International Journal of Shape Modeling
© World Scientific Publishing Company

Optimizations for Tensorial Bernstein–Based Solvers by using Polyhedral Bounds

Christoph Fünfzig
Lab Electronics, Imaging and Informatics (UMR CNRS 5158), Université de Bourgogne, Dijon, France
christoph.fuenfzig@u-bourgogne.fr

Dominique Michelucci
Lab Electronics, Imaging and Informatics (UMR CNRS 5158), Université de Bourgogne, Dijon, France
dmichel@u-bourgogne.fr

Sebti Foufou
Lab Electronics, Imaging and Informatics (UMR CNRS 5158), Université de Bourgogne, Dijon, France
Computer Science and Engineering, Qatar University, Doha, Qatar
sfoufou@u-bourgogne.fr, sfoufou@qu.edu.qa

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

The tensorial Bernstein basis for multivariate polynomials in n variables has a number 3^n of functions for degree 2. Consequently, computing the representation of a multivariate polynomial in the tensorial Bernstein basis is an exponential time algorithm, which makes tensorial Bernstein-based solvers impractical for systems with more than $n = 6$ or 7 variables. This article describes a polytope (*Bernstein polytope*) with a number $O(b(n,2))$ of faces, which allows to bound a sparse, multivariate polynomial expressed in the canonical basis by solving several linear programming problems. We compare the performance of a subdivision solver using domain reductions by linear programming with a solver using a change to the tensorial Bernstein basis for domain reduction. The performance is similar for $n = 2$ variables but only the solver using linear programming on the Bernstein polytope can cope with a large number of variables. We demonstrate this difference with two formulations of the forward kinematics problem of a Gough-Stewart parallel robot: a direct Cartesian formulation and a coordinate-free formulation using Cayley-Menger determinants, followed by a computation of Cartesian coordinates. Furthermore, we present an optimization of the Bernstein polytope-based solver for systems containing only the monomials x_i and x_i^2. For these, it is possible to obtain even better domain bounds at no cost using the quadratic curve (x_i, x_i^2) directly.

Keywords: Bernstein polynomials, algebraic systems, subdivision solver, linear programming, simplex algorithm

1991 Mathematics Subject Classification: 14Q15, 14Q20, 65G40

1. Introduction

Bernstein bases and Bernstein solvers are used in computer graphics and CAD-CAM for computing intersection points between algebraic non-linear curves or surfaces. These solvers reduce domain boxes while retaining the solutions of a given polynomial system, and detect when it does not have solutions. Subdivision of the current domain box can separate different solutions and allows to converge to single solutions. Usually, a simple bisection is used for simplicity. In the overview, the solving process is depicted in the pseudo code of Figure 1.

```
Start with domain D on candidate list C
While (the candidate list C is not empty) {
  Take box b from C and calculate ls=max longest side of b
  do {
     perform domain reduction on b
     if (b does not contain any solution) {
        exit and continue with next candidate b
     } else { calculate ls=max longest side of b }
  } while (b has been sufficiently reduced)
  if (ls < delta) {
     b contains potentially a solution
  } else {
     bisect b along longest side into b1 and b2,
     and add them to candidate list C
  }
}
```

Fig. 1. Subdivision solver

For simplicity in this paper, we assume that the system has total degree at most 2. Each algebraic system can be rewritten as a system of total degree 2 by introducing additional variables and equations. In terms of performance, the *domain reduction* and the test for a solution are of major importance. A classic way for domain reduction is based on the representation of the multivariate polynomial in n variables in the tensorial Bernstein basis (TBB). However, the representation of a multivariate polynomial in the tensorial Bernstein basis (TBB) has exponential length: the basis for degree d has $(d+1)^n$ basis functions. It makes TBB-based solvers impractical for large systems with more than $n = 6$ or 7 variables, which arise for example in geometric constraint solving.

An alternative for domain reduction uses a polytope enclosing the nonlinear momomials on the domain. This polytope can be described by a polynomial number of halfspaces in the number n of variables. By using linear programming, we can compute a range bound for each multivariate polynomial and a domain bound of all system solutions.

We give a short survey of the representation of polynomials in the tensorial Bernstein basis in Section 2 and introduce our Bernstein polytope in the following Section 3. We collect data of numerical experiments calculating the intersection of 2D algebraic curves in Section 7. Therein, we compare with the domain reduction method proposed in ref. 1 using the tensorial Bernstein basis. This method was selected because it has the best domain reduction performance currently available. In Section 6, we give a possible optimization of the subdivision solver for squared variables x_i^2. It is possible to derive bounds for x_i directly from the curve (x_i, x_i^2), which are tighter than the ones obtained by the Bernstein polytope alone.

Concerning industrial applications, Section 8 considers the kinematics of a Gough-Stewart parallel robot in two formulations: a direct Cartesian formulation and a coordinate-free formulation using Cayley-Menger determinants, which results in a system with much less variables. We present the systems and their solution statistics with both solvers. Finally, we give conclusions in Section 9.

2. Multivariate Polynomials in the Tensorial Bernstein Basis

The $d+1$ Bernstein polynomials $B_i^{(d)}$ of degree d, also written B_i for fixed d, are a basis for polynomials of degree $\leq d$

$$B_i^{(d)}(x) = \binom{d}{i} x^i (1-x)^{d-i}$$

The conversion from and to the canonical basis $(x^0, x^1, \dots x^d)$ is a linear mapping [2]

$$\begin{aligned} x^k &= \tbinom{d}{k}^{-1} \textstyle\sum_{i=k}^{d} \tbinom{i}{k} B_i^{(d)}(x) \\ x &= d^{-1} \textstyle\sum_{i=1}^{d} i B_i^{(d)}(x) \\ 1 &= \textstyle\sum_{i=0}^{d} B_i^{(d)}(x) \end{aligned}$$

Every Bernstein polynomial $B_i^{(d)}(x)$ is non-negative for $x \in [0,1]$, and the sum of all Bernstein polynomials equals 1.

These two properties imply that $p(x) = \sum p_i B_i(x)$ for $x \in [0,1]$ is a linear convex combination of the coefficients p_i. With coefficient components $p_i \in \mathbb{R}$, $p(x)$, $x \in [0,1]$ is contained in $[\min p_i, \max p_i]$, and this enclosure is tight. For control points p_i in 2D or 3D, $p(x)$ describes a Bézier curve, and the curve $p(x)$, $x \in [0,1]$ lies in the convex hull of its control points p_i. For example, since $x = \frac{0}{d} B_0(x) + \frac{1}{d} B_1(x) + \ldots + \frac{d}{d} B_d(x)$, the polynomial curve $(x, y = p(x))$ for $x \in [0,1]$ lies in the convex hull of its control points $(i/d, p_i)$.

In contrast to the canonical basis $(1, x, x^2, \dots x^d)$, the control points in the TBB depend on the domain interval. The classical de Casteljau algorithm provides the control points for the curve section $p(x)$, $x \in [0,t]$, and for the curve section $p(x)$, $x \in [t,1]$ with $t \in (0,1)$.

For multivariate polynomials, a Bernstein basis can be constructed using the tensorial product of univariate Bernstein basis functions

$$(B_0^{(d_1)}(x_1), \dots B_{d_1}^{(d_1)}(x_1)) \; \dots \; (B_0^{(d_n)}(x_n), \dots B_{d_n}^{(d_n)}(x_n))$$

Then, a multivariate polynomial p of degree $(d_1, \dots, d_n)$ is represented as

$$p(x) = \sum_{0 \leq i_1 \leq d_1} \dots \sum_{0 \leq i_n \leq d_n} p_{i_1 \dots i_n} B_{i_1}^{(d_1)}(x_1) \dots B_{i_n}^{(d_n)}(x_n)$$

The de Casteljau method extends to the TBB, and it can be used to subdivide the patch.

Subdivision solvers for algebraic equations, Bézier curves and surfaces exist in some variations. They all express a multivariate polynomial in the TBB since they require tight enclosures of the polynomial's range [3,4] or of the polynomial's roots in the domain [5,1]. In the *projected polyhedron algorithm* [5], a domain reduction of variable x_j is performed by intersecting the convex hull of projected control points $(i_j/d_j, p_{i_1,\dots,i_j,\dots,i_n})_{i_j=0,\dots,d_j}$ with the axis x_j. The method in ref. 1 improves the domain reduction by computing the first and last intersection of the two univariate, degree-d_j Bézier curves defined by control points $(i_j/d_j, \min_{i_k,k\neq j} p_{i_1,\dots,i_j,\dots,i_n})_{i_j=0,\dots,d_j}$ and $(i_j/d_j, \max_{i_k,k\neq j} p_{i_1,\dots,i_j,\dots,i_n})_{i_j=0,\dots,d_j}$ with the axis x_j. The reductions using the range-bounding curves are more effective than the ones using simply the convex hull. For the details of Mourrain/Pavone's method see Section 7.

Note that the TBB contains a number $(1+d_1)(1+d_2)\dots(1+d_n)$ of basis functions, where d_i is the maximum degree of variable x_i. Even for linear systems, this number 2^n has exponential growth. The canonical basis in n variables has the same number of functions but polynomial systems given in the canonical basis are sparse. They are not sparse anymore after the change to the TBB. For an illustration, the monomial 1 expressed in the TBB has a length, which is exponential in terms of n: $1 = (B_1^{(d_1)}(x_1)+\dots+B_{d_1}^{(d_1)}(x_1))\ \dots\ (B_1^{(d_n)}(x_n)+\dots+B_{d_n}^{(d_n)}(x_n))$. Clearly, the existing Bernstein-based solvers become impractical for a large number n of variables due to the exponential growth of the number of basis coefficients in terms of n. In geometric constraint solving, especially in 3D, systems with a large number of variables occur frequently. For an example, the vertex coordinates of a dodecahedron can be computed by means of geometric insight, i.e., that all its vertices lie on its circumscribed sphere. Alternatively, in a descriptive geometry system the dodecahedron might be described by its simplicial elements: the lengths of its 30 edges connecting the 20 vertices, and the coplanarities of its 12 faces.

3. Bernstein Polytope

For univariate polynomials, the inequalities $B_i(x) \geq 0$ are used to define the halfspaces of a convex polyhedron in $\mathbb{R}^d$, which encloses the curve $(x, x^2, \dots x^d)$, $x \in [0,1]$. We call it the *Bernstein polytope*. For $d = 2$, the Bernstein polytope is a triangle, see Figure 2. Its delimiting halfspaces are

$$\begin{array}{ll} B_0^{(2)}(x) = (1-x)^2 \geq 0 & \rightarrow y - 2x + 1 \geq 0 \\ B_1^{(2)}(x) = 2x(1-x) \geq 0 & \rightarrow 2x - 2y \geq 0 \\ B_2^{(2)}(x) = x^2 \geq 0 & \rightarrow y \geq 0 \end{array}$$

For $d = 3$, the Bernstein polytope is a tetrahedron, see Figure 3. Its vertices are $v_0 = (0,0,0)$, $v_1 = (1/3,0,0)$, $v_2 = (2/3,1/3,0)$ and $v_3 = (1,1,1)$. For example, v_0 lies on

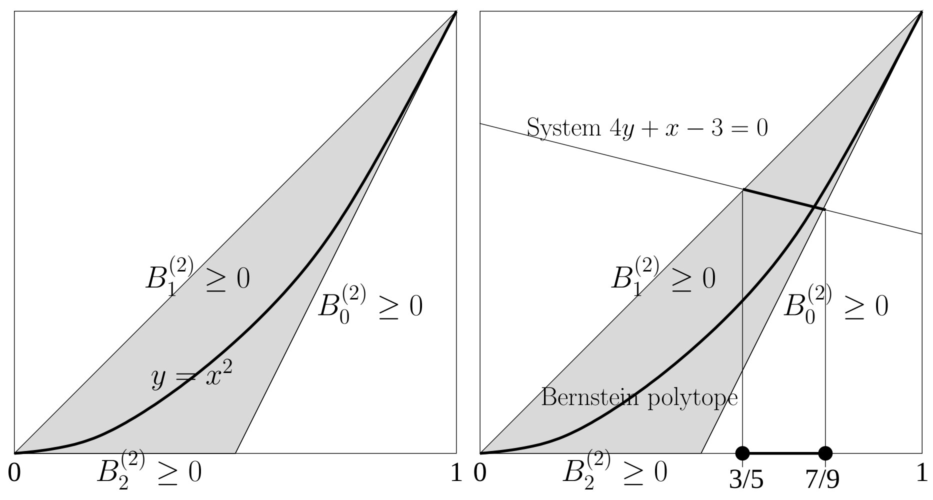

Fig. 2. The Bernstein polytope, a triangle, enclosing the curve $(x, y = x^2)$ for $x \in [0,1]$.

$B_1 = B_2 = B_3 = 0$ and v_1 lies on $B_0 = B_2 = B_3 = 0$.

$$\begin{aligned}
B_0^{(3)}(x) &= (1-x)^3 \geq 0 && \rightarrow 1 - 3x + 3y - z \geq 0\\
B_1^{(3)}(x) &= 3x(1-x)^2 \geq 0 && \rightarrow 3x - 6y + 3z \geq 0\\
B_2^{(3)}(x) &= 3x^2(1-x) \geq 0 && \rightarrow 3y - 3z \geq 0\\
B_3^{(3)}(x) &= x^3 \geq 0 && \rightarrow 3z \geq 0.
\end{aligned}$$

The extension to higher degrees is possible but illustrations are not easily possible anymore.

So far, we have considered curves with a single variable only. Representing a quadratic multivariate polynomial with them requires a separation of variables occurring in the mixed monomials of the form xy. This is possible, for example, using $xy = 1/2((x+y)^2 - x^2 - y^2)$.

But we prefer an extension of the Bernstein polytope for the mixed monomial xy as follows. Figure 4 shows the Bernstein polytope, enclosing the surface patch $(x, y, z = xy)$, a tetrahedron. Its halfspaces are defined by

$$\begin{aligned}
B_0^{(1)}(x)B_0^{(1)}(y) \geq 0 &\rightarrow 1 - x - y + z \geq 0\\
B_0^{(1)}(x)B_1^{(1)}(y) \geq 0 &\rightarrow y - z \geq 0\\
B_1^{(1)}(x)B_0^{(1)}(y) \geq 0 &\rightarrow x - z \geq 0\\
B_1^{(1)}(x)B_1^{(1)}(y) \geq 0 &\rightarrow z \geq 0.
\end{aligned}$$

This tetrahedron is optimal: it is the convex hull of the patch.

In summary, the inequalities for multivariate polynomials are obtained as the product of the inequalities for univariate polynomials.

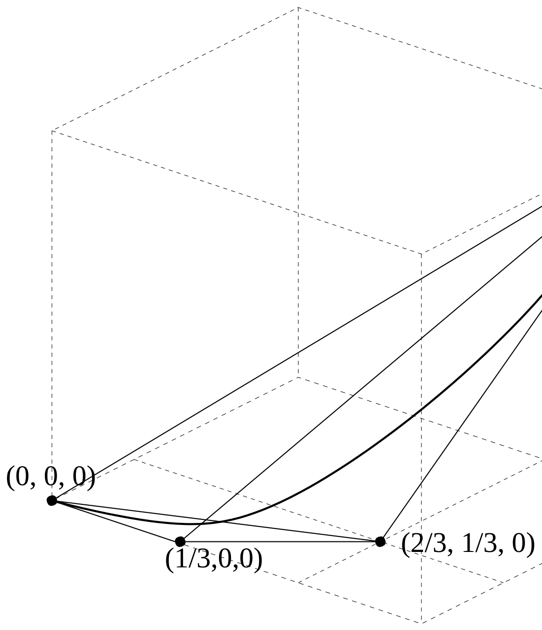

Fig. 3. The Bernstein polytope, a tetrahedron, enclosing the curve $(x, y = x^2, z = x^3)$ with $x \in [0,1]$.

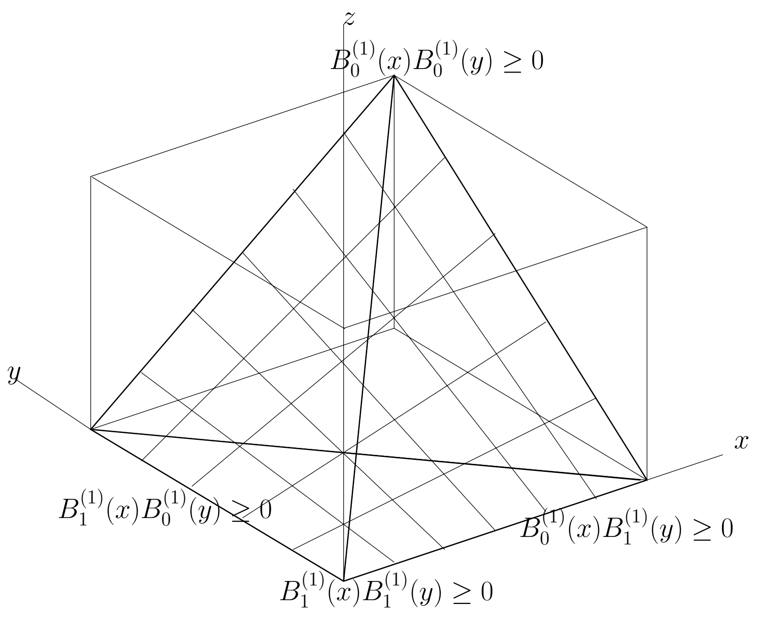
z
$B_0^{(1)}(x)B_0^{(1)}(y) \geq 0$
y
x
$B_1^{(1)}(x)B_0^{(1)}(y) \geq 0$
$B_0^{(1)}(x)B_1^{(1)}(y) \geq 0$
$B_1^{(1)}(x)B_1^{(1)}(y) \geq 0$

Fig. 4. The Bernstein polytope enclosing the surface patch $(x, y, z = xy)$.

4. Subdivision Solver

With the polytope of Section 3, linear programming (LP) [6] can be used for the computation of a bound of a polynomial's range and for the reduction of a domain box while preserving roots. In the following sections, we use the simplex algorithm in tableau form for the exposition of the method although in practice the revised simplex algorithm is recommended. We refer the reader to the exposition in ref. 6 for the efficient implementation of the simplex algorithm. In our experiments in Sections 7 and 8, we used the primal-dual revised simplex code SoPlex 1.4.2 [7] on a Windows XP 32Bit system (2GB RAM) with an Intel T7200 Core2 Duo processor (2.2GHz).

4.1. *Range Bounding*

To compute a lower and an upper bound of the polynomial $p(x) = 4x^2 + x - 3$ for $x \in [0,1]$, substitute a new variable for the nonlinear monomial: $y = x^2$. Then minimize and maximize the linear objective function $4y + x - 3$ on the Bernstein polytope enclosing the curve $(x, y = x^2)$, $x \in [0,1]$ (the triangle in Figure 2). After replacing the monomial x^2 with the LP variable y, it is an LP problem.

$$\begin{aligned}
&\min/\max p = 4y + x - 3\\
&B_0 = y - 2x + 1\\
&B_1 = -2y + 2x\\
&B_2 = y\\
&B_0 \geq 0, B_1 \geq 0, B_2 \geq 0, x \geq 0, y \geq 0
\end{aligned}$$

The simplex algorithm [6] provides the enclosure $[-3, 2]$.

$$\begin{array}{ll}
\min p = -3 + x + 4y, & \max p = 2 - 5B_0 - 9/2B_1\\
B_0 = 1 - 2x + y & x = 1 - B_0 - B_1/2\\
B_1 = 2x - 2y & y = 1 - B_0 - B_1\\
B_2 = y & B_2 = 1 - B_0 - B_1
\end{array}$$

Note that the variables on the right side ("not in basis") have value 0.

4.2. *Domain Reduction*

Solving an equation $4x^2 + x - 3 = 0$, $x \in [0,1]$ is equivalent to finding the intersection points between the line $4y + x - 3 = 0$, and the curve $(x, y = x^2)$. This curve is enclosed in its Bernstein polytope, the triangle of Figure 2. Intersecting the line and the triangle, i.e., finding the minimum and maximum value of x, reduces the domain of variable x. With the same polytope as above, we minimize and maximize the objective function x. The solutions are

$$\begin{array}{ll}
\min x = 3/5 + 2/5B_1, & \max x = 7/9 - 4/9B_0 \\
x = 3/5 + 2/5B_1 & x = 7/9 - 4/9B_0 \\
y = 3/5 - 1/10B_1 & y = 5/9 + 1/9B_0 \\
B_0 = 2/5 - 9/10B_1 & B_1 = 4/9 - 10/9B_0 \\
B_2 = 3/5 - 1/10B_1 & B_2 = 5/9 + 1/9B_0
\end{array}$$

Thus the domain interval $[0,1]$ for variable x has been reduced to $[3/5, 7/9]$.

For reducing the domain box while preserving roots, $2n$ LP problems are solved, a minimization and a maximization for each variable X_k, $k = 1, \ldots, n$. If the LP problem is not feasible then the domain box contains no root. Note that these $2n$ LP problems are independent and can be solved in parallel.

4.3. *Scaling*

After reduction, the domain box is not $[0,1]^n$ anymore. There are two possibilities to handle arbitrary domain boxes $[u,v] \subset \mathbb{R}^n$. Either, we scale each variable $x_i \in [u_i, v_i]$, $u_i \le v_i$, to $X_i \in [0,1]$, $i = 1, \ldots, n$, or we formulate the Bernstein polytope of Section 3 for an arbitrary domain box $[u,v] \subset \mathbb{R}^n$.

The variable scaling from the box $[u,v]$ to the unit hypercube $[0,1]^n$ is given by

$$\begin{array}{ll}
x_i & = u_i + (v_i - u_i)X_i, \\
x_i^2 & = (v_i - u_i)^2 X_i^2 + 2u_i(v_i - u_i)X_i + u_i^2, \\
x_i x_j & = (v_i - u_i)(v_j - u_j)X_i X_j + u_i(v_j - u_j)X_j + u_j(v_i - u_i)X_i + u_i u_j
\end{array}$$

The scaling is a linear map of the LP variables (X_i, X_{ii}, X_{ij}), $i, j = 1, \ldots, n$, $i < j$, which has to be performed on the system of polynomial equations.

With a formulation of the Bernstein polytope for an arbitrary domain box, the system of polynomial equations remains unchanged. For an arbitrary domain box $[u,v] \subset \mathbb{R}^n$, the Bernstein polytope is given by

$$\begin{array}{ll}
(v_i - x_i)^2 \ge 0 & \to X_{ii} - 2v_i X_i + v_i^2 \ge 0 \\
2(x_i - u_i)(v_i - x_i) \ge 0 & \to 2(-X_{ii} + (u_i + v_i)X_i - u_i v_i) \ge 0 \\
(x_i - u_i)^2 \ge 0 & \to X_{ii} - 2u_i X_i + u_i^2 \ge 0 \\
(v_i - x_i)(v_j - x_j) \ge 0 & \to X_{ij} - v_i X_j - v_j X_i + v_i v_j \ge 0 \\
(v_i - x_i)(x_j - u_j) \ge 0 & \to -X_{ij} + u_j X_i + v_i X_j - v_i u_j \ge 0 \\
(x_i - u_i)(x_j - u_j) \ge 0 & \to X_{ij} - u_j X_i - u_i X_j + u_i u_j \ge 0
\end{array}$$

5. Floating Point Errors

The Bernstein polytope encloses very tightly the underlying algebraic manifold:

$$(x_1, \ldots, x_n, x_1^2, \ldots, x_n^2, x_1x_2, \ldots, x_{n-1}x_n),\ x_i \in [0,1]$$

thus with a naive floating-point implementation, some roots are omitted because of rounding errors. For example, when solving $x^2 - x = 0$ with $x \in [0,1]$, the line $y - x = 0$ is considered (see Figure 2). If this line becomes $y - x = \varepsilon$ due to error $\varepsilon > 0$, the two roots are missed.

There are several strategies to cope with floating point inaccuracy:

- *Rational Arithmetic throughout:* Use exact computations with rational numbers for scaling and for the LP solver. Then the program becomes slower by several orders of magnitude due to an increasing representation size.
- *Rational Arithmetic for LP solver:* Use exact computations with rational numbers for the LP solver but work with a fixed precision at the interface to the LP solver. For example, round the box $[20001/10000, 29999/10000]$ to the box $[200/100, 300/100]$. The rounding ensures rational numbers of bounded representation size at least at the interface of the LP solver, i.e., in the polytope specification.
- *Rational Arithmetic for Linear Solver:* Compute an approximate solution with an LP solver in floating-point arithmetic. Then start from the approximate LP solution basis with an LP solver in rational arithmetic to compute the exact solution.[8]
- *Polytope Inflation by the Error Value:* During pivoting inside the simplex algorithm, the errors of floating point operations performed in each row are collected and stored with the row's constant. Finally, the hyperplane is pushed outward by this amount. It can happen that the exact feasible set is empty but after the inflation it is no more empty. This approach is conservative, i.e., it guarantees that no root is missed.

6. Nonlinear Reduction for Squared Variables

We introduce an important optimization of reduction for squared variables x_i^2, which is not based on the polytope for x_i^2 but uses the nonlinear curve (x_i, x_i^2) directly. Note that most of the complexity of multivariate polynomials comes from the occurrence of mixed monomials though.

It is possible to derive bounds from the nonlinear function (x_i, x_i^2) directly instead of from its Bernstein polytope, see Figure 5. If $\min x_i^2$ and $\max x_i^2$ is known, then $\min x_i \geq \sqrt{\min x_i^2}$ and $\max x_i \leq \sqrt{\max x_i^2}$. The values $\min x_i^2$ and $\max x_i^2$ can be obtained by solving two additional LP problems with objective functions $\min X_{ii}$ and $\max X_{ii}$ respectively. In the special case, where after computing $\min X_i$ and $\max X_i$ one of the inequalities $B_0(X_i) \geq 0$, $B_2(X_i) \geq 0$, or $B_1(X_i) \geq 0$, occur in the optimal bases of the LP, the range of X_{ii} can be obtained directly from the LP solution vectors $X^{\min} = (\ldots, X_i^{\min}, X_{ii}^{\min}, \ldots)$ and $X^{\max} = (\ldots, X_i^{\max}, X_{ii}^{\max}, \ldots)$ as:
$\min X_{ii} = \min(X_{ii}^{\min}, X_{ii}^{\max})$ and $\max X_{ii} = \max(X_{ii}^{\min}, X_{ii}^{\max})$.
Proof. Let $X_i^{\min}$ be the minimum value and $X_i^{\max}$ be the maximum value of variable X_i from the two LP solutions. Then the intersection of the polytope $\{(X_i, X_{ii}) : X_{ii} - 2X_i + 1 \geq 0, X_i - X_{ii} \geq 0, X_{ii} \geq 0\}$ with $X_i = X_i^{\min}$ or $X_i = X_i^{\max}$ is either the interval $\{X_{ii} : X_{ii} - 2X_i + 1 \geq 0, X_i - X_{ii} \geq 0\}$, or the interval $\{X_{ii} : X_{ii} \geq 0, X_i - X_{ii} \geq 0\}$, see Figure 5. Due to convexity of the feasible set $\{X_i, X_{ii} : X \text{ is feasible}\}$, the intersection of the two X_{ii}-intervals gives the possible interval for X_{ii}. $\square$

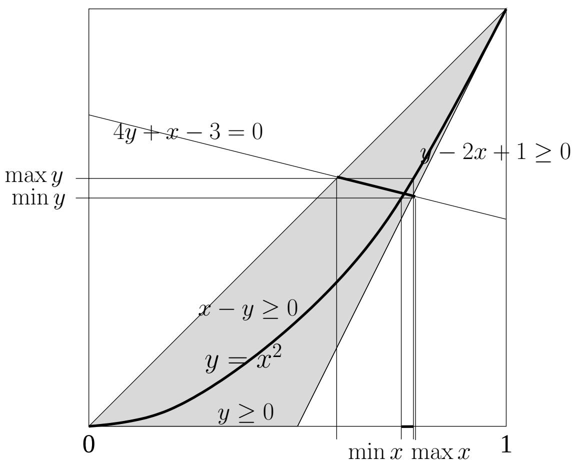

Fig. 5. Bounds obtained from the nonlinear function $(x, y = x^2)$ directly.

Consequently for systems containing only linear and squared monomials, the optimization is possible without additional LP calls. If also mixed monomials $x_i x_j$, $i < j$ occur, the nonlinear reduction is not possible without the explicit computation of $\min X_{ii}$ and $\max X_{ii}$ as we demonstrate with examples in Section 7.2. These $2n$ supplementary LP problems can be solved in parallel.

7. Comparison with a Tensorial Bernstein Based Solver

In this section, we compare the range bounds and the solution domain bounds obtained by our solver based on the Bernstein polytope and the TBB solver in ref. 1 for total degree 2.

The TBB solver computes the coefficients in the tensorial Bernstein basis for a domain box $[u, v]$

$$p((v-u)x+u) = \sum_{0\leq i_1\leq 2} \dots \sum_{0\leq i_n\leq 2} p_{i_1\dots i_n} B^{(2)}_{i_1}(x_1)\dots B^{(2)}_{i_n}(x_n) \text{ with } x \in [0,1]^n$$

The coefficients $b_{i_1\dots i_n}$ with respect to the TBB can be computed as described in refs. 1, 9 and 10. The polynomial's range $p((v-u)x-u)$, $x \in [0,1]^n$ is then bounded by $[\min p_{i_1\dots i_n}, \max p_{i_1\dots i_n}]$. See 1 for a proof of this statement.

In ref. 1, an even stricter bound is derived by projection for each dimension $j = 1, \dots, n$

$$\sum_{i_j=0,1,2} (\min_{i_k, k\neq j} p_{i_1\dots i_n}) B^{(2)}_{i_j}(x_j) \leq p((v-u)x+u) \leq \sum_{i_j=0,1,2} (\max_{i_k, k\neq j} p_{i_1\dots i_n}) B^{(2)}_{i_j}(x_j)$$

The bounds are quadratic Bézier curves, from which domain bounds for the variables x_j can be obtained: Simply intersect the bounding Bézier curves with the axis x_j and combine the results from the minimum and the maximum curve correctly. Assume the quadratic system $P(x) = 0$ has an invertible Jacobian matrix $P'(m)$ at the center m of the domain D. Then it is possible to make the system jacobian the unit matrix at the center m. This

Table 1. Comparison of the solvers' performance for the systems in Figure 6. Entries "n.a." denote systems, for which the nonlinear reduction of Section 6 can not be applied without solving additional LP problems.

	Nb. Reduction	Nb. Bisection	Avg. Range-Factor	Avg. Domain-Factor
Mourrain/Pavone				
system (a)	15	5	3.3896	(4.19672, 1.79867)
system (b)	53	15	3.3730	(1.79106, 1.96632)
system (c)	8	1	1024.0	(1374.09, 95.9520)
system (d)	10	3	2.31344	(1.11638, 2.31344)
Our LP method (reduction of all dimensions once)				
system (a)	18	5	13.1280	(1.9967,11.4821)
system (b)	26	7	6.5523	(2.17404, 5.2301)
system (c)	28	13	1.9291	(7.3423e+11, 1.3846)
system (d)	10	3	12.6969	(1.3142, 2.5067)
Our LP method with preconditioning (reduction of all dimensions once)				
system (a)	18	5	13.1130	(1.99673,11.4821)
system (b)	26	7	6.5248	(2.17404, 5.2301)
system (c)	28	13	1.9291	(7.3423e+11, 1.3846)
system (d)	10	3	12.2724	(1.3142, 2.5067)
Our LP method with square root opt (reduction of all dimensions once)				
system (a)	12	3	11.4926	(2.3002, 9.1139)
system (b)	n.a.	n.a.	n.a.	n.a.
system (c)	2	0	7.2149e+011	(7.3423e+011, 2.0)
system (d)	n.a.	n.a.	n.a.	n.a.
Our LP method (reduction of a dimension as long as reduction factor >2.0				
system (a)	19	1	17.7660	(9.2743,13.3139)
system (b)	50	7	7.1166	(2.6270, 4.1077)
system (c)	25	11	1.9178	(3.3640e+12, 1.3939)
system (d)	6	1	8.5097	(1.8468, 1.6842)

preconditioned system $P(x)P'(m)^{-1} = 0$ has the same roots as the given system $P(x) = 0$, and it ensures a quadratic convergence [1] for simple roots. In the comparison, we use this *preconditioning step* with the TBB solver.

For our experiments, we use the two-dimensional algebraic curves as shown in Figure 6. Firstly, we measure the reduction factors of range values during the execution of both methods (Section 7.1). Then we perform reduction steps once in each dimension and compute the reduction factors of the interval widths. These empirically determine the convergence behavior of the methods. Section 7.2 gives the comparison.

7.1. *Range Bounds*

Range bounds on system equations are not necessary for solving the system but can be determined with both methods. For linear programming with the Bernstein polytope, it is

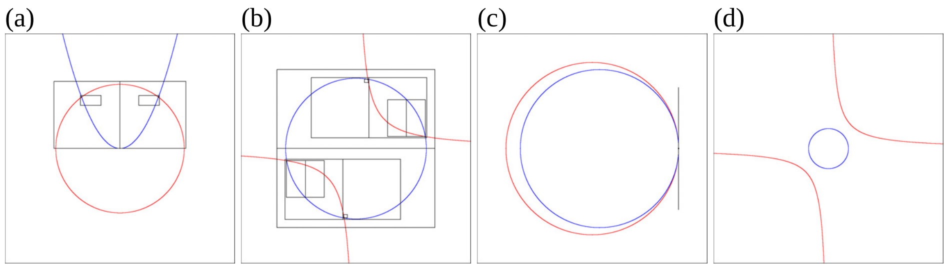

Fig. 6. Algebraic 2D curves: (a) parabola $x_1 - x_0^2 = 0$ and circle, (b) hyperbola $x_0 x_1 = 1$ and circle, (c) two touching circles, and (d) hyperbola $x_0 x_1 = 4$ and circle.

described in Section 4.1. With the TBB, range bounds are obtained from the minimum and the maximum TBB coefficient, as described in Section 7. For two variables, there is only a small difference between the ranges computed with the TBB coefficients and the ones computed with linear programming on the same domain interval. Differences in the range reduction factor are caused mainly by different domain intervals.

Table 1 contains the number of reduction steps, the number of bisections, the average reduction factor $\frac{|range(D)|}{|range(D')|}$ of range width $|range(D)|$ (column 3), and the average reduction factor $\frac{|D_i|}{|D_i'|}$ of domain widths $|D_i|$. Therein, D is the domain and D' is the domain after a single reduction step of all variables. Note that both methods were run by doing a bisection immediately after a single reduction step until all domain side lengths are smaller than $\delta = 10^{-3}$. The process started with initial domain box $[-10, 10]^2$. System (a) has two single roots, (b) has four single roots, (c) has a double root, and (d) has no root. From the measurements, it is remarkable that the TBB solver generates roughly the same range reductions (factor 3.3) in normal situations, and achieves a factor 1024.0 for the tangential contact of system (c). For linear programming on the Bernstein polytope, the number of reductions and bisections are similar, whereas the range reduction factors are more varied. An exception is system (c) with tangential contact, where our method requires some more reductions and bisections.

For a comparison, we applied the same preconditioning step, described in Section 7, to the system solved by the simplex algorithm with the Bernstein polytope. The effect of preconditioning on the revised simplex algorithm, which uses linear systems solving internally, is very small as shown in Table 1. The revised simplex algorithm is in this respect more similar to linear systems solving and its techniques to improve accuracy (pivoting, equilibration [11]).

Table 1 also gives the statistics for the simplex algorithm with the Bernstein polytope using the nonlinear reductions of Section 6, where applicable without solving additional LP problems, i.e., for systems (a) and (c). The nonlinear reductions are very effective and largely reduce the domain widths.

7.2. *Convergence*

For the domain reduction steps, Table 1 contains the domain reduction factors in column 4. It can be seen that linear programming with the Bernstein polytope for variable $x_{i'}$ profits from reductions of variable x_i, $i < i'$. Usually, the reduction factor of the second variable x_2 is considerably greater than the one for the first variable x_1. The TBB solver can not take advantage similarly as the conversion to the tensorial Bernstein basis is done once before reducing all the variables.

In the paper 1, the quadratic convergence of the method in the case of a single root was proved. For the double root of system (c), the reduction is very good due to the preconditioning step. The revised simplex algorithm with the Bernstein polytope achieves similar reduction factors as the TBB solver in cases with single roots as can be seen in Table 1, systems (a), (b), (d). Also in multiple root cases, it has an average reduction factor $f > 1$, and is thus at least linearly convergent.

8. Kinematics of Gough-Stewart Platform

Gough-Stewart platforms are used as parallel robots. We consider here the structure made of two triangles connected by jacks (translational joints) into an octahedron [12,13]. See Figure 8 for an illustration.

(a) (b)

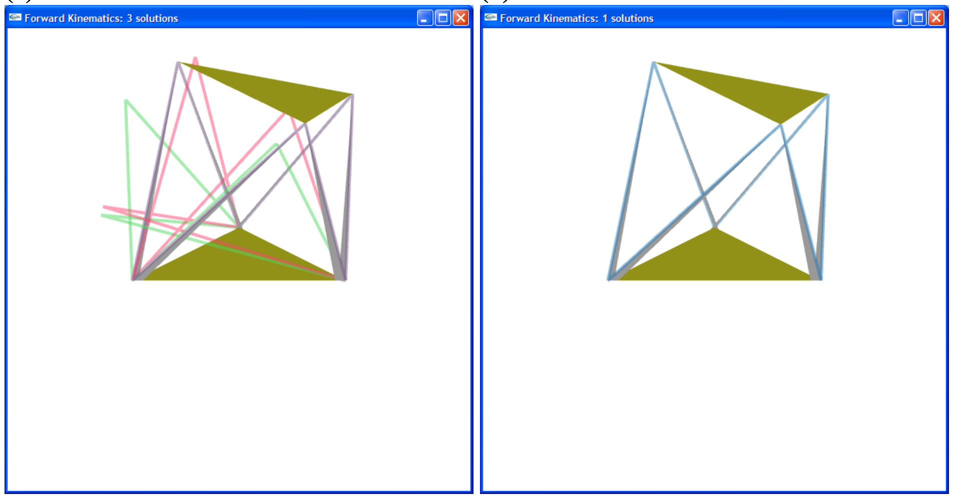
Forward Kinematics: 3 solutions
Forward Kinematics: 1 solutions

Fig. 7. (a) Several solutions for the given jack lengths above the base platform. Among them are two solutions that have a side change of the jack direction relative to the upper platform. (b) Solution without a side change of the jack directions.

The lower triangle serves as the base, and the upper triangle moves as the work platform. Edges of the platform and of the base are rigid, i.e., their lengths are fixed once (brown triangles in Figure 8): p_2p_3, p_3p_1, p_1p_2, p_6p_4, p_4p_5, p_5p_6. The lengths of the remaining edges are computer-controlled (gray lines in Figure 8): p_1p_4, p_1p_6, p_2p_4, p_2p_5, p_3p_5, p_3p_6.

In the following section, we consider in detail the *forward kinematics problem*, where the six side lengths d_{23}, d_{31}, d_{12}, d_{64}, d_{45}, d_{56}, and the six lengths of the jacks d_{14}, d_{16}, d_{24}, d_{25}, d_{35}, d_{36} are given, and the coordinates of the upper platform's points (three points of the octahedron) are to be computed. As usual for a parallel structure, the *inverse kinematics problem* is much easier, as the lengths of the six jacks can be computed directly from the points' coordinates.

8.1. *Forward Kinematics*

As shown in ref. 14, the forward kinematics problem can have up to 16 solutions.

The problem can be formulated directly in Cartesian coordinates if we fix the coordinates of the lower platform's points. Here is an example.

$$\begin{array}{l}
\text{fixed lower triangle}\\
x_1 = 0\\
y_1 = 0\\
z_1 = 0\\
x_2 = 1\\
y_2 = 0\\
z_2 = 0\\
x_3 = 0.5\\
y_3 = \sqrt{3/2}\\
z_3 = 0
\end{array}$$

$$\begin{array}{l}
\text{given distances of the upper triangle's points}\\
(x_4-x_5)^2+(y_4-y_5)^2+(z_4-z_5)^2 = d_{45}^2\\
(x_5-x_6)^2+(y_5-y_6)^2+(z_5-z_6)^2 = d_{56}^2\\
(x_6-x_4)^2+(y_6-y_4)^2+(z_6-z_4)^2 = d_{64}^2\\
\text{given lengths of jacks}\\
(x_2-x_4)^2+(y_2-y_4)^2+(z_2-z_4)^2 = d_{24}^2\\
(x_3-x_4)^2+(y_3-y_4)^2+(z_3-z_4)^2 = d_{34}^2\\
(x_3-x_5)^2+(y_3-y_5)^2+(z_3-z_5)^2 = d_{35}^2\\
(x_1-x_5)^2+(y_1-y_5)^2+(z_1-z_5)^2 = d_{15}^2\\
(x_1-x_6)^2+(y_1-y_6)^2+(z_1-z_6)^2 = d_{16}^2\\
(x_2-x_6)^2+(y_2-y_6)^2+(z_2-z_6)^2 = d_{26}^2
\end{array} \tag{3}$$

Interesting possibilities exist for selecting solutions. For example, due to mechanical restrictions the jacks can not change side with respect to the work platform. Let $n_{\text{work}} = (p_5-p_4)\times(p_6-p_4)$ be the normal of the work platform, where $v_1 \times v_2$ denotes the cross product of two vectors. Then the scalar products $n\cdot(p_1-p_6)\le 0$, $n\cdot(p_1-p_5)\le 0$, $n\cdot(p_2-p_6)\le 0$, $n\cdot(p_2-p_4)\le 0$, $n\cdot(p_3-p_5)\le 0$, $n\cdot(p_3-p_4)\le 0$ formulate this condition. We have to add three variables n_1, n_2, n_3 to the system for the normal direction. See Figure 7 for an example solution using this condition.

Other conditions can be formulated, i.e., the work platform should be upright with respect to the base platform ($n_{\text{base}} \cdot n_{\text{work}} \geq 0$). Note that this condition is not already implied by the condition that the jacks do not change side with respect to the work platform.

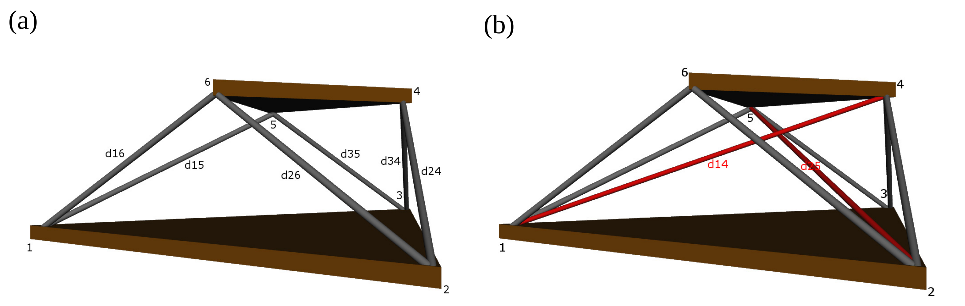

Fig. 8. (a) Gough-Stewart platform. (b) Gough-Stewart platform with two distances (red lines) used in the Cayley-Menger formulation.

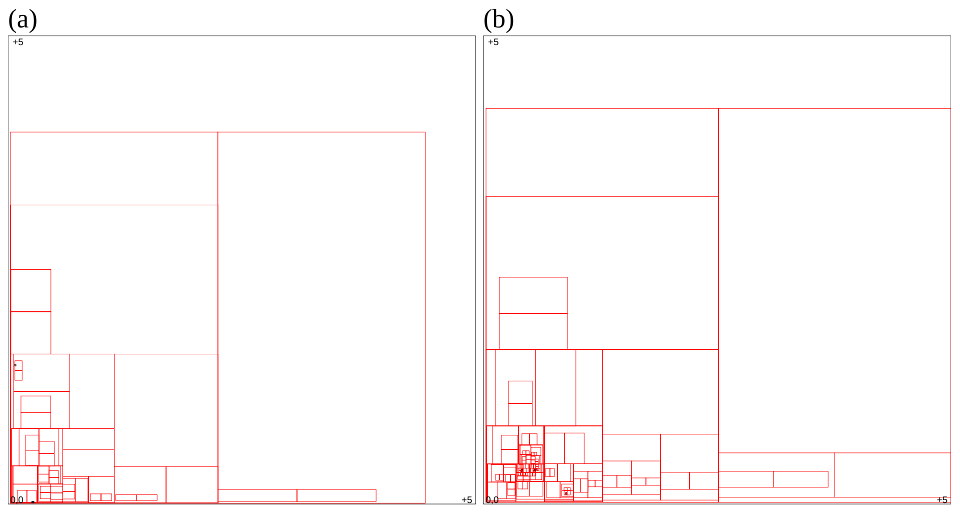

Fig. 9. Solving Cayley-Menger equations for D_{14} and D_{25}. Small black boxes mark the solution points (D_{14}, D_{25}): (a) 2 solutions, (b) 3 solutions.

With Cayley-Menger determinants, it is possible to give a coordinate-free formulation. The Cayley-Menger determinant gives a relation of all the squared distances between five points in 3-space.[15] For points p_1, p_2, p_3, p_4, p_5, the Cayley-Menger determinant contains the squared distances $D_{12} = d_{12}^2$, $D_{13} = d_{13}^2$, $D_{14} = d_{14}^2$, $D_{15} = d_{15}^2$, $D_{23} = d_{23}^2$, $D_{24} = d_{24}^2$, $D_{25} = d_{25}^2$, $D_{34} = d_{34}^2$, $D_{35} = d_{35}^2$, $D_{45} = d_{45}^2$. All these squared distances are known, except

Table 2. Comparison of the solvers' performance for the Gough-Stewart forward kinematics. Entries "?" denote computations that did not terminate within one minute.

	Nb. Solutions	Nb. Reduction	Nb. Bisection	Time
Mourrain/Pavone				
Cartesian n=9	?	?	?	>60.0s
Cartesian n=12, no side change	?	?	?	>60.0s
Cayley-Menger n=3	2	1153	103	0.0157s
Cayley-Menger n=3	3	4235	397	0.0549s
Our LP method				
Cartesian n=9,	3	393	47	16.555s
Cartesian n=12, no side change	1	365	43	17.946s
Cayley-Menger n=3	2	143	33	0.0680s
Cayley-Menger n=3	3	397	92	0.1783s

for D_{14} and D_{25} (red lines in Figure 8), and the equation has degree 4 in the 2 unknowns.

$$\begin{aligned}
\det(M) &= \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & D_{12} & D_{13} & D_{14} & D_{15} \\ 1 & D_{21} & 0 & D_{23} & D_{24} & D_{25} \\ 1 & D_{31} & D_{32} & 0 & D_{34} & D_{35} \\ 1 & D_{41} & D_{42} & D_{43} & 0 & D_{45} \\ 1 & D_{51} & D_{52} & D_{53} & D_{54} & 0 \end{vmatrix} \\
&= a_9 D_{14}^2 D_{25}^2 + a_8 D_{14}^2 D_{25} + a_7 D_{14} D_{25}^2 + a_6 D_{14}^2 + a_5 D_{25}^2 + a_4 D_{14} D_{25} \\
&\quad + a_3 D_{14} + a_2 D_{25} + a_1 \\
&= a_9 D_{14,25}^2 + a_8 D_{14} D_{14,25} + a_7 D_{14,25} D_{25} + a_6 D_{14}^2 + a_5 D_{25}^2 + a_4 D_{14} D_{25} \\
&\quad + a_3 D_{14} + a_2 D_{25} + a_1 \\
&\overset{!}{=} 0
\end{aligned} \tag{4}$$

A second, independent equation can be generated for the points p_1, p_2, p_4, p_5, p_6. We can reformulate this system of degree 4 into a system of degree 2 by introducing an auxiliary variable $D_{14,25}$ for the product $D_{14,25} = D_{14}D_{25}$. Then the monomials $D_{14}^2 D_{25}^2$, $D_{14}^2 D_{25}$, and $D_{14} D_{25}^2$ become degree 2: $D_{14,25}^2$, $D_{14}D_{14,25}$, and $D_{14,25}D_{25}$.

If the lengths d_{14} and d_{25} of two diagonals are known, we can compute the Cartesian coordinates of the three points on the upper triangle. For each point p_4 and p_5, the distances to points p_1, p_2, p_3 are known so that their coordinates can be computed as the intersection of three spheres. The coordinates of point p_6 can then be computed in the same way from the distances to points p_1, p_4, p_5. As an example, the intersection of the three spheres

$(p_1, r_1 = d_{14})$, $(p_2, r_2 = d_{24})$, $(p_3, r_3 = d_{34})$ results in two possible solutions p_4, p_4^*.

$$\begin{aligned}
p_{21} &= p_2 - p_1 \\
p_{31} &= p_3 - p_1 \\
c &= p_{21} \times p_{31} \\
u &= 0.5(d_{21}^2 + d_{14}^2 - d_{24}^2)p_{31} - 0.5(d_{31}^2 + d_{14}^2 - d_{34}^2)p_{21} \\
v &= \sqrt{d_{14}^2 - |u|^2}\, c/|c| \\
p_4 &= p_1 + u + v \\
p_4^* &= p_1 + u - v
\end{aligned} \tag{5}$$

Here, the vector u is from point p_1 to the intersection point in the plane $p_1 p_2 p_3$ of the two intersection circle planes (defined by the sphere intersections of (p_1, r_1), (p_2, r_2) and of (p_1, r_1), (p_3, r_3)). The vector v points from this intersection point along c, which is orthogonal to the plane $p_1 p_2 p_3$, to an intersection point of the three circles. As the three points p_1, $p_1 + u$, p_4 form a right-angled triangle with two known side lengths $|p_1 - p_4| = d_{14}$ and $|p_1 + u - p_1| = |u|$, the third length $|p1 + u - p_4|$ needs to be $\sqrt{d_{14}^2 - |u|^2}$ by a theorem of Pythagoras.

Note that the solutions for p_4 and p_5 have to be combined according to the given distance d_{45}, which can have up to two solutions for each solution p_4. Figure 10 shows a geometric configuration in 2D, where a solution for p_4 connects to the two possibilities for p_5. Each one leaves two possibilities for point p_6. In total, it can have up to $4\,|\{(d_{14}, d_{25}) : d(M(D_{12}, D_{13}, D_{14}, D_{15}, D_{23}, D_{24}, D_{25}, D_{34}, D_{35}, D_{45})) = 0, d(M(D_{12}, D_{16}, D_{14}, D_{15}, D_{26}, D_{24}, D_{25}, D_{46}, D_{56}, D_{45})) = 0\}|$ solutions, where one half of them lies above the base platform and the other half lies below it due to symmetry.

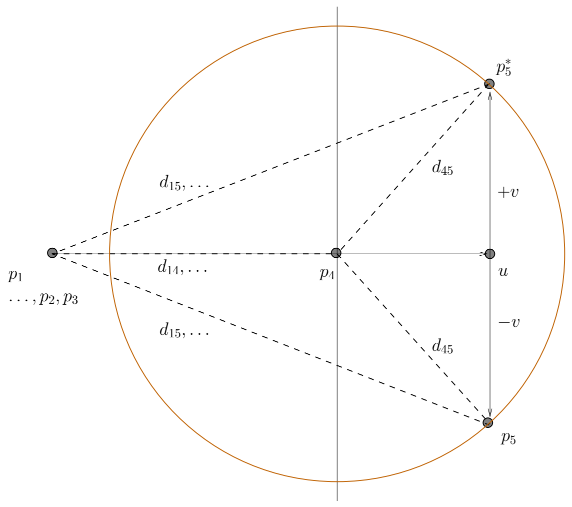

Fig. 10. A solution for point p_4 connects to up to two possibilities for point p_5.

The runtime for example systems is summarized in Table 2. A reduction step of all variables was immediately followed by a bisection of the domain box until all domain side lengths were smaller than $\delta = 10^{-4}$. For the initial domain box, $[-5,5]$ resp. $[0,5]$ was used for the Cartesian coordinate components, and $[0,5]$ for squared distances in the Cayley-Menger formulation. Only the LP reduction method can be used for the direct Cartesian formulation due to its large number of variables. For $n = 9$, the TBB has $3^9 = 19683$ basis functions, whose coefficients have to be calculated in each reduction step. For $n = 12$, there are already $3^{12} = 531441$ coefficients. The additional equations for selecting solutions (avoiding side changes of the jacks, upright orientation of the upper platform) have no significant effect on the runtime.

The Cayley-Menger formulation requires the solution of a small system only (2 equations of degree 4 in 2 variables, 3 equations of degree 2 in 3 variables), which is coordinate-free, and three calculations of the intersection of three spheres, which is coordinate-dependent but can be solved by a specialized code. The solution of the coordinate-free part is possible with both solvers. For $n = 3$ variables, the TBB solver is faster despite a larger number of reductions and bisections. A reduction step involves only $n = 3$ min/max searches over $3^3 = 27$ coefficients.

9. Conclusion

This article has compared two subdivision solvers for quadratic polynomial systems, which use the same algorithm only differing in the domain reduction step. The first solver in ref. 1 uses the tensorial Bernstein basis (TBB) representation of a multivariate polynomial. It uses the tensorial Bernstein basis to derive a pair of range-bounding Bézier curves for each variable x_j of the system. Based on these Bézier curves, the solver reduces the domain of each variable x_j by univariate root finding, which is beneficial especially in cases of multiple roots.[1] In contrast, our LP solver derives bounds of the solution domain from a polytope enclosure of the nonlinear patches over the domain. They are computed as solutions of linear programming problems, i.e., using the revised simplex algorithm. The polytope is defined by a number of halfspaces, which has polynomial length in terms of the input system, i.e., the number n of variables, and fixed total degree d.

In practice, problem formulations can easily incorporate a large number n of variables. In this case, solvers using the tensorial Bernstein basis suffer from the exponential number $(d+1)^n$ of basis functions for total degree d. With the LP-based method, the polytope is defined by a number of $O(n^d)$ halfspaces. For the Gough-Stewart platform, an octahedron, we use it to compute all solutions or specially selected solutions for the forward kinematics problem. The forward kinematics problem in Cartesian coordinates incorporates 9–12 variables and a similar number of (in-)equalities. This system size is currently intractable for a TBB solver. It can only be solved as a decomposed system, for example, using a coordinate-free formulation with Cayley-Menger determinants, which has only 2–3 variables.

The convergence behavior of both methods has been compared empirically for systems of two variables. At double roots, Mourrain's method benefits from preconditioning, which achieves a nearly unit jacobian matrix at the center of the domain. This preconditioning of the TBB solver has only a small effect with the revised simplex algorithm. The simplex algorithm is in this respect more similar to linear systems solving and its techniques to improve accuracy. In the comparison for single roots, numerical evidence confirms the quadratic convergence of both methods.

An optimization of the LP reduction-based solver is possible for systems containing squared variables x_i^2. In this case, we have demonstrated a nonlinear reduction of the domain interval for x_i based on the quadratic curve (x_i, x_i^2). The nonlinear reduction is directly applicable for systems containing only the monomials x_i and x_i^2. For arbitrary systems, it requires the solution of two additional LP problems $\min X_{ii}$ and $\max X_{ii}$ corresponding to x_i^2.

Acknowledgements

Christoph Fünfzig would like to thank the Conseil Régional de Bourgogne (Regional Council of Burgundy) for support in a postdoc scholarship.

References

1. Mourrain B., and Pavone J.-P. Subdivision methods for solving polynomial equations. *Journal of Symbolic Computation 44(3)*, pp. 292–306, March 2009.
2. Farin G. Curves and Surfaces for CAGD: A Practical Guide. *Academic Press Professional, Inc*, San Diego, CA, 1988.
3. Garloff J., and Smith A.P. Investigation of a subdivision based algorithm for solving systems of polynomial equations. *Journal of Nonlinear Analysis : Series A Theory and Methods 47*, 1 (2001), pp. 167–178.
4. Elber G., and Kim M.-S. Geometric constraint solver using multivariate rational spline functions. In *SMA'01: Proc. of the 6th ACM Symp. on Solid Modeling and Applications* (New York, NY, USA, 2001), ACM Press, pp. 1–10.
5. Sherbrooke E.C., and Patrikalakis N.M. Computation of the solutions of nonlinear polynomial systems. *Computer Aided Geometric Design 5(10)*, pp. 379–405, 1993.
6. Chvatal, V. Linear Programming. *Series of Books in the Mathematical Sciences, W.H. Freeman*, September 1983.
7. Wunderling R. SoPlex – The sequential object-oriented simplex. *Konrad-Zuse-Zentrum für Informationstechnik Berlin, Department Optimization*, http://soplex.zib.de/, Version 1.4.2, 2009.
8. Dhiflaoui M., Funke S., Kwappik C., Mehlhorn K., Seel M., Schömer E., Schulte R., and Weber D. Certifying and Repairing Solutions to Large LPs, How Good are LP-solvers? In *SODA*, pp. 255-256, 2003.
9. Lin Q., and Rokne J.G. Methods for bounding the range of a polynomial. *Journal of Computational and Applied Mathematics 58*, pp. 193-199, 1995.
10. Lin Q., and Rokne J.G. Interval approximation of higher order to the ranges of functions. *Computers & Mathematics with Applications 31(7)*, pp. 101-109, 1996.
11. Golub G.H., and Van Loan C.F. Matrix Computations. *Johns Hopkins Studies in Mathematical Sciences*, 3rd edition, Baltimore, Maryland, 1996.
12. Gough V.E. Contribution to discussion of papers on research in Automobile Stability, Control and Tyre performance. *Proc. Auto Div. Inst. Mech. Eng.*, pp. 392-394, 1956-1957.
13. Merlet J.P., Solving the Forward Kinematics of a Gough-Type Parallel Manipulator with Interval Analysis. *Int. Journal Robotic Research 23(3)*, pp. 221–235, 2004.

14. Charentus S., Diaz C., and Renaud M. Modular serial parallel redundant robot. In *IMACS*, Cetraro, Italy, September 18-21, 1988.
15. Michelucci D., and Foufou S., Using Cayley-Menger determinants for geometric constraint solving. *SM '04: Proceedings of the ninth ACM Symposium on Solid Modeling and Applications*, pp. 285–290, Genoa, Italy, 2004.