf{p} + \mathbf{C}\mathbf{v}_i - \mathbf{u}_i$$

Squaring each distance, and noting that $\mathbf{u}_1 = \mathbf{v}_1 = \mathbf{0}$, we obtain the following six polynomial constraints

$$\mathbf{p}^T\mathbf{p} - d_1^2 = 0 \tag{1}$$

$$-\mathbf{u}_i^T\mathbf{p} + \mathbf{v}_i^T\mathbf{C}^T\mathbf{p} + f_i = 0 \quad i = 2, \ldots, 6 \tag{2}$$

$$\text{where } f_i = \frac{1}{2}(d_1^2 + \mathbf{v}_i^T\mathbf{v}_i + \mathbf{u}_i^T\mathbf{u}_i - d_i^2) - \mathbf{u}_i^T\mathbf{C}\mathbf{v}_i$$

Due to its lack of singularities, we choose the (unit) quaternion to represent orientation. It is defined as $\bar{q} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T = \begin{bmatrix} \mathbf{q}^T & q_4 \end{bmatrix}^T$, and related to the rotational matrix by

$$\mathbf{C}(\bar{q}) = \mathbf{I}_3 - 2q_4\lfloor \mathbf{q} \times \rfloor + 2\lfloor \mathbf{q} \times \rfloor^2 \tag{3}$$

where $\lfloor \mathbf{a} \times \rfloor$ is the skew-symmetric cross-product matrix of a vector $\mathbf{a}$. Since the quaternions $\bar{q}$ and $-\bar{q}$ both represent the same rotation, the number of solutions is doubled. One can easily eliminate the spurious solutions by discarding those

with $q_4 < 0$. When using the unit quaternion representation, we need to add the unit-norm constraint

$$\bar{q}^T \bar{q} - 1 = 0 \tag{4}$$

to the polynomial system of (1) and (2).

A. Normalization of $\mathbf{p}$

In order to guarantee a bounded norm of the system's solution, in addition to using the unit-norm quaternion, we introduce the normalized translation

$$\bar{\mathbf{p}} = \mathbf{p}/||\mathbf{p}||_2 = \mathbf{p}/d_1 \tag{5}$$

This normalization can easily be realized by normalizing the position coordinates $\bar{\mathbf{u}}_i := \mathbf{u}_i/d_1$, $\bar{\mathbf{v}}_i := \mathbf{v}_i/d_1$, and distances, $\bar{d}_i := d_i/d_1$. Then, the system (1) and (2) becomes

$$\bar{\mathbf{p}}^T \bar{\mathbf{p}} - 1 = 0 \tag{6}$$

$$-\bar{\mathbf{u}}_i^T \bar{\mathbf{p}} + \bar{\mathbf{v}}_i^T \mathbf{C}^T \bar{\mathbf{p}} + \bar{f}_i = 0 \quad i = 2, \ldots, 6 \tag{7}$$

$$\text{with } \bar{f}_i = \frac{1}{2}(1 + \bar{\mathbf{v}}_i^T \bar{\mathbf{v}}_i + \bar{\mathbf{u}}_i^T \bar{\mathbf{u}}_i - \bar{d}_i^2) - \bar{\mathbf{u}}_i^T \mathbf{C} \bar{\mathbf{v}}_i \tag{8}$$

From the solution of this normalized system, we can easily recover the actual translation from $\mathbf{p} = d_1 \bar{\mathbf{p}}$.

B. Increasing speed by prior elimination of $\bar{\mathbf{p}}$

Since the complexity and hence the speed of polynomial solvers depend heavily on the number of unknowns, we reduce the number of variables by eliminating the translation, $\bar{\mathbf{p}}$. As a result, we obtain a new system of polynomials only in the elements of the quaternion $\bar{q}$. To this end, we first define the rotated normalized translation vector,

$$\bar{\mathbf{r}} := \mathbf{C}^T \bar{\mathbf{p}} \tag{9}$$

which allows us to write (7) as

$$\begin{bmatrix} \bar{\mathbf{v}}_i^T & -\bar{\mathbf{u}}_i^T & \bar{f}_i \end{bmatrix} \begin{bmatrix} \bar{\mathbf{r}} \\ \bar{\mathbf{p}} \\ 1 \end{bmatrix} = 0, \quad i = 2, \ldots, 6 \tag{10}$$

We also obtain the following constraints from (9)

$$(q_4 \mathbf{I}_3 - \lfloor \mathbf{q} \times \rfloor)\bar{\mathbf{r}} - (q_4 \mathbf{I}_3 + \lfloor \mathbf{q} \times \rfloor)\bar{\mathbf{p}} = \mathbf{0} \tag{11}$$

$$\mathbf{q}^T \bar{\mathbf{r}} - \mathbf{q}^T \bar{\mathbf{p}} = 0 \tag{12}$$

where (11) results from substituting (3) in (9) and multiplying by $(q_4 \mathbf{I}_3 - \lfloor \mathbf{q} \times \rfloor)$ (using $\lfloor \mathbf{q} \times \rfloor^3 = -\mathbf{q}^T \mathbf{q} \lfloor \mathbf{q} \times \rfloor$) while we arrive at (12) by noting that $\mathbf{q}$ is the unit eigenvector of $\mathbf{C}$ with corresponding eigenvalue 1 and pre-multiplying both sides of (9) with $\mathbf{q}^T$.

By stacking these equations, we obtain

$$\underbrace{\begin{bmatrix} \bar{\mathbf{v}}_2^T & -\bar{\mathbf{u}}_2^T & \bar{f}_2 \\ \vdots & \vdots & \vdots \\ \bar{\mathbf{v}}_6^T & -\bar{\mathbf{u}}_6^T & \bar{f}_6 \\ (q_4 \mathbf{I}_3 - \lfloor \mathbf{q} \times \rfloor) & -(q_4 \mathbf{I}_3 + \lfloor \mathbf{q} \times \rfloor) & \mathbf{0} \\ \mathbf{q}^T & -\mathbf{q}^T & 0 \end{bmatrix}}_{\boldsymbol{\Xi}} \begin{bmatrix} \bar{\mathbf{r}} \\ \bar{\mathbf{p}} \\ 1 \end{bmatrix} = \mathbf{0} \tag{13}$$

In order for (13) to have a solution, the matrix $\boldsymbol{\Xi}$ must be rank deficient, or the determinants of its (7×7)-submatrices must vanish. These determinants are polynomials in the elements of $\bar{q}$ only. Out of the 36 possibilities, we define the following five 4-th order polynomials (using Matlab notation)

$$\theta_1(\bar{q}) = \det\left(\boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 6 & 7 \end{bmatrix}, :\right)\right) \tag{14}$$

$$\theta_2(\bar{q}) = \det\left(\boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 6 & 8 \end{bmatrix}, :\right)\right) \tag{15}$$

$$\theta_3(\bar{q}) = \det\left(\boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 7 & 8 \end{bmatrix}, :\right)\right) \tag{16}$$

$$\theta_4(\bar{q}) = \det\left(\boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 6 & 9 \end{bmatrix}, :\right)\right) \tag{17}$$

$$\theta_5(\bar{q}) = \det\left(\boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 7 & 9 \end{bmatrix}, :\right)\right) \tag{18}$$

Notice that the $\theta_i(\bar{q})$ have not yet used the information in (6). To this end, we solve (13) for $\bar{\mathbf{p}}$ symbolically, using Cramer's rule. From the submatrix $\widetilde{\boldsymbol{\Xi}}_1 = \boldsymbol{\Xi}\left(\begin{bmatrix} 1:6 \end{bmatrix}, :\right)$ consisting of the first six rows of $\boldsymbol{\Xi}$, we obtain a first candidate

$$\bar{\mathbf{p}}_1 = -\frac{1}{p_{d,1}} \begin{bmatrix} p_{n_x,1} \\ p_{n_y,1} \\ p_{n_z,1} \end{bmatrix} \tag{19}$$

where

$$p_{n_x,1} = \det\left(\widetilde{\boldsymbol{\Xi}}_1\left(:, \begin{bmatrix} 1:3 & 7 & 5 & 6 \end{bmatrix}\right)\right) \tag{20}$$

$$p_{n_y,1} = \det\left(\widetilde{\boldsymbol{\Xi}}_1\left(:, \begin{bmatrix} 1:3 & 4 & 7 & 6 \end{bmatrix}\right)\right) \tag{21}$$

$$p_{n_z,1} = \det\left(\widetilde{\boldsymbol{\Xi}}_1\left(:, \begin{bmatrix} 1:3 & 4 & 5 & 7 \end{bmatrix}\right)\right) \tag{22}$$

$$p_{d,1} = \det\left(\widetilde{\boldsymbol{\Xi}}_1\left(:, \begin{bmatrix} 1:6 \end{bmatrix}\right)\right) \tag{23}$$

Analogously, we obtain three additional translation candidates, $\bar{\mathbf{p}}_2, \bar{\mathbf{p}}_3, \bar{\mathbf{p}}_4$ from the submatrices

$$\widetilde{\boldsymbol{\Xi}}_2 = \boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 7 \end{bmatrix}, :\right) \tag{24}$$

$$\widetilde{\boldsymbol{\Xi}}_3 = \boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 8 \end{bmatrix}, :\right) \tag{25}$$

$$\widetilde{\boldsymbol{\Xi}}_4 = \boldsymbol{\Xi}\left(\begin{bmatrix} 1:5 & 9 \end{bmatrix}, :\right) \tag{26}$$

Notice that the denominators and the three numerators of each candidate are polynomials in $\bar{q}$.

Direct substitution of each translation candidate $\bar{\mathbf{p}}_i$ in (6) yields four 6-th order polynomials in $\bar{q}$, $i = 1, \ldots, 4$

$$\phi_i(\bar{q}) = p_{n_x,i}^2 + p_{n_y,i}^2 + p_{n_z,i}^2 - p_{d,i}^2 = 0 \tag{27}$$

The new polynomial system resulting from the original equations (6), (7) after eliminating the translation $\bar{\mathbf{p}}$, is

$$\left.\begin{array}{c} \theta_i(\bar{q}) = 0, \ i = 1, \ldots, 5 \\ \phi_i(\bar{q}) = 0, \ i = 1, \ldots, 4 \\ \bar{q}^T \bar{q} - 1 = 0 \end{array}\right\} \tag{28}$$

Once we have obtained a solution for $\bar{q}$, we can recover the translation by evaluating (20)-(23) at that solution and back-substituting the resulting values in (19).

IV. BACKGROUND

Before presenting the algorithms to solve the 3DD problem, we briefly introduce necessary notation and sketch the theoretical basis of the multiplication matrix techniques for polynomial system solving. We recommend [18] and [14] for a detailed reference on this subject. Intuitively, multiplication matrices are the generalization of companion matrices (for solving univariate polynomials) to systems of polynomials in multiple variables.

Let a *monomial* in n variables be denoted by $\mathbf{x}^{\boldsymbol{\alpha}} := x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$, $\alpha_i \in \mathbb{Z}_{\geq 0}$, and a *polynomial* in n variables with complex coefficients correspondingly by $\psi := \sum_j c_j \mathbf{x}^{\boldsymbol{\alpha}_j}$, $c_j \in \mathbb{C}$. The ring of all polynomials in n variables with complex coefficients will be denoted as $\mathbb{C}\left[x_1, \dots, x_n\right]$. The *total degree* of a monomial is defined as $\sum_{i=1}^{n} \alpha_i$, and the total degree of a polynomial is the maximum total degree of all its monomials. In order to represent a polynomial ψ in vector notation, let ℓ denote the total degree of ψ, and stack all monomials up to and including total degree ℓ in a vector $\mathbf{x}_\ell := \begin{bmatrix} x_1^\ell & x_1^{\ell-1} x_2 & \dots & x_n & 1 \end{bmatrix}^T$. We can then write $\psi = \mathbf{c}^T \mathbf{x}_\ell$ where $\mathbf{c}$ is a potentially very sparse coefficient vector. For a polynomial system, $\psi_1 = 0, \dots, \psi_u = 0$, we can analogously define ℓ as the maximum total degree of all ψ_i, and by stacking the corresponding coefficient vectors $\mathbf{c}_i$ form the matrix system $\mathbf{C}\mathbf{x}_\ell = \mathbf{0}$. Let $\mathbf{x}_\ell(\mathbf{p})$ denote the vector of monomials evaluated at a point $\mathbf{p} \in \mathbb{C}^n$. Solving the system then means finding all points $\mathbf{p}$ that fulfill $\mathbf{C}\mathbf{x}_\ell(\mathbf{p}) = \mathbf{0}$.

Next, we define the *ideal* $I := \langle \psi_1, \dots, \psi_u \rangle = \{\sum_i h_i \psi_i, h_i \in \mathbb{C}\left[x_1, \dots, x_n\right]\}$, i.e., the set of all linear combinations of the original system multiplied with arbitrary polynomials. Each element of the ideal will equal zero when evaluated at a solution of the original system. Solving the system involves finding specific new elements of the ideal, which can be generated by (i) multiplication of an existing polynomial ψ_i with a monomial (which essentially shifts the coefficients inside $\mathbf{c}_i$), and (ii) linear combinations of polynomials, i.e., multiplication from the left of $\mathbf{C}\mathbf{x}_\ell$ by arbitrary matrices of full column rank. Notice in particular, that diagonal scaling does not change the ideal. We therefore scale each row of $\mathbf{C}$ to unit norm, which is a crucial prerequisite for numerical accuracy of our algorithms. New members of the ideal can be added to the system as additional rows in $\mathbf{C}$.

Let the set of polynomials $G = \{\gamma_1, \dots, \gamma_t\}$ be a *Gröbner basis* of I. We can define *division* by G by writing any polynomial $f \in \mathbb{C}\left[x_1, \dots, x_n\right]$ as $f = \sum_{k=1}^{t} h_k \gamma_k + r$, where $h_k, r \in \mathbb{C}\left[x_1, \dots, x_n\right]$, and no monomial in r is divisible by the leading term of any $\gamma_k \in G$. r is called the *remainder* of f on division by G, or $r = \overline{f}^G$. The special properties of Gröbner bases ensure that (i) $\overline{f}^G$ is unique, and (ii) $\overline{f}^G = 0 \Leftrightarrow f \in I$.

Our solution methods rely on the special structure of the so-called *quotient-ring* $A := \mathbb{C}\left[x_1, \dots, x_n\right]/I$, defined as the set of all remainders under division by G. In what follows, we assume that the polynomial system has a finite number s of discrete solutions[1]. Based on the Finiteness Theorem [18, p.39], A is then a finite dimensional vector space with exactly s dimensions over $\mathbb{C}$, and we can choose a monomial basis $B = \{\mathbf{x}^{\boldsymbol{\beta}_1}, \dots, \mathbf{x}^{\boldsymbol{\beta}_s}\}$, e.g., the standard monomials or *normal set* obtained from a Gröbner basis [18, p.38]. Importantly, this normal set usually remains constant for different numerical instances of the same problem, and therefore needs to be computed only once. Denote $\mathbf{x}_B$ as the vector of basis monomials $\mathbf{x}_B = \begin{bmatrix} \mathbf{x}^{\boldsymbol{\beta}_1} & \dots & \mathbf{x}^{\boldsymbol{\beta}_s} \end{bmatrix}^T$. Then, any remainder r can be written as $r = \mathbf{c}_r^T \mathbf{x}_B$.

Within the quotient ring we can define a multiplication map, which can be represented by a *multiplication matrix*.

Proposition 1: [18, Prop. 4.1, p. 56] Let $f \in \mathbb{C}\left[x_1, \dots, x_n\right]$, and $A = \mathbb{C}\left[x_1, \dots, x_n\right]/I$. Then we define the mapping $\mathbf{m}_f : A \to A$ by the rule: if $\overline{g}^G \in A$, then $\mathbf{m}_f(\overline{g}^G) := \overline{f \cdot g}^G \in A$. The map $\mathbf{m}_f(.)$ is a linear map from A to A. Furthermore, if $r' = \mathbf{m}_f(r)$, and $r = \mathbf{c}_r^T \mathbf{x}_B$, $r' = \mathbf{c}_{r'}^T \mathbf{x}_B$, then $\mathbf{c}_{r'} = \mathbf{M}_f \mathbf{c}_r$, where $\mathbf{M}_f$ is an $s \times s$ matrix, called the *multiplication matrix*.

The following two properties of the multiplication matrix are the key to solving the system of polynomials.

Proposition 2: Let the polynomial system have a finite number s of discrete solutions. Further, $f \in \mathbb{C}\left[x_1, \dots, x_n\right]$ is chosen such that the values $f(\mathbf{p}_j)$ evaluated at a solution $\mathbf{p}_j$, $j = 1, \dots, s$ are distinct. Then

1) Eigenvalues [18, Thm 4.5, p.59]: The eigenvalues of the multiplication matrix $\mathbf{M}_f$ are equal to the function f evaluated at each solution. In particular, the eigenvalues of the multiplication matrix $\mathbf{M}_{x_i}$ on A coincide with the x_i-coordinates of the solutions.
2) Eigenvectors [18, Prop. 4.7, p.64]: The left eigenvectors of the matrix $\mathbf{M}_f$ are given by the row vectors $\begin{bmatrix} \mathbf{p}_j^{\boldsymbol{\beta}_1} & \dots & \mathbf{p}_j^{\boldsymbol{\beta}_s} \end{bmatrix}$ for all solutions $\mathbf{p}_j$, $j = 1, \dots, s$.

Assuming a normal set $\mathbf{x}_B$ is known, the problem of solving a polynomial system has now been reduced to finding a multiplication matrix $\mathbf{M}_{x_i}$, for which we need to determine the column vectors $\mathbf{M}_{x_i}(:, j)$ as the coefficients of the remainders $\overline{x_i \mathbf{x}^{\boldsymbol{\beta}_j}}^G$, $j = 1, \dots, s$. The key observation [21] is that

$$x_i \mathbf{x}^{\boldsymbol{\beta}_j} = \sum h_k \gamma_k + \mathbf{M}_{x_i}(:, j)^T \mathbf{x}_B \tag{29}$$

$$\Leftrightarrow x_i \mathbf{x}^{\boldsymbol{\beta}_j} - \mathbf{M}_{x_i}(:, j)^T \mathbf{x}_B = \sum h_k \gamma_k \in I \tag{30}$$

In other words, we can read off the columns of $\mathbf{M}_{x_i}$ from certain elements of the ideal. Byröd *et al.* [19] describe one potential method how these elements can be found numerically if the normal set is known, using only linear algebra, without having to compute a Gröbner basis every time. This approach has two main drawbacks: First, it requires a normal set, which has to be computed once from a Gröbner basis (using a suitable computer algebra system, e.g., Macaulay 2). The normal set depends on the monomial order of this Gröbner basis, which can be unnecessarily restrictive [22]. Further, the

[1]More precisely, we assume I is zero-dimensional, radical and does not contain 1. We refer to [18] for what to do if these relatively mild assumptions are not met.

matrix inversion involved in computing $\mathbf{M}_{x_i}$ can be poorly conditioned for a particular normal set [6], which reduces numerical accuracy.

To mitigate these issues, [6], [7] and [8] proposed to use a set of (potentially) *polynomial* basis functions resulting from coordinate transformations of the monomials. Not only can an adaptively chosen transformation improve the conditioning, it also avoids having to compute a Gröbner basis to obtain a normal set. Unfortunately, [6], [7], [8] are all lacking a proof that their proposed transformations produce indeed a valid multiplication matrix. In what follows, we provide this proof for a more general version of [6], as well as for two solution algorithms based on [7] and [8] for a specific class of polynomial systems. In Section III we apply these algorithms to the 3DD problem.

The starting point for both algorithms is an expanded version, $\mathbf{C}_e\mathbf{x}_{l+1} = \mathbf{0}$, of the original system of polynomials. The matrix $\mathbf{C}_e$ is obtained as follows: define a vector of monomials of degree up to and including an integer l as $\mathbf{x}_l = \begin{bmatrix} x_1^l & x_1^{l-1}x_2 \dots & x_n & 1 \end{bmatrix}^T$. The length of this vector is $n_l = \binom{n+l}{n}$. Similarly, define the vector of monomials of total degree up to and including $l+1$ as $\mathbf{x}_{l+1}$ with length n_{l+1}. Finally, define the vector of monomials of total degree exactly $l+1$ as $\mathbf{x}_k$ with length $n_k = \binom{n+l}{n-1} = \binom{n+l+1}{n} - \binom{n+l}{n}$. Let ℓ_i be the total degree of ψ_i. Multiply ψ_i by each monomial of total degree up to and including $l+1-\ell_i$, and add the resulting polynomials to the system. Repeat this process for all polynomials $\psi_1, \dots, \psi_u$, and write the resulting extended system of m polynomials as

$$\mathbf{C}_e\mathbf{x}_{l+1} = \begin{bmatrix} \mathbf{C}_k & \mathbf{C}_l \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \mathbf{x}_l \end{bmatrix} = \mathbf{0} \tag{31}$$

so that $\mathbf{C}_k$ and $\mathbf{C}_l$ have dimensions $(m \times n_k)$ and $(m \times n_l)$. Notice that this process does not require any algebraic computations, but merely assigning the coefficients of the original polynomials to specific elements of $\mathbf{C}_e$. This matrix can be very large, but is usually extremely sparse (e.g., 0.5-5% nonzero elements for instances of 3DD, depending on the parameterization).

We will further need the *unreduced multiplication matrix* $\mathbf{M}'_{x_i}$ of dimension $(n_{l+1} \times n_l)$ that maps each monomial in $\mathbf{x}_l$ to $x_i \cdot \mathbf{x}_l$. In particular, $\mathbf{M}'_{x_i}(j,k) = 1$ if $\mathbf{x}_{l+1}(j) = x_i \cdot \mathbf{x}_l(k)$. As a consequence of this construction, we have

$$\mathbf{M}'^T_{x_i}\mathbf{x}_{l+1}(\mathbf{p}) = p_i\mathbf{x}_l(\mathbf{p}) \tag{32}$$

For illustration, for $n = 2$, $l = 1$, $i = 2$, we have $\mathbf{x}_1 = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix}$, $\mathbf{x}_2 = \begin{bmatrix} x_1^2 & x_1x_2 & x_2^2 & x_1 & x_2 & 1 \end{bmatrix}$, and $\mathbf{M}'_{x_2} = \begin{bmatrix} 0&1&0&0&0&0 \\ 0&0&1&0&0&0 \\ 0&0&0&0&1&0 \end{bmatrix}^T$.

V. Polynomial System Solvers

In this section, we describe 3 different polynomial system solvers, starting with the normal-set-based method, then introducing coordinate transformations to improve accuracy, and finally using the right nullspace directly to compute the multiplication matrix. We then see how these methods are related.

A. Computing the multiplication matrix using a normal set

As described in [19], if a normal set $\mathbf{x}_B$ is available for the problem at hand (e.g., computed from the Gröbner bases of example systems using a computer algebra program), the multiplication matrix is computed as follows.

First, define the monomials $\mathbf{x}_R$ as the set $x_i\mathbf{x}_B \backslash \mathbf{x}_B$, i.e., the monomials $x_i\mathbf{x}_B$ that do not belong to the normal set. These are the monomials that need to be reduced through division by the ideal. Finally, let $\mathbf{x}_E$ be the monomials in $\mathbf{x}_{l+1}$ that belong neither in $\mathbf{x}_B$ nor $\mathbf{x}_R$. By reordering the monomials and the columns of $\mathbf{C}_e$, we obtain

$$\mathbf{C}_e\mathbf{x}_{l+1} = \begin{bmatrix} \mathbf{C}_E & \mathbf{C}_R & \mathbf{C}_B \end{bmatrix} \begin{bmatrix} \mathbf{x}_E \\ \mathbf{x}_R \\ \mathbf{x}_B \end{bmatrix} = \mathbf{0} \tag{33}$$

Let $\mathbf{N}^T$ be the left nullspace of $\mathbf{C}_E$, i.e., $\mathbf{N}^T\mathbf{C}_E = \mathbf{0}$, and decompose $\mathbf{N}^T\mathbf{C}_R = \mathbf{QR} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}$ using QR factorization. Assuming that $\mathbf{R}_1$ is of full rank, we arrive at

$$\begin{bmatrix} \mathbf{N}^T\mathbf{C}_R & \mathbf{N}^T\mathbf{C}_B \end{bmatrix} \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_B \end{bmatrix} = \mathbf{Q} \begin{bmatrix} \mathbf{R}_1 & \mathbf{Q}_1^T\mathbf{N}^T\mathbf{C}_B \\ \mathbf{0} & \mathbf{Q}_2^T\mathbf{N}^T\mathbf{C}_B \end{bmatrix} \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_B \end{bmatrix} = \mathbf{0}$$

$$\Rightarrow \mathbf{Q}_1\mathbf{R}_1 \begin{bmatrix} \mathbf{I}_R & \mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_B \end{bmatrix} = \mathbf{0}, \quad \text{where } \mathbf{T} = \mathbf{R}_1^{-1}\mathbf{Q}_1^T\mathbf{N}^T\mathbf{C}_B$$

In a slight deviation from our previous notation, let the unreduced multiplication matrix $\mathbf{M}'_{x_i}$ correspond to the map $\mathbf{M}'_{x_i} : \mathbf{x}_B \to \begin{bmatrix} \mathbf{x}_R \\ \mathbf{x}_B \end{bmatrix}$ induced by multiplication with x_i. Then, the multiplication matrix becomes [19]

$$\mathbf{M}_{x_i} = \begin{bmatrix} -\mathbf{T}^T & \mathbf{I}_s \end{bmatrix} \mathbf{M}'_{x_i} \tag{34}$$

This method has two drawbacks: first, it requires determining the normal set, and second, the inverse in the expression for $\mathbf{T}$ can be very ill-conditioned.

B. Using a general coordinate transformation

In [6], a specific coordinate transformation was proposed to improve conditioning of this inverse. In what follows, we prove that a very general form of coordinate transformation indeed yields a valid multiplication matrix, of which the methods presented in [6], [7], [8] are special cases.

The involutive test described in [8] provides a means to determine the degree to which the system needs to be expanded or "prolonged" (cf. (31)). Passing this test also eliminates the need to determine a normal set. The 3DD polynomial system (28) passed the involutive test after prolongation or expansion to total degree $l+1 = 9$, and no projection. Another condition implied by this specific outcome of the involutive test is that $\mathbf{C}_k$ is of full rank, and that $\mathbf{C}_e$ is of rank $n_{l+1} - s$, i.e., it has an s-dimensional nullspace. These important conditions ensure that the rank conditions needed in the derivations below are fulfilled.

To start, we first decompose $\mathbf{C}_k$ (cf. (31)), such that

$$\mathbf{C}_k = \mathbf{Q}_1\mathbf{R}_1\mathbf{E}_1^T = \mathbf{Q}_1 \begin{bmatrix} \mathbf{R}_{11} \\ \mathbf{0} \end{bmatrix} \mathbf{E}_1^T \tag{35}$$

where $\mathbf{E}_1$ is an $(n_k \times n_k)$ column permutation matrix, i.e., $(\mathbf{E}_1)^{-1} = \mathbf{E}_1^T$, and $\mathbf{R}_{11}$ is upper triangular.

We now split $\mathbf{Q}_1^T\mathbf{C}_l = \begin{bmatrix}\mathbf{D}_{l1}^T & \mathbf{D}_{l2}^T\end{bmatrix}^T$, so that $\mathbf{D}_{l1}$ is of dimension $(n_k \times n_l)$ and we can write

$$\mathbf{C}_e = \mathbf{Q}_1 \begin{bmatrix}\mathbf{R}_{11} & \mathbf{D}_{l1} \\ \mathbf{0} & \mathbf{D}_{l2}\end{bmatrix} \begin{bmatrix}\mathbf{E}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n_l}\end{bmatrix} \tag{36}$$

A second QR-decomposition yields

$$\mathbf{D}_{l2} = \mathbf{Q}_2\mathbf{R}_2\mathbf{E}_2^T = \mathbf{Q}_2 \begin{bmatrix}\mathbf{R}_{21} & \mathbf{R}_{22} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} \mathbf{E}_2^T \tag{37}$$

where $\mathbf{R}_{21}$ is upper-triangular, $\mathbf{R}_{22}$ is of dimension $(n_l - s) \times s$, and $\mathbf{E}_2$ is an $(n_l \times n_l)$ permutation matrix (which is required, since $\mathbf{D}_{l2}$ is rank-deficient). We can further split $\mathbf{E}_2 = \begin{bmatrix}\mathbf{E}_{21} & \mathbf{E}_{22}\end{bmatrix}$, such that $\mathbf{E}_{22}$ is of dimension $(n_l \times s)$.

We now have

$$\begin{aligned}\mathbf{C}_e &= \mathbf{Q}_1 \begin{bmatrix}\mathbf{I}_{n_k} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2\end{bmatrix} \begin{bmatrix}\mathbf{R}_{11} & \mathbf{D}_{l1}\mathbf{E}_{21} & \mathbf{D}_{l1}\mathbf{E}_{22} \\ \mathbf{0} & \mathbf{R}_{21} & \mathbf{R}_{22} \\ \mathbf{0} & \mathbf{0} & \mathbf{0}\end{bmatrix} \begin{bmatrix}\mathbf{E}_1^T & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_2^T\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{Q}_1' & \mathbf{Q}_2'\end{bmatrix} \begin{bmatrix}\mathbf{R}' \\ \mathbf{0}\end{bmatrix} \mathbf{E}^T \end{aligned} \tag{38}$$

where $\mathbf{R}'$ is of dimension $(n_{l+1} - s \times n_{l+1})$ and upper triangular, and $\mathbf{E} = \left[\begin{smallmatrix}\mathbf{E}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{E}_2\end{smallmatrix}\right]$. Further, partition $\mathbf{R}' = \begin{bmatrix}\mathbf{R}_k' & \mathbf{R}_l'\end{bmatrix}$ so that $\mathbf{R}_k'$ is of dimension $(n_{l+1} - s \times n_k)$, and $\mathbf{R}_l'$ is of dimension $(n_{l+1} - s \times n_l)$.

We can now introduce a general, invertible coordinate transformation matrix $\mathbf{V}_e$ $(n_{l+1} \times n_{l+1})$,

$$\mathbf{V}_e = \begin{bmatrix}\mathbf{V}_{11} & \mathbf{V}_{12} \\ \mathbf{0} & \mathbf{V}_{22}\end{bmatrix} \tag{39}$$

where $\mathbf{V}_{22}$ is of dimension $(s \times n_l)$, and also define its inverse as $\mathbf{V}_e^{-1} = \begin{bmatrix}\mathbf{\Lambda}_1 & \mathbf{\Lambda}_2\end{bmatrix}$, where $\mathbf{\Lambda}_1$ is of dimension $(n_{l+1} \times n_{l+1} - s)$, and $\mathbf{\Lambda}_2$ is of dimension $(n_{l+1} \times s)$.

Using this coordinate transformation and (38) in (31) yields

$$\mathbf{Q}_1'\mathbf{R}'\mathbf{V}_e^{-1}\mathbf{V}_e\mathbf{E}^T\mathbf{x}_{l+1} = \mathbf{0} \tag{40}$$

$$\mathbf{Q}_1'\mathbf{R}'\mathbf{\Lambda}_1 \begin{bmatrix}\mathbf{I}_{n_{l+1}-s} & \mathbf{T}\end{bmatrix} \begin{bmatrix}\mathbf{V}_{11}\mathbf{E}_1^T & \mathbf{V}_{12}\mathbf{E}_2^T \\ \mathbf{0} & \mathbf{V}_{22}\mathbf{E}_2^T\end{bmatrix} \mathbf{x}_{l+1} = \mathbf{0} \tag{41}$$

$$\Leftrightarrow \begin{bmatrix}\mathbf{V}_{11}\mathbf{E}_1^T & \mathbf{V}_{12}\mathbf{E}_2^T\end{bmatrix} \mathbf{x}_{l+1} = -\mathbf{T} \begin{bmatrix}\mathbf{0} & \mathbf{V}_{22}\mathbf{E}_2^T\end{bmatrix} \mathbf{x}_{l+1} \tag{42}$$

where $\mathbf{T} = (\mathbf{R}'\mathbf{\Lambda}_1)^{-1}\mathbf{R}'\mathbf{\Lambda}_2$. Notice that the introduction of the coordinate transformation has given us leverage to improve the conditioning of the inversion of $(\mathbf{R}'\mathbf{\Lambda}_k)$ when computing $\mathbf{T}$ through judicious choice of $\mathbf{\Lambda}_k$. Equation (42) means that we can now express every polynomial in $\begin{bmatrix}\mathbf{V}_{11}\mathbf{E}_1^T & \mathbf{V}_{12}\mathbf{E}_2^T\end{bmatrix} \mathbf{x}_{l+1}$ as a linear combination of monomials in $\mathbf{x}_B = \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{x}_l$, which we will choose as a basis of the quotient ring.

For this basis, we will now prove that the multiplication matrix can be formed as

$$\mathbf{M}_{x_i} = \begin{bmatrix}-\mathbf{T}^T & \mathbf{I}_s\end{bmatrix} \mathbf{V}_e^{-T}\mathbf{E}^T\mathbf{M}_{x_i}'\mathbf{E}_2\mathbf{V}_{22}^T \tag{43}$$

using the unreduced multiplication matrix $\mathbf{M}_{x_i}'$ defined in Section IV.

Proposition 3: If we choose the basis as $\mathbf{x}_B = \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{x}_l$, and compute the multiplication matrix as in (43), then $\mathbf{x}_B(\mathbf{p}_j)$ is a right eigenvector of $\mathbf{M}_{x_i}^T$.

Proof:

$$\mathbf{M}_{x_i}^T\mathbf{x}_B(\mathbf{p}_j) \overset{(43)}{=} \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{M}_{x_i}'^T\mathbf{E}\mathbf{V}_e^{-1} \begin{bmatrix}-\mathbf{T} \\ \mathbf{I}_s\end{bmatrix} \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{x}_l(\mathbf{p}_j)$$

$$= \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{M}_{x_i}'^T\mathbf{E}\mathbf{V}_e^{-1} \begin{bmatrix}-\mathbf{T}\begin{bmatrix}\mathbf{0} & \mathbf{V}_{22}\mathbf{E}_2^T\end{bmatrix} \\ \begin{bmatrix}\mathbf{0} & \mathbf{V}_{22}\mathbf{E}_2^T\end{bmatrix}\end{bmatrix} \mathbf{x}_{l+1}(\mathbf{p}_j) \tag{44}$$

$$\overset{(42)}{=} \mathbf{V}_{22}\mathbf{E}_2^T\mathbf{M}_{x_i}'^T\mathbf{E}\mathbf{V}_e^{-1}\mathbf{V}_e\mathbf{E}^T\mathbf{x}_{l+1}(\mathbf{p}_j) \tag{45}$$

$$\overset{(32)}{=} \mathbf{V}_{22}\mathbf{E}_2^T p_i\mathbf{x}_l(\mathbf{p}_j) = p_i\mathbf{x}_B(\mathbf{p}_j) \tag{46}$$

■

Finally, we need to compute the solutions $\mathbf{p}_j$ from the eigenvectors $\mathbf{v}_j := c\mathbf{V}_{22}\mathbf{E}_2^T\mathbf{x}_l(\mathbf{p}_j)$ where c is some unknown scaling coefficient (resulting, e.g., from restricting the eigenvectors to unit norm). We define $\mathbf{v}_j' := \mathbf{E}\mathbf{V}_e^{-1} \begin{bmatrix}-\mathbf{T} \\ \mathbf{I}_s\end{bmatrix} \mathbf{v}_j$, knowing from the proof that $\mathbf{v}_j' = c\mathbf{x}_{l+1}(\mathbf{p}_j)$. Noting that the last $n+1$ elements of $\mathbf{x}_{l+1}$ are given by $\begin{bmatrix}x_1 & \dots & x_n & 1\end{bmatrix}$, we undo the scaling and recover the solution as

$$\mathbf{p}_j = \mathbf{v}_j'(n_{l+1} - n : n_{l+1} - 1)/\mathbf{v}_j'(n_{l+1}) \tag{47}$$

Obviously, only the last $n+1$ elements of the vector $\mathbf{v}_j'$ will actually need to be computed.

C. Using QR with column pivoting

A trivial choice for the coordinate transformation is $\mathbf{V}_e = \mathbf{V}_e^{-1} = \mathbf{I}$, which is precisely the case presented in [7]. In this case, the basis becomes $\mathbf{x}_B = \begin{bmatrix}\mathbf{0} & \mathbf{I}_s\end{bmatrix} \mathbf{E}_2^T\mathbf{x}_l = \mathbf{E}_{22}^T\mathbf{x}_l$. Notice that this algorithm only requires two QR factorizations, and computing $\mathbf{T}$ reduces to solving an upper triangular system.

Contrary to [6], [7], this paper provides more flexibility in the choice of $\mathbf{V}_e$, the proof that it actually yields a valid multiplication matrix, as well as guidance for the degree to which to expand (via the involutive test [8]). The appeal of this method is further that it requires no normal set and can be implemented purely using standard linear algebra tools.

D. Using the right nullspace

The following method [8] proposed by Reid and Zhi (which we will call R.Z. method for short), uses a different paradigm to find the multiplication matrix. The key observation is that if all solutions are distinct, and the dimension of the right nullspace of $\mathbf{C}_e$ equals the number of solutions, s, then the vectors $\mathbf{x}_{l+1}(\mathbf{p}_j)$ evaluated at the solutions span the right nullspace. In other words, if $\mathbf{B}_{l+1}$ is any matrix whose columns span the nullspace of $\mathbf{C}_e$, then $\mathbf{x}_{l+1}(\mathbf{p}_j) = \mathbf{B}_{l+1}\mathbf{c}_j$ for some coefficient vector $\mathbf{c}_j$.

Let $\mathbf{B}_{l+1} = \begin{bmatrix}\mathbf{B}_k \\ \mathbf{B}_l\end{bmatrix}$ be the right nullspace of $\mathbf{C}_e$ so that $\mathbf{B}_l$ is of dimension $(n_l \times s)$. Notice that $\mathbf{x}_{l+1}(\mathbf{p}_j) = \mathbf{B}_{l+1}\mathbf{c}_j$ and $\mathbf{x}_l(\mathbf{p}_j) = \begin{bmatrix}\mathbf{0} & \mathbf{I}\end{bmatrix} \mathbf{x}_{l+1}(\mathbf{p}_j) = \mathbf{B}_l\mathbf{c}_j$.

Form the SVD of $\mathbf{B}_l$

$$\mathbf{B}_l = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{bmatrix} \begin{bmatrix}\mathbf{S} \\ \mathbf{0}\end{bmatrix} \mathbf{V}^T \tag{48}$$

such that $\mathbf{U}_1$ is of dimension $(n_l \times s)$, and $\mathbf{S}$ is square. Moreover, based on the involutive test, $\mathbf{B}_l$ is of rank s, i.e., $\mathbf{S}$ is invertible.

We choose $\mathbf{x}_B = \mathbf{U}_1^T \mathbf{x}_l$ as basis for the quotient ring, and form the multiplication matrix as [8]

$$\mathbf{M}_{x_i} = \mathbf{S}^{-1}\mathbf{V}^T \mathbf{B}_{l+1}^T \mathbf{M}'^T_{x_i} \mathbf{U}_1 \tag{49}$$

Since $\mathbf{S}$ is diagonal, the computation of its inverse is trivial.

Proposition 4: If we choose the basis of the quotient ring as $\mathbf{x}_B = \mathbf{U}_1^T \mathbf{x}_l$, and compute the multiplication matrix as in (49), then $\mathbf{x}_B(\mathbf{p}_j)$ is a right eigenvector of $\mathbf{M}_{x_i}^T$.

Proof:

$$\begin{aligned}
\mathbf{M}_{x_i}^T \mathbf{x}_B(\mathbf{p}_j) &\overset{(49)}{=} \mathbf{U}_1^T \mathbf{M}'^T_{x_i} \mathbf{B}_{l+1} \mathbf{V} \mathbf{S}^{-1} \mathbf{U}_1^T \mathbf{B}_l \mathbf{c}_j \\
&\overset{(48)}{=} \mathbf{U}_1^T \mathbf{M}'^T_{x_i} \mathbf{B}_{l+1} \mathbf{c}_j \\
&= \mathbf{U}_1^T \mathbf{M}'^T_{x_i} \mathbf{x}_{l+1}(\mathbf{p}_j) \\
&\overset{(32)}{=} p_i \mathbf{U}_1^T \mathbf{x}_l(\mathbf{p}_j)
\end{aligned}$$

■

We can recover the solution $\mathbf{p}_j$ from an eigenvector $\mathbf{v}_j = c\mathbf{U}_1^T \mathbf{x}_l(\mathbf{p}_j)$ of $\mathbf{M}_{x_i}^T$ by writing

$$\begin{aligned}
\mathbf{v}'_j &= \mathbf{U}_1 \mathbf{v}_j = c\mathbf{U}_1 \mathbf{U}_1^T \mathbf{x}_l(\mathbf{p}_j) \\
&= c\mathbf{U}_1 \mathbf{U}_1^T \mathbf{B}_l \mathbf{c}_j = c\mathbf{U}_1 \mathbf{U}_1^T \mathbf{U}_1 \mathbf{S} \mathbf{V}^T \mathbf{c}_j \\
&= c\mathbf{B}_l \mathbf{c}_j = c\mathbf{x}_l(\mathbf{p}_j)
\end{aligned}$$

and by undoing the scaling as before.

Interestingly, as we will show, the R.Z. method can be cast as a special instance of the general transformation described in Section V-B. For this we proceed as follows (cf. (40)):

$$\mathbf{Q}'_1 \mathbf{R}' \mathbf{V}_e^{-1} \mathbf{V}_e \mathbf{E}^T \mathbf{x}_{l+1} = \mathbf{0} \tag{50}$$

$$\mathbf{Q}'_1 \begin{bmatrix} \mathbf{I}_{n_{l+1}-s} & \mathbf{T} \end{bmatrix} \begin{bmatrix} \mathbf{V}_{11}\mathbf{E}_1^T & \mathbf{V}_{12}\mathbf{E}_2^T \\ \mathbf{0} & \mathbf{V}_{22}\mathbf{E}_2^T \end{bmatrix} \mathbf{x}_{l+1} = \mathbf{0} \tag{51}$$

where we have (arbitrarily) imposed the structure

$$\begin{bmatrix} \mathbf{R}'_k & \mathbf{R}'_l \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{n_{l+1}-s} & \mathbf{T} \end{bmatrix} \mathbf{V}_e = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{V}_{12} + \mathbf{T}\mathbf{V}_{22} \end{bmatrix} \tag{52}$$

To make the correspondence to R.Z., we require that $\mathbf{V}_{22}\mathbf{E}_2^T = \mathbf{U}_1^T$. Also, we choose $\mathbf{V}_{12}\mathbf{E}_2^T\mathbf{U}_1 = \mathbf{0}$.

We obtain

$$\mathbf{V}_{11} = \mathbf{R}'_k \tag{53}$$

$$\mathbf{V}_{22} = \mathbf{U}_1^T \mathbf{E}_2 \tag{54}$$

$$\mathbf{R}'_l = \mathbf{V}_{12} + \mathbf{T}\mathbf{U}_1^T \mathbf{E}_2 \qquad | \cdot \mathbf{E}_2^T \mathbf{U}_1 \tag{55}$$

$$\mathbf{T} = \mathbf{R}'_l \mathbf{E}_2^T \mathbf{U}_1 \tag{56}$$

Backsubstitution of $\mathbf{T}$ yields

$$\mathbf{R}'_l = \mathbf{V}_{12} + \mathbf{R}'_l \mathbf{E}_2^T \mathbf{U}_1 \mathbf{U}_1^T \mathbf{E}_2 \tag{57}$$

$$\mathbf{V}_{12} = \mathbf{R}'_l (\mathbf{I}_{n_l} - \mathbf{E}_2^T \mathbf{U}_1 \mathbf{U}_1^T \mathbf{E}_2) \tag{58}$$

$$= \mathbf{R}'_l \mathbf{E}_2^T (\mathbf{U}_1 \mathbf{U}_1^T + \mathbf{U}_2 \mathbf{U}_2^T - \mathbf{U}_1 \mathbf{U}_1^T) \mathbf{E}_2 \tag{59}$$

$$= \mathbf{R}'_l \mathbf{E}_2^T \mathbf{U}_2 \mathbf{U}_2^T \mathbf{E}_2 \tag{60}$$

In summary, the (not necessarily unique) choice of $\mathbf{V}_e = \begin{bmatrix} \mathbf{R}'_k & \mathbf{R}'_l \mathbf{E}_2^T \mathbf{U}_2 \mathbf{U}_2^T \mathbf{E}_2 \\ \mathbf{0} & \mathbf{U}_1^T \mathbf{E}_2 \end{bmatrix}$ will yield exactly the same multiplication matrix as the R.Z. method.

VI. Simulation Results

In this section, we evaluate the accuracy and efficiency of solving the 3DD problem using QR decomposition (QR) (cf. Section V-C), the R.Z. method (cf. Section V-D), and the normal set-based method (Trad).

The simulation has been conducted on 1000 robot trajectories which are randomly generated with the two robots being 1 m - 2 m apart, and moving an average 3 m - 6 m between taking distance measurements. We computed the solutions for these 1000 samples using the three proposed techniques, and using homotopy continuation (PHC) [12] as ground truth.

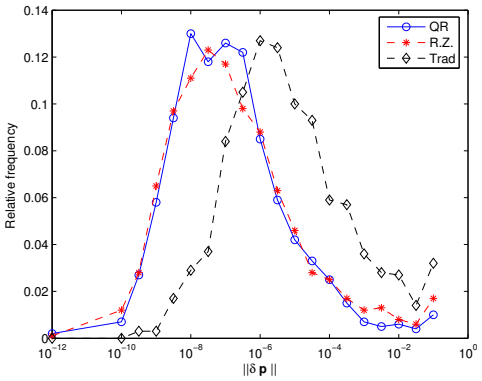
Our code was implemented in C++ on a Pentium T9400, 2.53 GHz Laptop, using inverse iterations with sparse LU factorization [23] to compute the right nullspace of $\mathbf{C}_e$ for R.Z. We obtained $\mathbf{C}_e$ of size (1100×715), with 41340 non-zero elements, of total degree $l+1 = 9$ according to the involutive test [8]. The QR method and R.Z. were run in 1.5 s and 0.37 s, respectively, while the normal set-based method runs significantly faster (0.08 s), at the cost of having to determine the normal set and potentially worse numerical conditioning. This runtime is competitive with the fastest reported time of any numerical solver for the 3DD problem.

To evaluate the accuracy of all the 40 solutions produced by the presented three algorithms, we compared our solutions to the PHC solutions. We first matched the 40 solutions of QR, R.Z., and traditional normal set method to the closest PHC solutions, and computed the average 2-norm of their errors over all 40 solutions. The error distributions of these three methods are shown in Fig. 2. The results show that the accuracy of the QR and R.Z. methods is about two orders of magnitude higher than the traditional normal set method, particularly for the orientation. In case the accuracy obtained is insufficient, the solutions of any of the three methods can easily be refined using Newton steps.

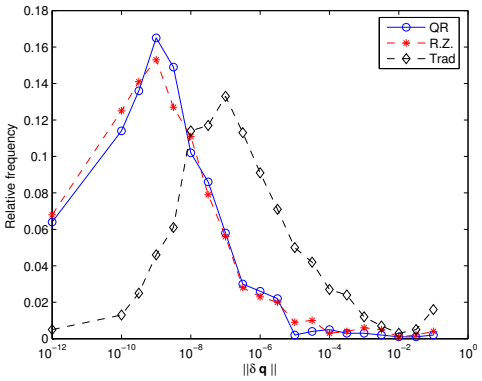
We also tried to compare the accuracy of our algorithms to that of Lee and Shim [17], which is currently the fastest reported solution to the 3DD problem, but which requires a special data type (30 significant digits) for accuracy. We implemented their method in Matlab using the symbolic toolbox to represent the coefficients symbolically for the required precision. Unfortunately, at the time of publication we were still unable to reproduce results of useable accuracy other than for problems with integer coefficients. Improving our implementation of their method is work in progress.

VII. Conclusion

In this paper, we have presented three fast algorithms to compute the forward kinematics of the general Stewart-Gough mechanism, or, equivalently, to compute the relative pose between two robots based on distance measurements. The algorithms use three different ways to compute the multiplication matrix, based on [19], [7], and [8], from whose eigenvectors we can extract the solutions. We have proven that these methods indeed produce valid multiplication matrices, and shown their relationship.



(a) Position error with respect to PHC solutions



(b) Orientation error with respect to PHC solutions

Fig. 2. Relative pose error distribution generated from the error histogram of 1000 samples. The markers on each curve indicate the centroids of each bin in the histogram, and the y-axis shows the relative frequency of samples in the bins. The QR and R.Z. methods shows significantly improved accuracy over the traditional normal set method, at the cost of increased run time.

Our simulation results have shown that we can compute the solution in less than 1.5 s, and most solutions achieve an accuracy in the order of 10^{-8}. Our methods' major advantage over currently available algorithms is their ease of implementation and the fact that they only use standard double data types and linear algebra libraries.

The hybrid algebraic-numeric techniques for solving polynomial systems used in this paper have only very recently appeared in the robotics and computer vision. Keeping in mind that many geometric problems in robotics can be converted to a system of polynomials, we believe that fast and principled polynomial system solvers such as the ones presented in this paper can be the key to solving interesting long-standing as well as new research problems.

Acknowledgements

This work was supported by the University of Minnesota (DTC), and the National Science Foundation (IIS-0643680, IIS-0811946, IIS-0835637). The authors gratefully acknowledge Ryan Elmquist for helping with code implementation.

References

[1] N. Trawny, X. S. Zhou, K. X. Zhou, and S. I. Roumeliotis, "3D relative pose estimation from distance-only measurements," in *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, Oct. 29 – Nov. 2, 2007, pp. 1071–1078.

[2] D. Stewart, "A platform with six degrees of freedom," in *Proc. of the Inst. of Mechanical Engineers*, vol. 180, no. 15, 1965, pp. 371–376.

[3] M. Raghavan, "The Stewart platform of general geometry has 40 configurations," *Journal of Mechanical Design*, vol. 115, pp. 277–282, 1993.

[4] C. W. Wampler, "Forward displacement analysis of general six-in-parallel SPS (Stewart) platform manipulators using soma coordinates," *Mechanism and Machine Theory*, vol. 31, no. 3, pp. 331–337, 1996.

[5] P. Dietmaier, "The Stewart-Gough platform of general geometry can have 40 real postures," in *Advances in Robot Kinematics: Analysis and Control*, J. Lenarcic and M. L. Husty, Eds. Kluwer Academic Publishers, 1998, pp. 7–16.

[6] M. Byröd, K. Josephson, and K. Åström, "Improving numerical accuracy of Gröbner basis polynomial equation solvers," in *Proc. of the IEEE International Conference on Computer Vision*, Rio de Janeiro, Oct. 14–20, 2007, pp. 1–8.

[7] ——, "A column-pivoting based strategy for monomial ordering in numerical Gröbner basis calculations," in *Proc. of the 10th European Conference on Computer Vision*, Marseille, France, Oct. 12–18, 2008.

[8] G. Reid and L. Zhi, "Solving polynomial systems via symbolic-numeric reduction to geometric involutive form," *Journal of Symbolic Computation*, vol. 44, no. 3, pp. 280–291, Mar. 2009.

[9] W. Auzinger and H. J. Stetter, "An elimination algorithm for the computation of all zeros of a system of multivariate polynomial equations," in *Numerical Mathematics (Singapore 1988), Int. Ser. Numer. Math.* Basel: Birkhäuser, 1988, vol. 86, pp. 11–30.

[10] P. R. McAree and R. W. Daniel, "A fast, robust solution to the Stewart platform forward kinematics," *Journal of Robotic Systems*, vol. 13, no. 7, pp. 407–427, 1996.

[11] C. W. Wampler, A. P. Morgan, and A. J. Sommese, "Numerical continuation methods for solving polynomial systems arising in kinematics," *Journal of Mechanical Design*, vol. 112, no. 1, pp. 59–68, 1990.

[12] J. Verschelde, "Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation," *ACM Transactions on Mathematical Software*, vol. 25, no. 2, pp. 251–276, 1999.

[13] L. Rolland, "Synthesis of the forward kinematics problem algebraic modeling for the general parallel manipulator: displacement-based equations," *Advanced Robotics*, vol. 21, no. 9, pp. 1071–1092, Sep. 2007.

[14] D. Cox, J. Little, and D. O'Shea, *Ideals, Varieties, and Algorithms*, 3rd ed. Springer, 2008.

[15] M. L. Husty, "An algorithm for solving the direct kinematics of general Stewart-Gough platforms," *Mechanism and Machine Theory*, vol. 31, no. 4, pp. 365–380, 1996.

[16] C. Innocenti, "Forward kinematics in polynomial form of the general Stewart platform," *Journal of Mechanical Design*, vol. 123, no. 2, pp. 254–260, 2001.

[17] T.-Y. Lee and J.-K. Shim, "Improved dialytic elimination algorithm for the forward kinematics of the general Stewart-Gough platform," *Mechanism and Machine Theory*, vol. 38, pp. 563–577, Jun. 2003.

[18] D. Cox, J. Little, and D. O'Shea, *Using Algebraic Geometry*, 2nd ed. Springer, 2005.

[19] M. Byröd, Z. Kukelova, K. Josephson, T. Pajdla, and K. Åström, "Fast and robust numerical solutions to minimal problems for cameras with radial distortion," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, Jun. 23–28, 2008, pp. 1–8.

[20] H. Stewénius, C. Engels, and D. Nistér, "Recent developments on direct relative orientation," *International Journal of Photogrammetry and Remote Sensing*, vol. 60, no. 4, pp. 284–294, Jun. 2006.

[21] Z. Kukelova and T. Pajdla, "A minimal solution to the autocalibration of radial distortion," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, Jun. 18–23, 2007, pp. 1–7.

[22] H. J. Stetter, *Numerical Polynomial Algebra*. SIAM: Society for Industrial and Applied Mathematics, 2004.

[23] C. Gotsman and S. Toledo, "On the computation of null spaces of sparse rectangular matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 30, no. 2, pp. 445–463, May 2008.